CrowdAidRepair: A Crowd-Aided Interactive Data Repairing Method

Jian Zhou[†], Zhixu Li[†⋆], Binbin Gu[†], Jia Zhu[‡],
An Liu[†], Guanfeng Liu[†], Pengpeng Zhao[†], and Lei Zhao[†]

[†] School of Computer Science and Technology, Soochow University, China
[‡] School of Computer Science, South China Normal University, China
{jzhou_jz,gu.binbin}@hotmail.com
{zhixuli,anliu,gfliu,ppzhao,zhaol}@suda.edu.cn,jzhu@m.scnu.edu.cn

Abstract. Data repairing aims at discovering and correcting erroneous data in databases. Traditional methods relying on predefined quality rules (e.g., FD/CFDs) to detect the conflict between data may fail to choose the right way to fix the detected conflict. Recent efforts turn to use the power of crowd in data repairing, but the crowd power has its own drawbacks such as high human intervention cost and inevitable low efficiency. In this paper, we propose a crowd-aided interactive data repairing method which takes the advantages of both rule-based method and crowd-based method. Particularly, we investigate the interaction between crowd-based repairing and rule-based repairing, and show that by doing crowd-based repairing to a small portion of values, we can greatly improve the repairing quality of the rule-based repairing method. Although we prove that the optimal interaction scheme using the least number of values for crowd-based repairing to maximize the imputation recall is not feasible to be achieved, still, our proposed solution identifies an efficient scheme through investigating the inconsistencies and the dependencies between values in the repairing process. Our empirical study on three data collections demonstrates the high repairing quality of CrowdAidRepair, as well as the efficiency of the generated interaction scheme over baselines.

1 Introduction

Data repairing aims at discovering and correcting erroneous data in databases. So far, various data repairing solutions have been developed to automatically detect and repair erroneous data in databases [13]. The main stream of rule-based solutions [2, 8, 9] rely on a variety of quality rules such as FD/CFDs [1, 4, 12] to detect violations and conflicts between data. By resolving these violations and conflicts, they expect to fix the erroneous data. However, without having the background knowledge, the existing rule-based method just follows some simple modification-strategy (such as minimum-modification) to make modifications [2, 9], which as a result, may produce more errors.

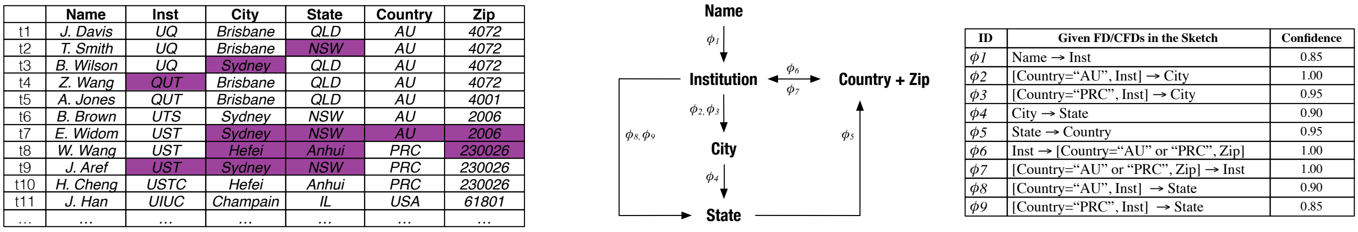
[⋆] The corresponding author

Recent efforts use the power of Crowd for data repairing, which let the crowd to help make right modification decisions according to predefined quality rules. Basically, these crowd-based methods can effectively improve the quality of the data after the repairing. For instance, Yakout et. al. [14] use user's feedback to repair a database and to adaptively refine the training set for a repairing model. However, dislike using rules, no repairing method can solve all the conflicts with one single repairing model. On the other hand, the NADEEF system [3] allows the users to specify data quality rules and how to repair it through writing code that implements predefined classes. However, although some efforts are made, it still requires high labor cost for data repairing. In addition, any methods relying on humans can not be very efficient since humans need to take rest anyway.

In this paper, we propose a novel combined repairing method, CrowdAidRepair, which performs crowd-based repairing and rule-based repairing alternatively for achieving a high repairing quality at the minimum crowd cost. Specifically, we still rely on FD/CFDs to identify conflicts between values, but we do rule-based repairing to a conflict only when this repairing operation can satisfy a predefined quality constraint. When no more conflict can be repaired by the rules, we select some values for crowd-based repairing to let more values be repairable to rule-based method. We continue with this interactive repairing process iteratively until no more values can be modified. To this end, CrowdAidRepair faces a challenge of selecting the least number of values for crowd-based repairing to maximize the number of values for rule-based repairing. Ideally, an optimal interaction scheme minimizes the number of issued crowd-based repairing operations for resolving the detected conflicts correctly.

This scheduling problem for the interaction is nontrivial: Primarily, to reach the minimum crowd cost, we hope to do crowd-based repairing only to those conflicts that can never be resolvable to rule-based repairing. However, we do not know a priori which conflicts can never be resolved correctly by rule-based repairing until all the other conflicts are resolved. Furthermore, the whole interaction issue is considered in a dynamic setting. As more and more conflicts are resolved, the rule-based repairing result to every unresolved conflict might be changed, and the set of unresolved conflicts will also be changed as some new conflicts will be generated while some old ones will be solved/dismissed.

We analyze in theory that the optimal interaction scheme is not feasible to be achieved, and thus we propose our alternative algorithm that can generate an efficient scheme for the interaction between rule-based repairing and crowd-based repairing. In particular, we investigate the inconsistency that each value brings to the database, according to which we estimate a *disharmonious score* for each value. We will justify that a value with a higher disharmonious score should have a high priority to be checked with the crowd. Besides, although we are in a dynamic setting to schedule conflicts and their covered values for repairing, we can still fix something based on the dependency relations between conflicts and then make decisions accordingly. A challenge is to solve the dependency loop between conflicts and some greedy heuristic algorithm will be proposed to tackle with this NP-hard problem.



