Repair Diversification for Functional Dependency Violations

Chu He, Zijing Tan[⋆], Qing Chen, Zhihui Wang, and Wei Wang

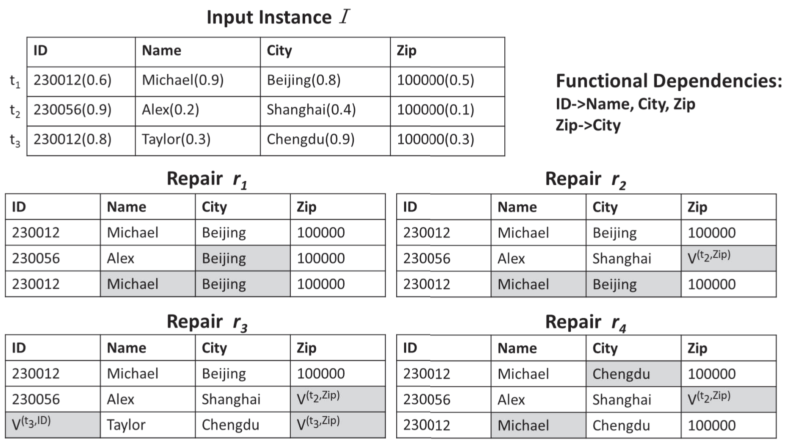
School of Computer Science,
Shanghai Key Laboratory of Data Science,
Fudan University, Shanghai, China
{12210240018,zjtan,13210240082,zhhwang,weiwang1}@fudan.edu.cn

Abstract. In practice, data are often found to violate functional dependencies, and are hence inconsistent. To resolve such violations, data are to be restored to a consistent state, known as "repair", while the number of possible repairs may be exponential. Previous works either consider optimal repair computation, to find one single repair that is (nearly) optimal *w.r.t.* some cost model, or discuss repair sampling, to randomly generate a repair from the space of all possible repairs.
This paper makes a first effort to investigate repair diversification problem, which aims at generating a set of repairs by minimizing their costs and maximizing their diversity. There are several motivating scenarios where diversifying repairs is desirable. For example, in the recently proposed interactive repairing approach, repair diversification techniques can be employed to generate some representative repairs that are likely to occur (small cost), and at the same time, that are dissimilar to each other (high diversity). Repair diversification significantly differs from optimal repair computing and repair sampling in its framework and techniques. (1) Based on two natural diversification objectives, we formulate two versions of repair diversification problem, both modeled as bi-criteria optimization problem, and prove the complexity of their related decision problems. (2) We develop algorithms for diversification problems. These algorithms embed repair computation into the framework of diversification, and hence find desirable repairs without searching the whole repair space. (3) We conduct extensive performance studies, to verify the effectiveness and efficiency of our algorithms.

1 Introduction

Data consistency is one of the issues central to data quality. We say data is inconsistent when they violate predefined data dependencies, *e.g.,* functional dependencies, and repairing data means restoring the data to a consistent state, *i.e.,* a *repair* of the data. Violations of functional dependencies are commonly found in practice. There is generally no single deterministic way of rectifying inconsistencies; former works address this issue mainly based on two approaches.

[⋆] Corresponding Author


Input Instance I

	ID	Name	City	Zip
t_1	230012(0.6)	Michael(0.9)	Beijing(0.8)	100000(0.5)
t_2	230056(0.9)	Alex(0.2)	Shanghai(0.4)	100000(0.1)
t_3	230012(0.8)	Taylor(0.3)	Chengdu(0.9)	100000(0.3)

Functional Dependencies:
ID->Name, City, Zip
Zip->City

Repair r_1

ID	Name	City	Zip
230012	Michael	Beijing	100000
230056	Alex	Beijing	100000
230012	Michael	Beijing	100000

Repair r_2

ID	Name	City	Zip
230012	Michael	Beijing	100000
230056	Alex	Shanghai	$V^{(t_2,Zip)}$
230012	Michael	Beijing	100000

Repair r_3

ID	Name	City	Zip
230012	Michael	Beijing	100000
230056	Alex	Shanghai	$V^{(t_2,Zip)}$
$V^{(t_3,ID)}$	Taylor	Chengdu	$V^{(t_3,Zip)}$

Repair r_4

ID	Name	City	Zip
230012	Michael	Chengdu	100000
230056	Alex	Shanghai	$V^{(t_2,Zip)}$
230012	Michael	Chengdu	100000

Fig. 1. Instance, functional dependencies and repairs

The first approach, namely optimal repair computation [3, 5, 6, 10, 13], aims to find a repair with the minimum repair cost, while the second one, namely repair sampling [4], aims to randomly generate a repair from the space of all repairs.

