Ranking for Data Repairs

Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville

Purdue University, West Lafayette, IN 47907, USA
{myakout, ake, neville}@cs.purdue.edu

***Abstract*— Improving data quality is a time-consuming, labor-intensive and often domain specific operation. A recent principled approach for repairing dirty database is to use data quality rules in the form of database constraints to identify dirty tuples and then use the rules to derive data repairs. Most of existing data repair approaches focus on providing fully automated solutions, which could be risky to depend upon especially for critical data. To guarantee the optimal quality repairs applied to the database, users should be involved to confirm each repair. This highlights the need for an interactive approach that combines the best of both; automatically generating repairs, while efficiently employing user's efforts to verify the repairs. In such approach, the user will guide an online repairing process to incrementally generate repairs. A key challenge in this approach is the response time within the user's interactive sessions, because the process of generating the repairs is time consuming due to the large search space of possible repairs. To this end, we present in this paper a mechanism to continuously generate repairs only to the current top k *important violated* data quality rules. Moreover, the repairs are grouped and ranked such that the most *beneficial* in terms of improving data quality comes first to consult the user for verification and feedback. Our experiments on real-world dataset demonstrate the effectiveness of our ranking mechanism to provide a fast response time for the user while improving the data quality as quickly as possible.**

I. Introduction

There is unanimous agreement among researchers and practitioners alike on the importance of data quality for many real world applications and the unimaginable consequences of making decisions based on inconsistent, inaccurate or incomplete data. Poor data quality is a fact of life for most organizations [1]. For example, poor data quality in retail databases alone costs US consumers \$2.5 billion annually [2]. Not to mention the importance of data quality in the healthcare domain. In such critical applications, incorrect information about patients in an Electronic Health Record (EHR) may lead to wrong treatments and prescriptions, which consequently may cause severe medical problems including death.

A general and recent domain independent approach for improving data quality is to (i) discover and identify some data quality rules (DQRs), and then, (ii) use these rules to derive data repairs for dirty instances that violate these rules. Various techniques have followed this approach for *automatic* data repairs, (e.g., [3], [4], [5], [6], [7], [8], [9]). However, in real-world scenario, this is not sufficient and domain users have to verify the applied repairs, especially, when in critical domains like healthcare.

Automatic repair techniques usually employ heuristics by defining a cost model (or evaluation function) to greedily select the best repairs among the many possible repairs for a dirty tuple violating the rules. In most cases, the cost model prefers repairs that minimally change the original data. For example, choosing values that are close in distance to the dirty values. However, inconsistent values could be completely different, i.e., not close in distance. It is also usually assumed that the values derived directly from a pre-specified cleaning rule are always correct. For example, if the rule is "zip=47906 → state=IN" and a dirty tuple contains "zip=47906, state=IL", then the repair is to change the state to IN. However, it could be the case that the zip code is actually wrong. Consequently, domain users will have to be involved in interactively monitoring the repair process [4].

One of the key challenges in an interactive data repair system is to provide a fast response time during the user interactions. More specifically, we need to determine *how* to explore the repair search space to generate repairs. This will require developing a set of principled measures to estimate the improvement in quality to reason about these tasks. This should help achieve a good trade-off between high quality, responsive system and minimal user involvement, which are top priorities of an interactive data cleaning approach.

In this paper, we present a ranking mechanism for the DQRs to explore the search space with the objective of satisfying only the top k rules. This will help reducing the response time spent in each interactive repairing session to search for repairs. Moreover, we summarize a ranking mechanism for groups of repairs to provides the most beneficial repairs to the user first. We show empirically that by combining both rules ranking with repair group ranking, we can achieve fast convergence to better data quality with minimal user efforts.

Example

We consider the relation `Person`(`Name`, `SRC`, `STR`, `CT`, `STT`, `ZIP`), which specify personal address information Street (`STR`), City (`CT`), State (`STT`) and (`ZIP`), in addition to the source (`SRC`) of the data or the identity of the data entry operator. An instance of this relation is shown in Fig. I.