	Name	Inst	City	State	Country	Zip
t1	J. Davis	UQ	Brisbane	QLD	AU	4072
t2	T. Smith	UQ	Brisbane	NSW	AU	4072
t3	B. Wilson	UQ	Sydney	QLD	AU	4072
t4	Z. Wang	QUT	Brisbane	QLD	AU	4072
t5	A. Jones	QUT	Brisbane	QLD	AU	4001
t6	B. Brown	UTS	Sydney	NSW	AU	2006
t7	E. Widom	UST	Sydney	NSW	AU	2006
t8	W. Wang	UST	Hefei	Anhui	PRC	230026
t9	J. Aref	UST	Sydney	NSW	PRC	230026
t10	H. Cheng	USTC	Hefei	Anhui	PRC	230026
t11	J. Han	UIUC	Champain	IL	USA	61801
...	...	...	...	...	...	...

ID	Given FD/CFDs in the Sketch	Confidence
$\phi 1$	Name → Inst	0.85
$\phi 2$	[Country="AU", Inst] → City	1.00
$\phi 3$	[Country="PRC", Inst] → City	0.95
$\phi 4$	City → State	0.90
$\phi 5$	State → Country	0.95
$\phi 6$	Inst → [Country="AU" or "PRC", Zip]	1.00
$\phi 7$	[Country="AU" or "PRC", Zip] → Inst	1.00
$\phi 8$	[Country="AU", Inst] → State	0.90
$\phi 9$	[Country="PRC", Inst] → State	0.85

(a) Example Table with 13 Errors Highlighted

(b) A set of the Constraints (FDs/CFDs) holding on the Table

Fig. 1. A Running Example for Illustration

Contributions. We develop CrowdAidRepair, a novel Crowd-Aided Data Repairing approach, which performs crowd-based and rule-based repairing alternatively for achieving a high repairing quality at the minimum crowd cost. We identify and study the quality-constrained interaction problem between crowd-based and rule-based repairing, targeting at a balance between repairing quality and repairing cost. After proving in theory that the optimal interaction scheme is unlikely to be identified, we propose our algorithm to generate efficient interaction schemes.

Roadmap. The rest of the paper is organized as follows: We define the problem in Sec. 2, and then present our algorithm in Sec. 3. The experiments are reported in Sec. 4, followed with related work in Sec. 5. We conclude in Sec. 6.

2 Preliminary and Problem Statement

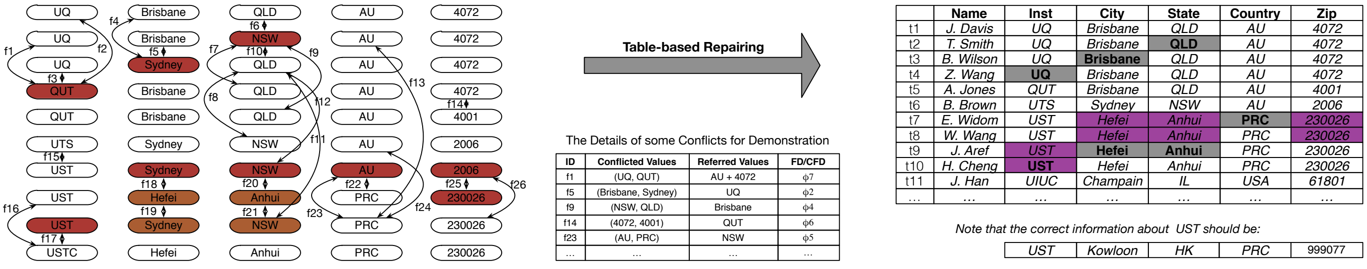
2.1 Preliminary on Rule-based Repairing

Definition 1. *We say a set of values are correct if all values in this set are correct. We say a* **Conflict** *happens between two sets of values if the two sets of values cannot be both correct.*

The rule-based repairing method relies on a set of predefined quality rules to detect conflicts between data, and then work to resolve these conflicts with expecting to clean relevant errors that have aroused these conflicts. Particularly in this paper, we take FD/CFD as an example of quality rules to show how our method works. For easier understanding, we present the preliminary with a running example.

Example 1. Given a personal contact data depicted in Fig. 1(a), where each tuple contains the `Name`, `Email` and `Inst` (Institution) of a person, in addition to one's address information: `City`, `State`, `Country` and `Zip`. We highlight errors waiting to be unveiled and corrected in the table. A set of FD/CFDs holding on the table are listed in Fig. 1(b).

1) Conflicts Detection. According to the given FD/CFDs, a number of conflicts between data can be detected from the table. For example, according to ϕ_2,



The Details of some Conflicts for Demonstration

ID	Conflicted Values	Referred Values	FD/CFD
f1	(UQ, QUT)	AU + 4072	$\phi7$
f5	(Brisbane, Sydney)	UQ	$\phi2$
f9	(NSW, QLD)	Brisbane	$\phi4$
f14	(4072, 4001)	QUT	$\phi6$
f23	(AU, PRC)	NSW	$\phi5$
…	…	…	…

	Name	Inst	City	State	Country	Zip
t1	*J. Davis*	*UQ*	*Brisbane*	*QLD*	*AU*	*4072*
t2	*T. Smith*	*UQ*	*Brisbane*	**QLD**	*AU*	*4072*
t3	*B. Wilson*	*UQ*	**Brisbane**	*QLD*	*AU*	*4072*
t4	*Z. Wang*	**UQ**	*Brisbane*	*QLD*	*AU*	*4072*
t5	*A. Jones*	*QUT*	*Brisbane*	*QLD*	*AU*	*4001*
t6	*B. Brown*	*UTS*	*Sydney*	*NSW*	*AU*	*2006*
t7	*E. Widom*	*UST*	*Hefei*	*Anhui*	**PRC**	*230026*
t8	*W. Wang*	*UST*	*Hefei*	*Anhui*	*PRC*	*230026*
t9	*J. Aref*	*UST*	**Hefei**	**Anhui**	*PRC*	*230026*
t10	*H. Cheng*	**UST**	*Hefei*	*Anhui*	*PRC*	*230026*
t11	*J. Han*	*UIUC*	*Champain*	*IL*	*USA*	*61801*
…	…	…	…	…	…	…

Note that the correct information about UST should be:

UST	*Kowloon*	*HK*	*PRC*	999077

(a) The Conflicts Identified with FD/CFDs in the Running Example

(b) The Table after Table-based Repairing

Fig. 2. The Performance of Rule-based Repairing Method

t_1[`City`] ("Brisbane") and t_3[`City`] ("Sydney") are conflicted with each other as they both correspond to the same `Inst` ("UQ"). Fig. 2(a) shows that 26 conflicts can be identified according to the constraints in Table 1(a), where each node denotes an attribute value in the table (erroneous values are highlighted), and each line between two nodes denotes a conflict between the two nodes.