This paper presents a novel approach, referred to as *repair diversification*. We leave the formal framework to Section 3, where repair diversification is formalized as a bi-criteria optimization problem *w.r.t.* objective functions in terms of repair cost and distance. Informally, the goal of repair diversification is to generate a set of repairs, aiming at minimizing costs of repairs and simultaneously maximizing their diversity: generating repairs that have small cost to be practical, and that are dissimilar to each other to avoid the redundancy.

Example 1: We give an illustrative example, and leave formal definitions to Sections 2 and 3. Fig. 1 presents an instance I of schema $(ID, Name, City, Zip)$ and some functional dependencies defined on this schema. A weight (number in bracket) is associated with every attribute value in I, reflecting the confidence the data owner places in that value [3, 5, 13]; a larger weight implies a more reliable value. Instance I violates functional dependencies, and hence is inconsistent.

We also give some (not all) repairs for instance I, denoted by r_i ($i \in [1, 4]$). Each repair is a possible consistent state for this instance, obtained by modifying some attribute values. Note that some repairs have variables as attribute values, *e.g.*, $v^{(t_2,Zip)}$ in r_2, where each variable indicates a new value not used in that attribute in I. As an example, in r_3, $v^{(t_2,Zip)} \neq v^{(t_3,Zip)} \neq 100000$. □

We highlight the properties of repair diversification.

(1) Cost. Intuitively, it costs more to modify a value with a large weight in a repair, and a repair with a small cost is more likely to occur in practice. Therefore, small-cost repairs are expected to carry more meaningful information, and are hence more instructive. Optimal repair computation employs the notion of cost to find a single optimal repair, while discarding all other repairs even if they have a similar (or even the same) cost as the found optimal repair. In contrast, repair sampling generates a random repair each time; some repairs may

be impractical in terms of repair cost. As will be seen later, repair diversification includes optimal repair computation as its special case, which is already proved to be NP-complete in various settings. Worse still, neither of the former work has considered finding a set of repairs when considering their costs.

(2) Diversity. Repairs each with a small cost, may be similar to each other, and hence may fail to show its own uniqueness. Therefore, it is generally insufficient to consider only costs, when generating a set of repairs; each repair is expected to carry some *novel* information, in order to avoid redundancy. Recall repairs in Fig. 1. When diversity is concerned, repair r_3 is preferable to r_2 in forming a set with r_1. This is because r_3 shows a totally different way to restore consistency compared to r_1; their sets of modified values are disjoint. On the contrary, r_1 and r_2 deal with tuples t_1 and t_3 in the same way. To our best knowledge, neither of the former work on data repairing has considered the notion of diversity.

Repair cost and diversity may compete with each other; we therefore need to take them both into consideration, and find a tradeoff between them.

Motivating scenarios. Intuitively, repair diversification results in a set of repairs that are likely to occur (small cost), and simultaneously, that are dissimilar to each other (high diversity). Complementary to existing methods, we contend that repair diversification is useful in various practical situations. For example, as remarked in [4], the notion of consistent query answering [1] can be generalized to *uncertain query answering*, when each possible repair is regarded as a possible world. In this setting, repair diversification technique can be employed to compute a set of repairs that effectively summarizes the space of all repairs, which may be sufficient to obtain meaningful answers. As another example, since fully automated repair computation may bring the risk of losing critical data, the guided data repair framework [15] is recently proposed to involve users (domain experts) in repair computation: several representative repairs are first presented to the users, and machine learning techniques are then applied to user feedback or comments on these repairs, for improving the quality of future repairs. Due to the large number of possible repairs, techniques for selecting good representative repairs become very important. As noted earlier, repair diversification technique lends itself as an effective approach to generating such repairs, while neither optimal repair computation nor repair sampling is suitable for this setting.

Contributions. We make a first effort to investigate repair diversification.

(1) We provide a formal framework for repair diversification problem (Section 3). We first study two functions to rank repairs: a *distance* function that measures the dissimilarity of repairs, and a *cost* function that measures the possibility of a repair. We then present two diversification objectives, both to minimize repair cost and to maximize repair distance. Based on them, we propose two versions of repair diversification problem, both modeled as bi-criteria optimization problem, and show that their decision problems are NP-complete.