Data quality rules can be defined in the form of conditional functional dependencies (CFDs) as described in Fig. I(b). A CFD is a pair consisting of embedded standard functional dependance (FD) and a pattern tableau. For example, $\phi_1 - \phi_3$ state that the FD $\texttt{ZIP} \rightarrow \texttt{CT}, \texttt{STT}$ (i.e., zip codes uniquely identify city and state) holds in the *context* where the `ZIP` is 46360, 46774 or 46825[1]. Moreover, the pattern tableau enforces bindings between the attribute values, e.g., if `ZIP`= 46360, then `CT`= Michigan City. ϕ_5 states that the FD $\texttt{STR}, \texttt{CT} \rightarrow \texttt{ZIP}$ holds in the context where `CT` = Fort Wayne, i.e., street names uniquely identify the zip codes whenever the city is Fort Wayne. Note that all the tuples have violations.

Typically, a repair algorithm will use the rules and the current database instance to find the best possible repairs to satisfy the violated rules. For example, t_5 violates ϕ_4 and a possible repair

[1]Some zip codes in the US may span more than one city

	Name	SRC	STR	CT	STT	ZIP
t1:	Jim	H1	REDWOOD DR	MICHIGAN CITY	MI	46360
t2:	Tom	H2	REDWOOD DR	WESTVILLE	IN	46360
t3:	Jeff	H2	BIRCH PARKWAY	WESTVILLE	IN	46360
t4:	Rick	H2	BIRCH PARKWAY	WESTVILLE	IN	46360
t5:	Joe	H1	BELL AVENUE	FORT WAYNE	IN	46391
t6:	Mark	H1	BELL AVENUE	FORT WAYNE	IN	46825
t7:	Cady	H2	SHERDEN RD	FT WAYNE	IN	46774

(a) Data

$\phi_1 : (\texttt{ZIP} \rightarrow \texttt{CT}, \texttt{STT}, \{46360 \ \| \ \text{MichiganCity}, \text{IN}\})$
$\phi_2 : (\texttt{ZIP} \rightarrow \texttt{CT}, \texttt{STT}, \{46774 \ \| \ \text{NewHaven}, \text{IN}\})$
$\phi_3 : (\texttt{ZIP} \rightarrow \texttt{CT}, \texttt{STT}, \{46825 \ \| \ \text{FortWayne}, \text{IN}\})$
$\phi_4 : (\texttt{ZIP} \rightarrow \texttt{CT}, \texttt{STT}, \{46391 \ \| \ \text{Westville}, \text{IN}\})$

$\phi_5 : (\texttt{STR}, \texttt{CT} \rightarrow \texttt{ZIP}, \{ \_ \ , \text{FortWayne} \ \| \ \_ \})$

(b) CFD Rules

Fig. 1. Example data and rules

would be to either replace CT by Westville or ZIP by 46825 which would fall in the context of ϕ_3 but without violations.

Then a question is raised; do we need to generate the possible repairs for all the dirty tuples? This will be time consuming for large databases. Assuming that the persons leaving in zip codes 46360 are currently more important for the business and their information should be more accurate than the others. Then, we can limit the repair generation mechanism to suggest repairs for ϕ_1 since it is more valuable to the business and it is highly violated. Focusing on repairing the violations of ϕ_1 will improve the quality of an important portion of the data, and at the same time, this will reduce the response time spent in searching for repairs for the first four tuples instead of finding repairs for all the tuples. This motivates our rules ranking mechanism to guide the repairing process and provide a responsive interactive repairing system.

To involve the user in guiding the repair process, it is important to provide the repairs into groups that can be easily handled by users, e.g., displaying a group of dirty tuples where the CT is suggested to be Michigan City. From our experience, this grouping proved to be highly effective in manually verifying large number of repairs quickly. Moreover, assuming that the repair mechanism suggests for repairing t_2, t_3, and t_4 that CT be replaced with Michigan City (which is a correct value for t_2, t_3 but incorrect for t_4), and suggests for repairing t_3 and t_4 that ZIP be replaced by 46391 (which are both incorrect repairs). It will be useful to consult the user for the first group because (i) the repairs are more likely to be correct, and (ii) the high number of correct repairs will allow for fast convergence to a cleaner database. For the second group, we will not get such impact.□

The rest of the paper has three main parts: The first part is Section II, which provides guidelines for our interactive repairing framework. The second part is Section III, which covers the main novelty of our framework, namely the mechanisms for ranking the DQRs and suggested repairs. The third part is a discussion of our experimental evaluation in Section IV. We wrap up and conclude the paper in Section V.

II. Framework

In this section, we illustrate our interactive repairing framework that combines together an automatic repairing technique with user involvement. Procedure 1 provides an outline of the main steps of the framework.