2) Conflicts Resolution. When a conflict happens between values, some values should be modified in order to resolve the conflict. In order to resolve all the conflicts in a database, some works tend to make the least changes to the data set [2, 9], while others prefer to make the most likely correct changes based on some simple prediction model [8, 13]. For example in Fig. 2(a), since t_4[`Inst`] ("QUT") is conflicted with three other values ("UQ"). To resolve the three conflicts, we either change t_4[`Inst`] ("QUT") into "UQ" (cost is 1), or change the three "UQ" into "QUT" (cost is 3). The first modification way is preferred according to either of the two criterions. Fortunately, this is also the correct modification way.

However, the criterions will make wrong decisions in three situations below:

1) It is very likely to make wrong decisions based on a simple criterion. For example, both t_8[`Inst`] ("UST") and t_9[`Inst`] (" UST") are conflicted with t_{10}[`Inst`] ("USTC"), to make the least change, t_{10}[`Inst`] ("USTC") will be changed into "UST". As a result, one more error is produced.
2) Some conflict contains no errors, such as the conflict f_{10} between t_8[`Inst`] ("UST") and t_{10}[`Inst`] ("USTC") in Fig. 2, but the method tends to make corrections to every conflict;
3) The method can not make right corrections when there is no correct correction value to a position within the data set. For instance, in the conflict f_{13} and f_{14}, both the two values ("Sydney" and "Hefei") are incorrect `City` that "UST" locates at (which should be "Kowloon (HK)"), but the method will still pick one from these erroneous values as the correction value.

3) Correction Confidence Estimation. A correction by rule-based repairing is decided jointly by the FD/CFD and all relevant values that used in deducing this correction value. Therefore, the quality of a correction is also determined by the quality of these referred values that are used to deduce this correction value, and the confidence of the referred FD/CFD, that is,

$$c(v_c) = c(\phi) \times \prod_{v_i \in V_R} c(v_i) \tag{1}$$

, where V_R contains a set of referred values that used to deduce v_c for the position, $c(v)$ denotes the confidence of a value v, and $c(\phi)$ denotes the confidence of ϕ.

2.2 Problem Statement in the Interaction

We still rely on FD/CFDs to identify conflicts between data, but to identify and correct erroneous values in these conflicts, we consider to involve the crowd into the repairing process to help improve the repairing quality in an efficient interactive way. Particularly, we temporarily neglect the wrong modifications that might be made by the crowd in this paper, and will take it as future work.

The basic interaction can be described as: We set a quality constraint and do rule-based repairing to those conflicts that can satisfy the quality constraint. Then we select some values for crowd-based repairing to let more values be repairable (or so-called *deducible*) to rule-based method. We continue with this interactive repairing process iteratively until no more values can be modified.

The interaction between crowd-based and rule-based repairing can be represented by a sequence of value sets, denoted as $\mathcal{S} = \langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \mathcal{W}_2, \mathcal{T}_2, \cdots, \mathcal{W}_n, \mathcal{T}_n \rangle$, where $\mathcal{W}_i$ is a set of values for repairing at the i-th crowd-based repairing step and $\mathcal{T}_i$ is a set of values for repairing at the i-th rule-based repairing step, $\forall i \neq j, \mathcal{W}_i \cap \mathcal{T}_i = \mathcal{W}_i \cap \mathcal{T}_j = \mathcal{W}_i \cap \mathcal{W}_j = \mathcal{T}_i \cap \mathcal{T}_j = \emptyset$, and $\forall i, \mathcal{W}_i \subseteq \mathbb{V}, \mathcal{T}_i \subseteq \mathbb{V}$, where $\mathbb{V}$ denotes the domain of all values in the data set. Note that there is no fix number of values for repairing. An interaction scheme is a qualified one as long as it resolves all the detectable conflicts in the data set.

Since the cost of a crowd-based repairing operation is much more expensive than a rule-based repairing operation or any other computational process, the cost of CrowdAidRepair following an interaction scheme $\mathcal{S}$ can be roughly represented by the number of values for crowd-based repairing in $\mathcal{S}$, i.e., $cost(\mathcal{S}) = \sum_{1 \leq i \leq n} |\mathcal{W}_i|$, where $|\cdot|$ is the size of a set.

Definition 2. *(***Quality-Constrained Interaction Problem***). Given a relational table T for repairing, a set of predefined FD/CFDs Φ holding on T, a quality measuring scheme* $c(\cdot)$ *and a quality threshold τ ($0 \leq \tau \leq 1$), the object is to identify an optimal interaction scheme $\mathcal{S}_{op}$ for repairing values in T, which satisfies: (1) resolving all the conflicts in T w.r.t. Φ; (2) $\forall v_c, c(v_c) \geq \tau$, where v_c denotes a correction value; (3) $\forall \mathcal{S}'$ satisfying the above two conditions, we have $cost(\mathcal{S}_{op}) \leq cost(\mathcal{S}')$.*

See the situation in the running example, an optimal interaction scheme constructed manually can be:

$< \{t_4[\texttt{Inst}]\}_w, \{t_3[\texttt{City}], t_2[\texttt{State}]\}_t, \{t_7[\texttt{Country}], t_7[\texttt{Zip}]], t_8[\texttt{Zip}]], t_9[\texttt{Inst}]\}_w,$ $\{t_8[\texttt{City}]], t_8[\texttt{State}]]\}_t, \{t_7[\texttt{City}], t_7[\texttt{State}]]\}_w, \{t_8[\texttt{City}]], t_8[\texttt{State}]]\}_t >$, which has 7 values for crowd-based repairing, and the left 6 values for rule-based repairing.