(2) Despite the intractability, we develop algorithms for repair diversification problems, *i.e.,* to find diversified top-k repairs based on diversification objectives (Section 4). Our algorithms embed repair computation into the framework of diversification, by employing diversification objectives to guide repair gener-

ations. Hence, these algorithms have the early termination property: they stop as soon as desirable repairs are found, without searching the whole repair space. (3) Using both real-life and synthetic data, we conduct an extensive experimental study to verify the effectiveness and efficiency of our algorithms (Section 5).

Related work. As remarked earlier, repair diversification aims to generate a set of repairs by considering both repair cost and diversity, and hence significantly differs from former works on data repairs. Specifically, (1) optimal repair computation [3, 5, 6, 10, 13] generates exactly one (nearly) optimal repair in its cost; and (2) repair sampling [4] is proposed to generate a random repair from the space of all possible repairs in each run, considering neither cost nor diversity.

There has been a host of work on data repairing (see [2, 9] for a survey of more related works). Specifically, guided (interactive) data repairing [15] is proposed to incorporate user feedback in the cleaning process for improving automatic repairing results. Moreover, the dashboard discussed in [7] supports user interaction, which provides several summarization of data violations.

One may find there are some similarities between repair diversification and query result diversification [8, 11], if regarding repair computation as a query finding repairs from the repair space, and regarding the cost of a repair as the opposite of its relevance to the query. However, repair diversification introduces new challenges that we do not encounter in query result diversification. Query result diversification is proposed to pick diversified results from a known set of relevant results. In contrast, repair diversification aims to compute and pick diversified repairs from those low-cost ones, when taking as input the exponential repair space with many costly repairs that are supposed to rarely occur in practice. These costly repairs should always be excluded from the result, even if they contribute large distances to other repairs. It is infeasible to compute all the repairs in advance, since the number of repairs is exponential and the computation of a single repair is also expensive, especially when repair cost is involved. Therefore, a tricky technique of repair diversification concerns employing diversification objectives to guide repair generation, hopefully avoiding as much as possible repair computations.

2 Preliminaries

We review some basic notations and the definition of repair [4].

An instance I of a relation schema $R(A_1, \ldots, A_m)$ is a set of tuples in $Dom(A_1) \times \cdots \times Dom(A_m)$, where $Dom(A_i)$ is the domain of attribute A_i. We denote by $Dom_I(A_i)$ the set of values that appear in attribute A_i in I. We assume that every tuple is associated with an identifier t that is not subject to updates, and use the terms tuple and tuple identifier interchangeably. We denote an attribute A_i of a tuple t in an instance I by $I(t, A_i)$, called a *cell*, and abbreviate it as $t[A_i]$ if I is clear from the context.

For an attribute set $X \subseteq \{A_1, \ldots, A_m\}$, an instance I satisfies a functional dependency (FD) $X \to A_i$, written as $I \models X \to A_i$, if for every two tuples t_1, t_2 in I such that $t_1[X] = t_2[X]$, we have $t_1[A_i] = t_2[A_i]$. We say I is *consistent* *w.r.t.* a set Σ of FDs if I satisfies every FD in Σ; otherwise I is *inconsistent w.r.t.* Σ. Similar to former works on repairs [4, 5, 13], we adopt attribute value modification as the only repair operation, which is sufficient to resolve FD violations. Specifically, for any two tuples t_1, t_2 in I that violate an FD $X \rightarrow A_i$, we fix this violation either (1) by modifying $I(t_1, A_i)$ to be equal to $I(t_2, A_i)$ (or vice versa), or (2) by introducing a new value in $Dom(A_k)\backslash Dom_I(A_k)$ to $I(t_1, A_k)$ or $I(t_2, A_k)$, where $A_k \in X$.

To distinguish introduced new values from existing constant values in I, we denote these new values by variables. Specifically, the new value introduced to $I(t_i, A)$ is denoted by $v^{(t_i,A)}$; $v^{(t_i,A_j)}$ and $v^{(t_{i'},A_{j'})}$ denote the same value if and only if $i = i'$ and $j = j'$. Observe that $v^{(t_i,A)}$ introduced to $I(t_i, A)$ may also be used as the value of $I(t_j, A)$ $(i \neq j)$ in repairing I, due to the enforcement of an FD of the form $X \rightarrow A$. To simplify notation, in this case we replace $v^{(t_i,A)}$ as $v^{(t_k,A)}$ where k is the minimum tuple identifier such that $I(t_k, A) = v^{(t_i,A)}$. Note that this is just a symbol mapping between variable names.