Procedure 1 Interactive_Repair(D dirty database, Σ DQRs)

```
1: Identify the dirty tuples in $D$ using $\Sigma$.
2: repeat
3:    Identify the top k important violated rules $\Sigma_k$.
4:    $PossibleRepairs \leftarrow$ Generate initial suggested repairs for the tuples
      violating $\Sigma_k$.
5:    while $PossibleRepairs$ is not empty and the user is available do
6:       Rank groups of repairs such that the most beneficial come first and
         the user selects group $c$ from the top.
7:       User interactively gives feedback on suggested repairs in $c$
8:       Confirmed repairs are applied to the database.
9:       Replace rejected repairs in $PossibleRepairs$ as needed.
10:      Identify emerged dirty tuple and generate repairs as necessary.
11:   end while
12: until $Dirty_Tuples$ is empty or user is not available
```

The primary input to the framework is a relational database instance D, which is dirty. The second input is a set of DQRs Σ that represent data integrity semantics. DQRs are a set of constraints that need to be satisfied by all database tuples in order for the database to be considered of high quality. Examples of such rules include general integrity constraints, standard functional and inclusion dependencies (FDs and INDs resp.) as well as their recent extension conditional functional dependencies (CFDs) proposed in [10], and conditional inclusion dependencies (CINDs) [11]. Another type is Dedupalog [12] rules, which are used to eliminate duplicates. For example, in Dedupalog, $\texttt{Paper}\star(id, id') \leftrightarrow \texttt{TitleSimilar}(t, t')$ means that papers with similar titles are likely to be duplicate.

In Procedure 1, after identifying the dirty tuples which violate the DQRs in step 1, our framework provides two types of guidance: (i) *guide* the automatic repairing mechanism to focus on discovering repairs for the most important violated DQRs, and consequently, improve the system response time in each repairing session. (ii) *guide* the user to focus his/her efforts on verifying repairs that would improve the quality faster. This guidance proceeds in a continuous feedback process in the steps 2-12, while there are dirty tuples and the user is available and willing to give feedback.

Particularly in step 3, the system picks the top k important violated rules Σ_k. This is followed by automatically generating suggested repairs in step 4 for the tuples that violate only Σ_k. This way we reduce the time spent in exhaustively searching the whole space of repairs for all the dirty tuples and provides a responsive system. The generated repairs are stored in $PossibleRepairs$ list.

Afterward, a user interactive session starts with each iteration of the loop in steps 5-11. In each session, the repairs are grouped and ranked (step 6), such that the most important groups comes first, and the user starts to work on the first group (c) in the list. (Selecting the top k rules and ranking the repair groups is discussed in Section III.) The user then provides feedback in step 7 (e.g. confirm, reject, or provide new repairs) on the suggested repairs. Confirmed repairs are immediately applied to the database in step 8, and for rejected repairs, the system incrementally provide alternative repairs, as needed (step 9). Since applied repairs will change the database state, this will require, in step 10, identifying new emerged violations to the DQRs due to this change.

III. Guided Repair

The key challenge to our framework hinges in steps 3 and 6 of Procedure 1. In this section, we address the problem of quantifying the *importance* of satisfying a data quality rule, as well as, the *benefit* from a group of suggested repairs to the data quality. This will help us devising a scoring mechanism to rank the DQRs (step 3) to focus on satisfying the top k rules when generating repairs. Moreover, providing a ranking to the groups of repairs (step 6) for user feedback according to its anticipated benefit to the data quality. Both of these ranking mechanisms will help in involving the user to verify the repairs that would improve the data quality most, while providing a good response time for the user between repairing sessions.

Before we proceed, we need to quantify the rule violations by introducing the concept of a *database violation* as follows:

Definition 1: Database Violation to DQR ϕ: Given a database D and a DQR ϕ, a database violation with respect to ϕ, denoted by $vio(D, \{\phi\})$, is defined as the number of tuples violating ϕ, $vio(D, \{\phi\}) = |\{t \mid t \not\models \phi \wedge t \in D\}|$, where $t \not\models \phi$ denotes that t violates ϕ.

Consequently, the total database violations for D with respect to the set of DQRs Σ is:

$$vio(D, \Sigma) = \sum_{\phi \in \Sigma} vio(D, \{\phi\}).$$

A. Guided Generation For Data Repairs

In the following, we propose an approach to *guide* the automatic generation of data repairs. This approach provides a mechanism for exploring the data repairs by *prioritizing* the repairs search space for the sake of having a responsive interactive system. The automatic generation of data repairs is a time consuming process due to the large search space to find the best repairs, even when greedy heuristics is applied. In our repairing framework, instead of greedily deciding upon data repairs, the user is involved to provide the necessary decisions upon the repairs. If the framework starts by generating possible repairs for all the identified dirty tuples, the user will have to wait long time for system to initially respond. Moreover in later user sessions, rejected repairs will require generation for other repairs leading to more waiting time from the user.