However, the optimal interaction scheme is not feasible to be constructed automatically as described below. Due to the limitation of space, we put the proof to Theorem 1 at: http://pan.baidu.com/s/1c05Cpqg.

Theorem 1. *The optimal interaction scheme to the Quality-Constrained Interaction problem is not feasible to be achieved.*

3 A Quality-Constrained Interaction Algorithm

We present our algorithm for generating an efficient interaction scheme. The key problem here lies on how to select values for crowd-based repairing at each crowd-based repairing step.

Initially, we tend to choose the value that has aroused the most conflicts between data for crowd repair, such that the most values will become deducible in the next rule-baed repairing step. In order to find out the value that has aroused the most conflicts between data, we estimate a so-called "disharmonious degree" (or *dScore* for short) for each value, to denote the "disharmony" between this value and all the other values in the data set. We will introduce how we estimate the dScore of each value in Sec. 3.1.

In addition to dScore, the dependency relations between conflicts should also be taken into account. We say a conflict f_a depending on another conflict f_b, if some values in f_b are the reasons (or part of the reasons) that have aroused the conflict in f_a according to some FD/CFD. Let a conflict f_a depends on another conflict f_b, we normally should process f_b prior to processing f_a for three reasons below: (1) Initially, It is possible that after the conflict in f_b is resolved, the conflict in f_a is dismissed automatically without any repairing operations. (2) To say the least, even if f_a is a true conflict and we need to do crowd-based operations to check the values inside it, sometimes we have no other choices but to rely on those values in the conflicts that f_a depends on to formulate crowd-based repairing queries. (3) Lastly, after we process all conflicts it depends on, we can update the *dScores* for the values in f_a for better judging which value is more likely an error.

Although we are in a dynamic setting to schedule conflicts and their covered values for repairing, we can still fix something based on the dependency relations between conflicts and then make decisions accordingly. A challenge is to solve the dependency loop between conflicts. We will discuss this in Sec. 3.2.

3.1 dScore: Estimating the Incorrectness of Values

The dScore of a value can be roughly reflected by the number of conflicts it brings to the data set. We first introduce how to calculate the *dScore* for each value in a simplified case. To begin with, we assume that the data set is consistent without the value at a position, that is, all the other values in the data set appear to be in harmony. Then the value at this position comes, which may bring conflicts in two ways: (1) itself conflicts with some values; (2) it may let some values involved in a conflict. Usually, the more conflicts it brings to the data set, the higher probability it is an erroneous value. In other words, the *dScore* of a value can be manifested as the number of conflicts it caused in this simple setting.

We now consider the situation in real case, where there are already erroneous values and conflicts in the data set. When a new value at a position comes, either an erroneous one or not, it brings some changes anyway, such as producing new conflicts, or voting for existing conflicts. In this case, the *dScore* of a value can be manifested by two things: (1) the new conflicts produced, and the "credibility", or what we call the *cScore* of these conflicts, which will be discussed in Eq. (3); (2) the changes on the *cScore* of existing conflicts. Specifically, $dScore(v)$ of a value v can be calculated by:

$$dScore(v) = \alpha \times \sum_{f \in F(v)} \Delta(cScore(f)) \tag{2}$$

where α is a normalization factor to scale $dScore(v)$ between 0 and 1, $F(v)$ contains all conflicts that are influenced by putting v into the data set, and $\Delta(cScore(f))$ is the change on the *cScore* of a conflict f.

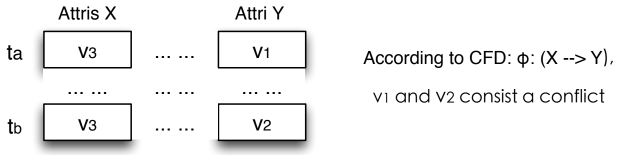


Fig. 3. An Example Conflict with Relevant Values and CFD

In particular, the *cScore* of a conflict f is decided by four relevant values as given in Fig. 3. Previous work considers that a conflict is consisted by two values such as v_1 and v_2 in the figure, but a conflict is also closely related to another two values which are referenced to identify the conflict according to a certain CFD, such as the two v_3 in the figure. Thus, the correctness of the four values jointly decide the *cScore* of a conflict f. Furthermore, when a conflict is voted as a conflict by several groups of values w.r.t. different CFDs, we only pick the one with the highest *cScore* as the final *cScore* of the conflict. More specifically,

$$cScore(f) = \underset{\phi \subseteq \Phi(f)}{\mathrm{ArgMax}}[c(\phi) \times \prod_{v'_i \in V(f,\phi)} (1 - dScore(v'_i))] \tag{3}$$

where $\Phi(f)$ is the set of CFDs that voted f as a conflict, and $V(f, \phi)$ contains all values related to the conflict f w.r.t. ϕ.

3.2 Employing Dependencies between Conflicts

We consider the dependencies between conflicts in scheduling conflicts for repairing. We first get the dependency relations among all conflicts, and then build a conflicts dependency graph based on these relations.

1) Relations between Conflicts. Basically, there are three kinds of relationships between each pair of conflicts. The first is the *Dependency Relation* as we introduced above. Note that the dependency relation is transitive, that is, if f_a depends on f_b, and f_b depends on f_c, then f_a also depends on f_c. Secondly, we say two conflicts are in a *Overlapped Relation* if they share some positions, such as f_1 and f_2 sharing t_4[`Inst`](“QUT”) in Fig. 2(a). Finally, If two conflicts are in neither of the two relations above, they are *Independent* from each other.

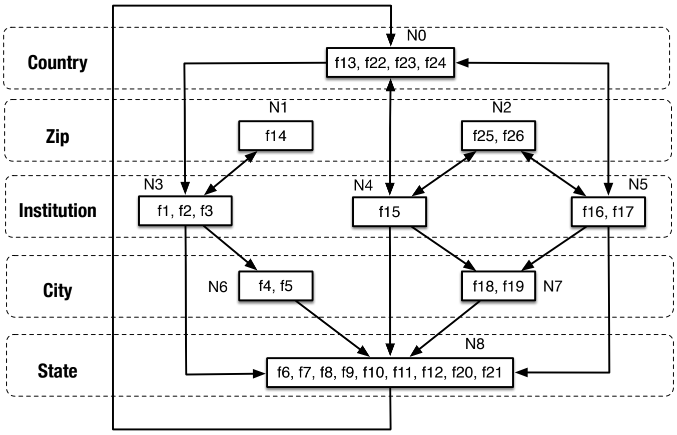


Fig. 4. The Dependency Graph of the Conflicts in Table 1(a)

2) Building Conflicts Dependency Graph. With the relations between all conflicts, we can built a conflicts dependency graph as in Fig. 4 (which is built on Table 1(a)) through the following steps:

(1) Initially, we take each conflict as a node in the dependency graph.
(2) We then put a directed edge pointing from every conflict f_a to every other conflict f_b if f_b depends on f_a. Note that we only need to put an edge between two conflicts if one directly depends on the other.
(3) Finally, to make the graph easier to process, we merge nodes sharing at least one value into one node (i.e., we put overlapped conflicts into one node), and the directed edges of the same direction between the two nodes are merged into one directed edge.

As introduced above, a conflict should be processed after all the conflicts it depends on are processed. But for those overlapped conflicts in the same node, we need to consider the priority of each value that involved in the conflicts for checking. Here we can still rely on the *dScores* of these values. A value with a highest *dScore* in a node can be checked firstly. Each time a value is modified, the graph needs to be updated accordingly.

3.3 Tackling Dependency Loops

The main challenge here is how to schedule those conflicts in dependency loops for processing. We say a number of conflicts are in dependency loop if they depend on each other such as f_1, f_2, f_3 and f_{14}. In this situation, the dependency-based interaction principle mentioned above dose not work at all. Things become more intractable when there are several loops overlapped with each other at different nodes. As in Fig. 4, there are 19 loops in total and almost every loop is overlapped with some other loops at some nodes. Basically, we have to choose one (or more than one) node in a loop to process to "break up" the loop. In order to minimize the cost, we have to be very careful in selecting the *break-up node* for a loop as different break-up nodes will bring different costs.

Theorem 2. *It is an NP-hard problem to break up loops in a dependency graph with the minimum crowd-based repairing cost.*

Proof: We prove it by reduction from constrained minimum spanning tree problem [11]. Constrained minimum spanning tree is defined over an undirected graph $G = (V, E)$ and nonnegative integers l_e and w_e for each edge $e \in E$, where l_e and w_e are the length and weight of e respectively. The problem is to find a spanning tree that has low total cost with respect to both the cost functions l and w. Although the dependency graph in our case is a directed graph, it is worthy noticing that each undirected graph can be considered as a directed graph.

To reduce the constrained minimum spanning tree problem to our problem, we let the length of $e = (v_1, v_2)$ is the confidence between v_1 and v_2. And we let the weight of $e = (v_1, v_2)$ is the cost for repairing v_2. Assuming a spanning tree $G' = (V', E') \in G$ is generated from G, then our object is to maximize $C(G') = \sum_{e \in E'} w_e$, where $C(G')$ is the total cost of G'. This is equivalent to minimize the cost of crowd-based repairing denoted by $C(crowd)$, i.e. we have

$$C(crowd) = \sum_{e \in E \setminus E'} w_e = C(G) - C(G') \tag{4}$$

Hence, the objective function is consistence with the objective function in the constrained minimum spanning tree problem. Thus, theorem 2 is proved.

Due to the limitation of space, we put the proof to Theorem 2 at: http://pan.baidu.com/s/1c05Cpqg. □

In the following, we give our greedy algorithm to break up loops in a dependency graph.

1) Breaking up a Single Loop. We basically consider two factors in selecting the break-up node for a loop: (1) factor 1: the number of values that must be verified in a node for breaking up the loop (for easier presentation, we call these values as break-up values); (2) factor 2: the *dScores* of these break-up values in a node. Usually, we tend to select the node with the least number of break-up values holding the highest *dScores* as the break-up node for the loop. More specifically, we calculate a break-up score, or *bScore* for short, for each node in a loop as given in Eq. 5 below. Among all nodes in a loop, the node with the highest *bScore* will be selected as the break-up node in priority.

$$bScore(\mathcal{N}, \mathcal{L}) = \prod_{v \in V_b(node, loop)} dScore(v) \tag{5}$$

where $V_b(\mathcal{N}, \mathcal{L})$ is the set of break-up values in *node* for breaking up *loop*.

Algorithm 1: Dependency-Aware Interaction

```
Input  : A table with a set of conflicts $\mathbb{F}$
Output: A repairing scheme $\mathcal{S} = \langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \cdots, \mathcal{W}_n, \mathcal{T}_n \rangle$
Set $i = 0$;
while $\mathbb{F} \neq \emptyset$ do
    1. $\mathcal{T}_i \leftarrow$All deducible values at the moment;
    2. Deducing all values in $\mathcal{T}_i$;
    3. Updating $\mathbb{F}$;
    4. $i++$;
    5. Calculating $dScores$ for all values in $\mathbb{F}$ with Eq. 2;
    6. Building the Dependencies Graph on $\mathbb{F}$;
    7. while no new deducible values do
        $V \leftarrow$ Values in conflicts depending on nothing;
        if $V \neq \emptyset$ then $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup V$ else
            Calculating $gbScores$ for all conflicts in $\mathbb{F}$ with Eq. 6;
            $V \leftarrow$ Values with the highest $dScore$ in conflicts with the highest $gbScore$;
            $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup V$
        end
        Checking/Repairing $V$ with the Web;
        if $V$ is updated with correction values then
            Updating $\mathbb{F}$ and $dScores$;
        end
    end
end
return $\langle \mathcal{T}_0, \mathcal{W}_1, \mathcal{T}_1, \cdots, \mathcal{W}_n, \mathcal{T}_n \rangle$;
```