To start with, we need first to define a *data quality loss* function, which will be the objective to be minimized while exploring the repairs search space, accordingly, we show how to quantify the importance of a rule to be satisfied.

Data Quality Loss: We define the data quality loss, $L(D)$, as inversely proportional to the degree of satisfaction of the specified DQRs Σ. To compute $L(D)$, we first need to measure the *quality loss* with respect to a DQR $\phi_i \in \Sigma$, namely $ql(D|\phi_i)$. Assuming that D^{opt} is the clean optimal database instance, where all the DQRs Σ are satisfied, we can express ql by:

$$ql(D|\phi_i) = 1 - \frac{\models (D, \phi_i)}{\models (D^{opt}, \phi_i)} = \frac{\models (D^{opt}, \phi_i) - \models (D, \phi_i)}{\models (D^{opt}, \phi_i)} \tag{1}$$

where $\models (D, \phi_i)$ and $\models (D^{opt}, \phi_i)$ are the numbers of tuples satisfying the rule ϕ_i in the current database instance D and D^{opt}, respectively. Consequently, the data quality loss can be computed as follows:

$$L(D) = \sum_{\phi_i \in \Sigma} ql(D|\phi_i) \times w_i. \tag{2}$$

where w_i is a weight to express the business value for the quality of the database of satisfying the rule ϕ_i. These weights are user defined parameters. In our experiments, we used the values $w_i = \frac{|D(\phi_i)|}{|D|}$, where $|D(\phi_i)|$ is the number of tuples that fall in the context of the rule ϕ_i (e.g., in Figure 1, t_5 and t_6 fall in the context of ϕ_5 and t_1-t_4 fall in the context of ϕ_1). The intuition is that the more tuples fall in the context of a rule, the more valuable it is to satisfy this rule. Note that $|D(\phi_i)|$ covers tuples that both satisfy and violate ϕ_i.

Concerning the numerator expression in Eq. 1, namely the difference between the numbers of the tuples satisfying ϕ_i in D^{opt} and D, respectively. This quantity can be approximated using the number of tuples violating ϕ_i. Accordingly, we use the expression $vio(D, \{\phi_i\})$ (cf. Definition 1) as the numerator in Eq. 1. Moreover, we can approximate D^{opt} by D as it is not expected to have most of the data dirty. This will lead to the expected data quality loss as follows:

$$E[L(D)] = \sum_{\phi_i \in \Sigma} \frac{vio(D, \{\phi_i\}) \times w_i}{\models (D, \phi_i)}. \tag{3}$$

Consequently, we can assign each rule ϕ_i an importance score as follows:

$$I(\phi_i) = \frac{vio(D, \{\phi_i\}) \times w_i}{\models (D, \phi_i)}. \tag{4}$$

Satisfying the rules ϕ_i with the highest values of $I(\phi_i)$, would improve the quality most according to Eq. 3. Therefore, we can guide the repairing mechanism to focus on finding repairs only for the *currently* top k rules with the highest I values.

B. Ranking Repair Groups

In the following, we discuss how to best present repairs to the user, in a way that will provide the *most benefit* for improving the quality of the data.

At any iteration of the process outlined in the steps 2-11 of Procedure 1, there will be several possible suggested repairs (in $PossibleRepairs$ list) to forward to the user for feedback. These repairs are grouped into groups $\{c_1, c_2 \ldots\}$ according to a grouping function. This function is selected in a way that exposes the structure of data relationships with the objective of providing a *useful-looking* set of repairs to be easier for the user to handle and process. Example of grouping functions include grouping repairs for tuples that have the same value in a given attribute (e.g., age, city) and grouping repairs that suggest the same value for attribute A (we used the later one for our experiments). Similar grouping ideas have been explored in [13], [14].