2) Breaking up Multiple Loops. For a number of loops overlapped with each other, we can not simply decide the break-up nodes for a single loop. Otherwise, we may not be able to reach the best performance in minimizing the number of crowd-based operations. For each node, we consider a global *bScore*, or *gbScore* for short, to denote its break-up score for all loops in the graph, and the one with the highest *gbScore* will be selected as the break-up node in priority. The *gbScore* of a node is decided by two factors: (1) the local *bScore* of the node in each loop; and (2) the benefit of solving each loop, which is actually the number of values that can be moved out from the loops. More specifically,

$$gbScore(\mathcal{N}) = \sum_{\mathcal{L} \in L(\mathcal{N})} [bScore(\mathcal{N}, \mathcal{L}) \times benefit(\mathcal{N}, \mathcal{L})] \tag{6}$$

where $L(\mathcal{N})$ is the set of loops having $\mathcal{N}$ as its node in the graph, and the $benefit(\mathcal{N}, \mathcal{L})$ is the benefit of breaking up $\mathcal{L}$ by solving $\mathcal{N}$, which is mainly decided by the number of values in $\mathcal{L}$.

3.4 Dependency-Aware Interaction Algorithm

A formal description of this algorithm is given in Alg. 1. Initially, we build the conflicts dependency graph for a data set. For those nodes depending on nothing, we keep on choosing the value with the highest *dScore* within each node for crowd-based repairing until all the conflicts in the node are resolved. When there is no node of this kind but only loops, we calculate the *gbScores* for all nodes in these loops, and choose the one with the highest *gbScore* to process to break up the loops. Each time a value is modified, we need to update the graph and all *bScores* and *gbScores*. The algorithm stops when the graph is empty. Basically, the computation complexity of Alg. 1 is $O(mlog\ m + n)$, where m is the number of nodes which are in the loops of the graph and n is the number of nodes which are not in the loops of the graph.

Table 1. The Interaction Scheme generated by Alg. 1

$\mathcal{T}_0$	$\emptyset$
$\mathcal{W}_1$	t_7[Inst](“UST”) is correct, not changed; t_4[Inst](“QUT”) is incorrect, modified; t_3[City](“Sydney”) is incorrect, modified; t_2[State](“NSW”) is incorrect, modified; t_8[Inst](“UST”) is correct, not changed; t_9[Inst](“UST”) is incorrect, modified;
$\mathcal{T}_1$	t_9[City](“Sydney”) is incorrect, modified; t_9[State](“NSW”) is incorrect, modified; t_9[Country](“PRC”) is correct, not changed;
$\mathcal{W}_2$	t_8[Zip](“230026”) is incorrect, modified;
$\mathcal{T}_2$	t_7[Zip](“2006”) is incorrect, modified;
$\mathcal{W}_3$	t_8[City](“Hefei”) is incorrect, modified;
$\mathcal{T}_3$	t_7[City](“Sydney”) is incorrect, modified.
$\mathcal{W}_4$	t_8[State](“Anhui”) is incorrect, modified;
$\mathcal{T}_4$	t_7[State](“NSW”) is incorrect, modified;
$\mathcal{W}_5$	t_7[Country](“AU”) is incorrect, modified;
$\mathcal{T}_4$	t_8[Country](“PRC”) is correct, not changed

Example 2. We apply the algorithm to the running example and the interaction scheme generated is depicted in Table 1. Overall, we issue crowd-based operations for 10 values, among which 8 values are true erroneous values while the other 2 values are correct values. Meanwhile, 5 erroneous values are corrected by rule-based repairing.

Theorem 3. *The crowd-based repairing cost of Algorithm* 1 *is not larger than* $\frac{e-1}{e}$ *times of the cost of the optimal dependency-aware interaction algorithm.*

Proof: Consider the iteration where the dependency-aware interaction algorithm implements. We define the following function to represent our algorithm.

$$B'(S) = B(G(V)) - B(G(V)\backslash S) \tag{7}$$

□

Denote the sequence $S = \{G(v_1), G(v_2), ..., G(v_n)\}$ the result of algorithm 1. And denote $OPT = \{G(v_1'), G(v_2'), ..., G(v_n')\}$ the optimal sequence for the dependency-aware interaction algorithm. And let $P = \{G(v_1'), G(v_2'), ..., G(v_k')\}$, where $k < n$. Let $G(v)$ the first element from $OPT\backslash P$ and $\Delta = C(G(E\backslash v) \cup P) - C(G(E\backslash v))$, then we have the following inequality according to [6].

$$B'(G(E)\backslash S) + \Delta > (1 - 1/e)B'(OPT\backslash P) \tag{8}$$

Notice that $B'(\cdot)$ is an non-decreasing function which means repairing more nodes will produce more benefit. Besides, please recall Eq. 7, we then have

$$\begin{aligned} B(G(V)) &= B(G(E)\backslash S) + B(G(E)\backslash G(E\backslash v)) \\ &> B(G(E)\backslash S) + (1 - 1/e)B(OPT\backslash P) - \Delta \\ &> B(P) + (1 - 1/e)B(OPT\backslash P) \\ &= (1 - 1/e)B(OPT) \end{aligned} \tag{9}$$

where $B'(\cdot)$ and $B(\cdot)$ are the benefit functions when we identify $\cdot$. Consider the nodes which are not in the loops, then our algorithm is equivalent to the optimal one in this situation. Thus, $B(G(V)) > (1 - 1/e)B(OPT)$ always holds. Meanwhile, we notice that the benefit of identifying a node is inversely proportional to its cost. Thus we have

$$C(G(V)) < \frac{e-1}{e}C(OPT) \tag{10}$$

Hence, theorem 3 is proved.