The formula to compute the data quality benefit to database D of consulting the user for the repair group c can be written as follows:

$$E[g(c)] = \sum_{\phi_i \in \Sigma} w_i \sum_{r_j \in c} s_j \frac{vio(D, \{\phi_i\}) - vio(D^{r_j}, \{\phi_i\})}{\models (D^{r_j}, \phi_i)} \tag{5}$$

where $r_j \in c$ is a currently suggested repair in the system within the group c, $s_j \in [0, 1]$ is a score attached to r_j

resulted from the repairing mechanism using a repair evaluation function to represent the mechanism *certainty* of the repair r_j, the database instance resulting from applying the repair r_j is denoted by D^{r_j}, $\models (D^{r_j}, \phi_i)$ is the number of tuples satisfying the DQR ϕ_i in the database instance D^{r_j}, and w_i is a weight to express how much valuable ϕ_i for the data domain to be applied on the data.

To principally reason about the above benefit formulation in Eq. 5, we developed a decision theoretic approach based on the *value of information* [15] concept. The derivation details are provided in a paper that is currently under submission. However, the above formulation can be intuitively justified by the following.

The main objective to improve the quality is to reduce the database violations. Therefore, the difference in the amount of database violations as defined in Definition 1, before and after applying r_j, is a major component to compute the repair benefit. This component is computed, under the first summation, for every DQR ϕ_i as a fraction of the number of tuples that would be satisfying ϕ_i, if r_j is applied. Since the database does not know the correctness of the repair r_j, we can not use the term under the first summation as a final benefit score. Instead, we compute the expected repair benefit by approximating our *certainty* about the benefit by using s_j, which is a repair evaluation score for r_j. s_j is the score resulted from the repair evaluation function used in the repairing mechanism. Finally, it is usually expected for the DQRs to not have same (business or sematic) value in the database. There are some rules that are more valuable to satisfy than the others. Therefore, the overall expected benefit will be scaled using the rule's weights w_i.

IV. Experiments

In this section, we present an evaluation for our framework. Specifically, the objectives of the experimental study are as follows:

- Study the the reduction in the average response time within the user's consultation sessions because of the rules ranking mechanism.
- Demonstrate the fast convergence to a better data quality due to the repair groups ranking mechanism and the effect of the rules ranking, as well on this convergence.

Framework Implementation. In our implementation to the framework, we used the CFDs as the data quality rules. CFDs are an extension to the standard FDs and have proved to be more effective in catching data inconsistencies than FDs [10]. Several efforts were triggered [16], [17], [18], [19] to facilitate their automatic discovery. We implemented a repair generation process that is inspired by the technique described in [7]. However, we extended this technique to find repairs for each attribute of the dirty tuples separately given the current database state. More precisely, given a dirty tuple t, suggesting a new value for $t[A]$ is independent from any other previously discovered repair, unless it was confirmed by the user. The work in [7] uses a greedy search for repairs that may depend on already found repairs, which could themselves be incorrect.

Datasets. In our experiments, we used two datasets, denoted as Dataset 1 and 2 respectively. Dataset 1 is a real world dataset resulting from the integration of (anonymized) emergency room visits from 74 hospitals. Such patient data is used to monitor naturally occurring disease outbreaks, biological attacks, and chemical attacks. Since such data is coming from several sources, a myriad of data quality issues arise due to the different information systems used by these hospitals and the different data entry operators responsible for entering this data. For our experiments, we selected a subset of the available patient attributes, namely Patient ID, Age, Sex, Classification, Complaint, HospitalName, StreetAddress, City, Zip, State, and VisitDate. For Dataset 2, we used the adult dataset from the UCI repository[2]. For our experiments, we used the attributes education, hours_per_week, income, marital_status, native_country, occupation, race, relationship, sex, and workclass.

Ground truth. To evaluate the quality of our technique against a ground-truth, we manually repaired 20,000 patient records in Dataset 1. We used address and zip code lookup web sites for this purpose. We assumed that Dataset 2 is already clean and hence can be used as our ground truth. We synthetically introduced errors in the attribute values as follows. We randomly picked a set of tuples, and then for each tuple, we randomly picked a subset of the attributes to perturb by either changing characters or replacing the attribute value with another value from the domain attribute values. All experiments are reported when 30% of the tuples are dirty.

Data Quality Rules. For dataset 1, we used CFDs similar to what was illustrated in Figure I. The rules were identified while manually repairing the tuples. For Dataset 2, we identified 8 rules that include the attributes workclass, occupation, relationship, sex, education, marital_status, hours_per_week and income.

Settings. All the experiments were conducted on a server with a 3 GHz processor and 32 GB RAM running on Linux. We used Java to implement the proposed techniques and MySQL to store and query the records.

User interaction simulation. User feedback to suggested repairs was simulated by providing answers as determined by the ground truth datasets.

Data quality state metric. We measure the database quality using the data quality loss (Eq. 2). For Dataset 1, we used the manually repaired database as the desired optimal database, and for Dataset 2, we used the original adult dataset as the optimal one.

A. Improving the Average Response Time Using Rules Ranking

In this experiment, we evaluate the rules ranking mechanism described in Section III-A to guide the generation of data repairs and reducing the system response time. More precisely, we study the system response time through the user interactions when using the ranked top k rules for several values of k.

In Fig. 2 for the two datasets, we report the overall average response times over all the sessions required to clean all the database against the value of k. We compare the result to the same technique when ignoring the rules ranking.

For Dataset 1, there is an overall 2 to 3 order of magnitude decrease in the response time when ignoring the ranking. Also, with the increase of k the average response time is slightly increasing. This is justified by increasing the number of rules to be satisfied, and consequently, increasing the scope of the search space to be explored to generate more repairs. For Dataset 2, the property of increasing the average response time

[2]http://archive.ics.uci.edu/ml/

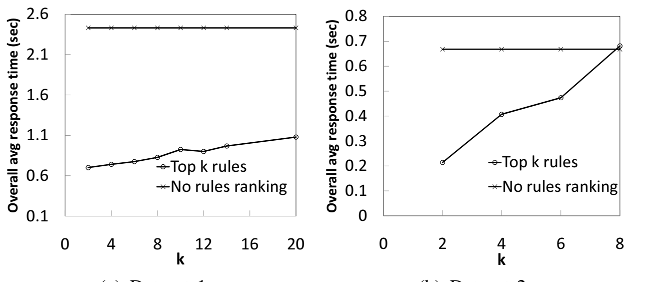


Fig. 2. Comparing the overall average response time when using the rules ranking (top k) and when ignoring the rules ranking. The rules ranking is effective in reducing response time.

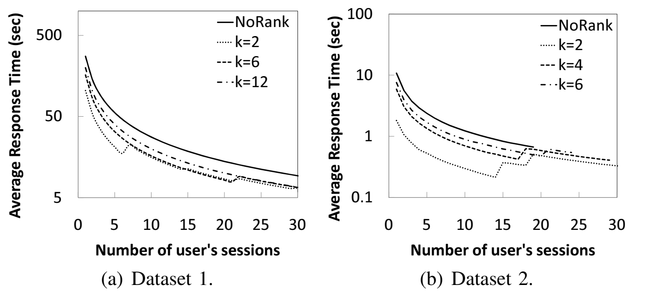


Fig. 3. Comparing the average response time through the user's interactive sessions. The use of the rules ranking with different k is always maintaining less response time.

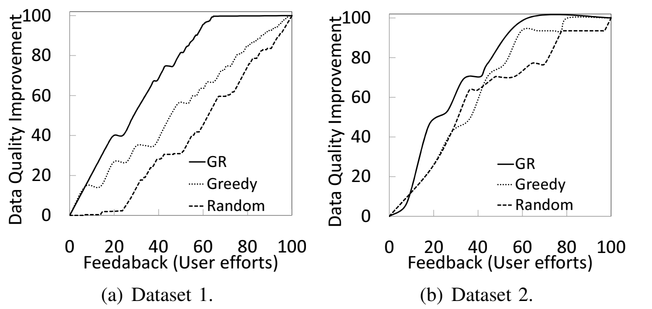


Fig. 4. Comparing three techniques (GR, Greedy, and Random) for ranking the generated repair groups. Feedback is reported as the percentage of the maximum number of verified repairs required by a technique. GR shows superior performance compared to other naïve ranking strategies.

with the increase of k still hold. However, the the use of the top k technique quickly perform similar to the case of ignoring the ranking when $k = 8$. This is because we used only 8 rules for repairing. In the case of Dataset 1, we used about 200 rules and this give more chance to demonstrate the effectiveness of using the top k rules mechanism.

To further get more insights about the response time behavior through the iterations, we reported in Fig. 3 the average response time achieved so far after each iteration for the first 30 interactive sessions. Particularly, after session i, we sum all the user waiting time so far and divide it by i, for $i = 1, \ldots, 30$. This is reported when using the rules ranking technique for $k = 2, 6, 12$, as well as, for the case when there is no ranking for the rules.