4 Experiments

We experiment on two real and one synthetic data sets.

1) Personal Information Table (**PersonInfo**): This is a 50k-tuples, 9-attributes table, which contains contact information for academics including name, e-mail, title, university, street, city, state, country and zip code.
2) DBLP Publication Table (**DBLP**): This is a 100k-tuples, 5-attributes table. Each tuple contains information about a published paper, including its title, first author and his/her affiliation, conference name, year and venue.
3) Synthetic Table (**Syn**): We also generate a 1million-tuples, 100-attributes table following a scheme containing 100 randomly generated approximate attribute dependencies with confidences near-uniformly distributed between 0.7 and 1, where the first attribute is the key attribute.

All the three data sets are relational tables without erroneous data. To generate tables with errors for the experiments, we keep the key attribute value in each tuple and replace non-key attribute values at random positions with attribute values selected from random picked tuples of the table.

4.1 Repairing Quality Evaluation

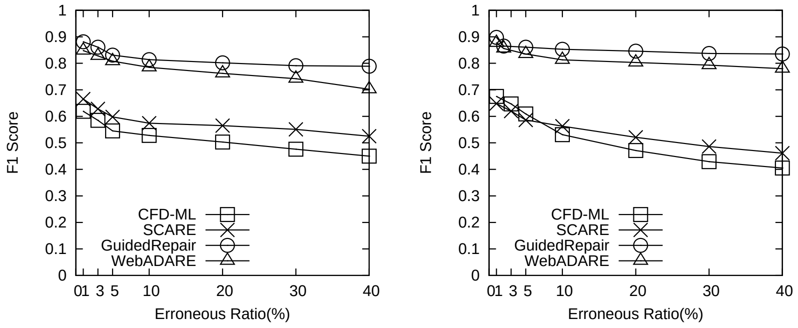
In the following experiments, we compare the repairing quality of PureCrowdRepair (Pure crowd-based Repairing) and CrowdAidRepair with four state-of-the-art general textual data repairing approaches on the three data sets.

1) Rule-based Least-Change (**CFD-LC**): This is the mainstream approach relying on FD/CFDs to detect and correct erroneous data in a data set [4, 8], which follows the least-change modification criterion in repairing.
2) Rule-based Most-Likely (**CFD-ML**): This approach also relies on FD/CFDs to detect and correct erroneous data [2], but follows the most-likely correct modification criterion as introduced in Sec. 2.1.
3) Model-based 1 (**ERACER**): This is a model-based repairing based on belief propagation and relational dependency networks [10]. In contrast to prior work that cleans tuples in isolation, this approach exploits the graphical structure of the data to propagate inferences throughout the database [10].
4) Model-based 2 (**SCARE**): This is another model-based repairing approach based on maximizing the correctness likelihood of replacement data given the data distribution, which is modeled using statistical machine learning techniques [13].

Table 2. Comparing the Repairing Quality with Previous Methods

	PersonInfo			**DBLP**		
	Prec	Recall	F1	Prec	Recall	F1
CFD-LC	0.506	0.517	0.511	0.613	0.423	0.501
CFD-ML	0.535	0.521	0.528	0.645	0.451	0.531
ERACER	0.638	0.489	0.554	0.698	0.421	0.525
SCARE	0.655	0.512	0.574	0.743	0.453	0.563
PureCrowdRepair	**0.836**	**0.756**	**0.794**	**0.931**	**0.722**	**0.813**
CrowdAidRepair	**0.821**	**0.753**	**0.785**	**0.928**	**0.719**	**0.810**

We first make a comprehensive comparison on the Precision, Recall and F1 of all methods at an erroneous ratio of 10% on the two real-world data sets. The parameter setting for each method lets the method reach the best repairing quality (w.r.t. F1). As shown in Table 2, the precision and recall of the two rule-based methods (CFD-LC and CFD-ML) are not high, as they can only make correct modifications to about half of the erroneous values in the data sets, and in 40-60% chances they make wrong corrections. The reasons are two: (1) about 20% erroneous values in each data set can not be detected at all by the employed conflict detector as these errors do not arouse any conflicts in the data set; (2) without external knowledge for guidance, following a simple modification criterion is very likely to make mistakes. Comparatively, the precision of the two model-based methods (ERACER and SCARE) is a bit (5-10%) higher than the rule-based methods since the models they build can understand the correlation between data and thus make better judgements. On the other hand, their recall is as low as that of the rule-based method, since there are some non-quantitive attributes like email, street, author, and venue which can not be handled well by models. In contrast, the precision and recall of PureCrowdRepair and CrowdAidRepair are much higher (80-90% precision and 80-85% recall), which benefits from the external data on the Web.



(a) PersonInfo (b) DBLP

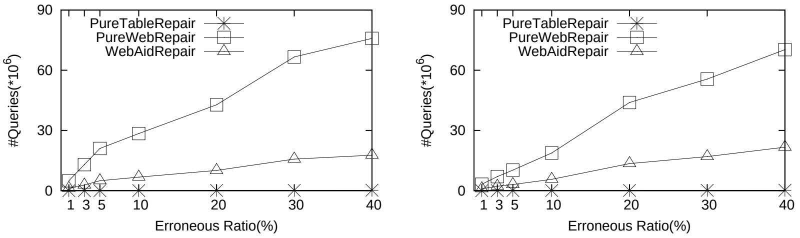
Fig. 5. Comparing the F1 Scores of all Methods

We then compare the F1 scores of all methods at various erroneous ratios $(1\%, 3\%, 5\%, 10\%, 20\%, 30\%, 40\%)$ by setting τ=0.7 over the two real-world data sets. As demonstrated in Fig. 5, CrowdAidRepair always reaches much higher F1 scores than all the other four methods, which proves the advantage of CrowdAidRepair over the four other methods. Besides, although PureCrowdRepair reaches as high (or even a bit higher) F1 scores as CrowdAidRepair, it is much more expensive to do pure crowd-based repairing than CrowdAidRepair, which will be reflected in the next experiments.