In Fig. 3 for all the reported curves, the average response time starts high in the first few sessions and then quickly decreases in later sessions. However, the techniques that uses the rules ranking maintain always lower response time compared with the technique without rules ranking. As we decrease k, we will have lower curve and better response time.

Concluding the above experiment, our rules ranking technique helps the repairing mechanism in reducing the repairs search space by focusing on finding repairs for those tuples that violate only the top k important violated rules. This resulted in having an a overall better response time than the case without the rules ranking as well as always maintaining better response time through all the repairing interactive time with the user.

B. The Effect of the Rules Ranking on the Progress of Improving the Data Quality

In the following, we study the impact of the rules ranking mechanism on the progress of improving the data quality as we acquire user feedback for data repairs.

Ranking Groups of Repairs: First, we empirically prove that ranking the groups of repairs according to Eq. 5 for user feedback provides the best convergence to better data quality as the system receives user feedback. Fig. 4 provides a comparison between two intuitive techniques (namely *Greedy* and *Random*) for ranking the groups of repairs in addition to our technique described in Section III-B, denoted as GR (Guided Repair).

In the *Greedy* ranking, we rank the repair groups according to their sizes. The rationale behind this strategy is that groups that cover larger numbers of repairs may have high impact on the database quality if most of the suggestions within them are correct. The *Random* ranking is the naïve strategy where we randomly order the repair groups; all repair groups are equally important.

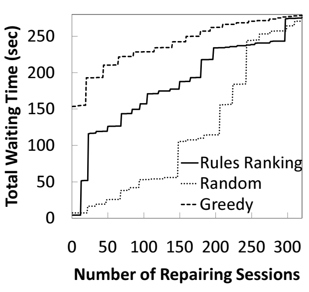
As illustrated for both datasets, the GR approach performs well compared to the *Greedy* and *Random* approaches. This is because the *GR* approach perfectly identifies the most beneficial repair groups that are more likely to have correct repairs. While the *Greedy* approach improves the quality, most of the content of the repair groups is sometimes incorrect leading to wasted user efforts. The *Random* approach showed the worst performance in Dataset 1, while for Dataset 2, it was comparable with the *Greedy* approach especially in the beginning of the curves. This is because in Dataset 2, most of the sizes of the repair groups were close to each others making the *Random* and *Greedy* approaches behave almost identically, while in Dataset 1 the groups sizes varies widely making the random choices ineffective.

The results reported above justify clearly the importance and effectiveness of the GR ranking. The GR approach is well suited for repairing "very" critical data, where every suggested repair has to be verified before applying it to the database.

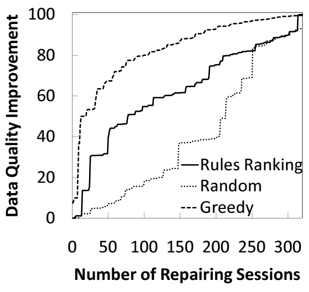
Ranking Rules To Generate Repairs: In the following experiment, we used the GR ranking mechanism to rank the repairs generated when using the rules ranking mechanism. Recall that the rules ranking guide the repair generation to generate repairs for only the tuples that violates the most important violated rules identified by the highest score, I, in Eq. 4.

First, we compare our rules ranking mechanism with other strategies: (i) *Greedy*, which rank rules according the amount of violations, i.e. using $vio(D, \{\phi\})$. The intuition here is to highly rank rules that are most violated. (ii) *Random*, which treats all the rules equally and randomly pick k rules at a time. We used $k = 5$ for this experiment.

We report the evaluation on Dataset 1in Fig. 5. Fig. 5(a)

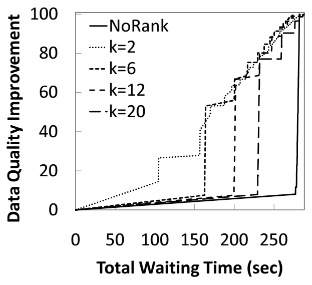


(a) Comparing the total waiting time through the sessions.

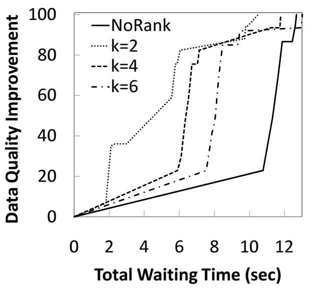


(b) Comparing the improvement in data quality through the sessions.

Fig. 5. Comparing three techniques (Rules Ranking, Greedy, and Random) for ranking rules to guide repairs generation. Our Rules Ranking mechanism provides the best trade-off between the total waiting time and improving the quality.



(a) Dataset 1.



(b) Dataset 2.

Fig. 6. Reporting the progress in data quality improvement vs. the total waiting time during the sessions for different values of k. The ranking is effective in achieving better quality with less waiting times between the sessions. Increasing k, increases the waiting time because of increasing the repairs to be search for the dirty tuples.

shows the total waiting time through the repairing sessions and Fig. 5(b) shows the progress in improving the quality through the seesions as well. The *Greedy* is the best to improve the quality over the sessions, but it is the worst in the required waiting time. Conversely, the *Random* required the least waiting time, while it is the worst in improving the quality. Our rules ranking mechanism provides a performance that is in between.

The *Greedy* approach picks the rules that are most violated, hence more dirty tuple will need repairs after a session, consequently, more waiting time is needed. Moreover, when repairing many tuples that violate the same rule, most likely, the repairs would be related and grouped together in few sessions. This situation and argument are reversed for the *Random*. To this end, our rules ranking provide the best trade-off between these approaches.

In Fig. 6, we study the effect of k on improving the data quality and the waiting time. It is noted that the rules ranking is helpful in the beginning of the curves. As we increase k, the waiting time is increasing.

The rules ranking technique affects the currently suggested repairs in the $PossibleRepair$ list. This will affect the computation of the group benefit in Eq. 5, and consequently, change the repairs presented to the user for verification. Therefore, for the initial iterations, the rules ranking (for different values of k) is more responsive than for the case of NoRank.

In conclusion, the rules ranking and using the top k rules technique provide the best trade-off between faster conversion to better data quality and faster response time within the interactive sessions. This trade-off is controlled by k, lower values of k makes the system responsive, while higher values slow down the quality improvement.

V. Conclusion and Future Work

We presented guidelines for an interactive data repair framework. The objective is to leverage automatic repairing techniques alongside with user feedback in a responsive system to monitor and improve the data quality as quickly as possible while guaranteeing optimal repairs to the data. The framework presented assumes that all repairs should verified by the user, however in our current work, we have already involved machine learning algorithms to minimize user's efforts.

The work presented is part of bigger research project aiming at tackling the problem of improving data quality from a more pragmatic and practical point of view. Our approach is to provide techniques for *guided data quality improvement*. The key idea is to *efficiently* and *effectively* involve the user in guiding the automatic mechanisms for data quality tasks (e.g. repairing data, discovering data quality rules), instead of having the automatic techniques work and produce their results which would be far more than one can expect the user to comment on. To this end, we are developing principled measures to reason about these tasks, as well as, involving machine learning algorithms to learn and minimize user interactions.

References

[1] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*. Addison-Wesley.
[2] L. English, "Information quality management: The next frontier," *Information Management Magazine*, 2000.
[3] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*, 1999.
[4] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *ACM SIGMOD*, 2005.
[5] R. Bruni and A. Sassano, "Errors detection and correction in large scale data collecting," in *IDA*, 2001.
[6] J. Chomicki and J. Marcinkowski, "Minimal-change integrity maintenance using tuple deletions," in *Information and Computation*, 2005.
[7] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: consistency and accuracy," in *VLDB*, 2007.
[8] E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello, "Census data repair: a challenging application of disjunctive logic programming," in *LPAR*, 2001.
[9] A. Lopatenko and L. Bravo, "Efficient approximation algorithms for repairing inconsistent databases," in *ICDE*, 2007.
[10] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *ICDE*, 2007.
[11] L. Bravo, W. Fan, and S. Ma, "Extending dependencies with conditions," in *VLDB*, 2007.
[12] A. Arasu, C. Re, and D. Suciu, "Large-scale deduplication with constraints using dedupalog," in *ICDE'09*, 2009.
[13] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *ACM SIGKDD*, 2002.
[14] M. A. Hernndez and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," in *Data Mining and Knowledge Discovery*, 1998.
[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*.
[16] F. Chiang and R. J. Miller, "Discovering data quality rules," in *VLDB*, 2008.
[17] W. Fan, F. Geerts, L. V. Lakshmanan, and M. Xiong, "Discovering conditional functional dependencies," in *ICDE*, 2009.
[18] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "Increasing the expressivity of conditional functional dependencies without extra complexity," in *VLDB*, 2008.
[19] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the confidence of conditional functional dependencies," in *ACM SIGMOD*, 2009.