4.2 Repairing Cost Evaluation

We now compare the cost of CrowdAidRepair with pure crowd-based method (PureCrowdRepair) and pure table-based method (CFD-ML) on **#Queries**, which is the number of issued queries to the Web.

(2) #Queries: To better reflect how can CrowdAidRepair be much more efficient than PureCrowdRepair, we also compare the two methods on their issued queries over the two data sets. As demonstrated in Figure 6(c)(d), CrowdAidRepair only retrieves about 20% of the values retrieved by PureCrowdRepair and thus greatly reduces the overhead of the repairing process. Note that the #Queries of a method is usually a bit larger than the number of retrieved values, provided that a fired retrieving query may fail to retrieve the target erroneous value, and then we need to fire alternative ones.

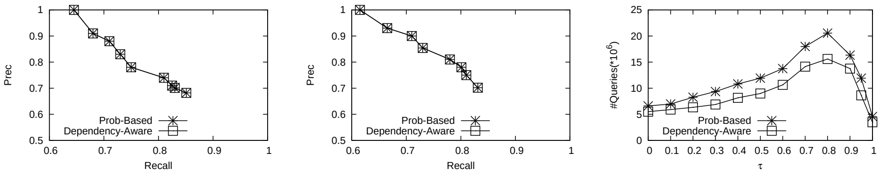


(a) PersonInfo (# Queries) (b) DBLP (# Queries)

Fig. 6. Comparing #Queries on the Real Data Sets($\tau = 0.7$, erroneous ratio = 10%)

4.3 Interaction Schemes Evaluation

We do experiments to evaluate the interaction schemes generated by the three proposed algorithms over the three data set. We set the erroneous ratio to 10%, and then compare the repairing quality (precision and recall) and cost of the interactive repairing following each interaction scheme by changing the quality threshold τ from 0 to 1. As shown in Fig. 7(a)(b)(c), although the probabilistic-based scheme and the dependency-aware scheme can reach a higher recall than the quality-aware scheme, their precision is always 5% lower than that of the quality-aware scheme. Overall, the quality-aware scheme reaches a higher combination of precision and recall than the other two methods. On the other hand, the cost of all the schemes increases as τ increases from 0 to about 0.8, but decreases sharply as τ increases from 0.8 to 1.0 as shown in Fig. 7(d). This makes sense since when the quality constraint becomes too strict, much less values can be repaired to satisfy the constraint. Nonetheless, the cost of the quality-aware interaction scheme is always about 20% less than the dependency-aware scheme and about 40% less than the probabilistic-based scheme, which proves the advantage of the quality-aware scheme over the other two schemes.



(a) PR (PersonInfo) (b) PR (DBLP) (c) #Queries (Syn)

Fig. 7. Comparing the Schemes: Precision and Recall in Subfigures (a)(b) (on the two real data set), and #Queries in Subfigures (c) (Syn data set)

5 Related Work

Existing data repairing solutions can be roughly put into three categories below.

The traditional category of methods relies on a variety of constraints including FDs [1, 12], CFDs [4], Integrity Constraints [9] and Inclusion Dependencies (INCs) [1] to detect inconsistency (or conflicts) between data aroused by erroneous data, and then work on resolving all the conflicts with expecting to fix all erroneous data in this way [2, 8, 9]. For general textual databases, most work in this category use FD/CFDs for repairing as they are the constraints within a single relational table, while some other work uses INCs for repairing between multiple relational tables. Usually, this category of methods can effectively detect a large percent of erroneous data involved in the identified conflicts in a wide range of databases, but to repair these errors and resolve the conflicts, some work tends to make the least changes to the data set [2, 9], while others prefer to make the most likely correct changes based on some simple prediction model [8, 13]. However, neither criterion can have all errors modified correctly.

The second category of solutions are model-based repairing, which usually build some prediction models for detecting and correcting erroneous values in a data set [7, 10, 13, 15]. The construction of the model employs statistical Machine Learning (ML) techniques for data cleaning, which can effectively capture the dependencies and correlations between data in the dataset based on various analytic, predictive or computational models [13, 15]. However, not every erroneous data can be identified and corrected in the right way since there are always outliers that do not obey the captured constraints.

The third category of solutions are external source based repairing approaches, which leverage the information in reference master data set [5] or user's interaction data [14] for better data cleaning performance. However, the required external information is not always available and thus the methods can not be applied in general scenarios. In this paper, we propose CrowdAidRepair, which is also an external source-based repairing approach. Compared to previous approaches of the kind, CrowdAidRepair is more general as we rely on the Web.

6 Conclusions and Future Work

We propose CrowdAidRepair, a novel crowd-aided data repairing approach that can greatly enhance the repairing quality of the existing rule-based repairing method with the Crowd help. Extensive experimental results based on several data collections demonstrate that the generated interaction scheme decreases on average 80% cost of a baseline, and reaches almost the same high repairing quality that was reached by a pure crowd-based retrieving approach. Future work may consider combining CrowdAidRepair with state-of-the-art model-based methods, and apply CrowdAidRepair to databases with both incorrect values and missing values.

References

1. P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

2. G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *PVLDB*, pages 315–326, 2007.
3. M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.
4. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):6, 2008.
5. W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1-2):173–184, 2010.
6. S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.
7. J. L. Koh, M. L. Lee, W. Hsu, and K. T. Lam. Correlation-based detection of attribute outliers. In *Advances in Databases: Concepts, Systems and Applications*, pages 164–175. Springer, 2007.
8. S. Kolahi and L. V. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.
9. A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, pages 216–225, 2007.
10. C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010.
11. R. Ravi and M. Goemans. The constrained minimum spanning tree problem. *Algorithm TheorySWAT'96*, pages 66–75, 1996.
12. J. Wijsen. Database repairing using updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768, 2005.
13. M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, pages 553–564, 2013.
14. M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.
15. X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004.