Repair *w.r.t.* FD. Assume that instance I of a relation schema $R(A_1, \ldots, A_m)$ is a set of tuples $\{t_1, \ldots, t_n\}$, and that I is inconsistent *w.r.t.* a set Σ of FDs. We can apply attribute value modifications to I, for a new instance I_r that is consistent *w.r.t.* Σ, called a *repair* of I. As an auxiliary notion, we denote by $mod(I, I_r)$ cells $t_i[A_j]$ that have different values in I and I_r.

We now formally define the notion of a *repair*. A repair of I *w.r.t.* Σ is an instance I_r such that (1) I_r has the same set of tuples (with identifiers) as I, and is obtained from I by a list of attribute value modifications, in which some $I(t_i, A_j)$ $(i \in [1, n], j \in [1, m])$ is replace by either (a) a different constant in $Dom_I(A_j)$, or (b) a variable denoting a new value in $Dom(A_j)\backslash Dom_I(A_j)$; (2) I_r is consistent *w.r.t.* Σ; and (3) there is no $I_{r'}$ that satisfies (1) and (2), such that $mod(I, I_{r'}) \subset mod(I, I_r)$ and for each $t_i[A_j] \in mod(I, I_{r'})$, $I_{r'}(t_i, A_j) = I_r(t_i, A_j)$, where $I_{r'}(t_i, A_j)$ is either a constant or a variable.

Example 2: All repairs presented in Fig. 1 satisfy our notion of repair. $\square$

3 Framework of repair diversification

Distance & Cost. We are to define a *distance* function d to measure the dissimilarity between two instances carrying the same set of tuples (identifiers), with modified attribute values. Following former works [3, 5, 13], we use *weight* to reflect the confidence of the accuracy of data. These weights can be manually placed by the users, be collected through analysis of existing data, or be set to an equal value by default; our approach does not depend on a particular setting. Specifically, a weight $w(t, A) \in [0, 1]$ is attached to each attribute A of each tuple t, *i.e.,* $t[A]$. We assume that two instances carrying the same set of tuple identifiers, *e.g.,* instance I and its repair I_r, or two repairs $I_r, I_{r'}$ of I, have the same weight $w(t, A)$ for $t[A]$. Distance function d is defined as follows:

$$d(I_1, I_2) = \sum_{i\in[1,n],j\in[1,m]} w(t_i, A_j) \times \triangle(I_1(t_i, A_j), I_2(t_i, A_j))$$

I_1, I_2 are instances of schema $R(A_1, \ldots, A_m)$, both carrying a set of tuple identifiers $\{t_1, \ldots, t_n\}$. $\triangle(a, b) = 0$ if $a = b$, otherwise $\triangle(a, b) = 1$, where a (resp. b) is either a constant or a variable.

Example 3: Considering the weights presented in Fig. 1, the distance between r_3, r_4 is 0.8+0.8+0.3+0.3=2.2. Note that $v^{(t_2,Zip)}$ in $t_2[Zip]$ of r_3, r_4 does not increase their distance: any new value that can be assigned to $v^{(t_2,Zip)}$ in r_3 can also be used in r_4, and vice versa. □

The function d, when applied to an instance I and its repair I_r, measures the *cost* of I_r. defined as follows:

$$c(I_r) = d(I, I_r)$$

Example 4: r_1, r_2, r_3, r_4 have a cost of 1.6, 1.3, 1.2, 1.2, respectively. □

Since I is clear from the context, below we write repair I_{r_k} of I as r_k. We use $U_{(I,\Sigma)}$ to denote the space of all possible repairs of I *w.r.t.* a set Σ of FDs. To formalize repair diversification problem, we introduce two natural diversification objective functions of the form $f(S, c(\cdot), d(\cdot,\cdot))$, where $S \subseteq U_{(I,\Sigma)}$, and $c(\cdot)$, $d(\cdot,\cdot)$ is the above mentioned cost and distance function, respectively. $c(\cdot)$ and $d(\cdot,\cdot)$ are assumed to be fixed here; we hence abbreviate the function as $f(S)$.

(1) **Diversification function** $f_d(S)$. On a set $S = \{r_1, \ldots, r_k\}$ of k repairs, one natural bi-criteria objective concerns the difference between the cost sum and the distance sum of the repairs in S, which is encoded as follows:

$$f_d(S) = (1 - \lambda) \sum_{r_i \in S} c(r_i) - \frac{2 \cdot \lambda}{(k-1)} \sum_{r_i, r_j \in S, i<j} d(r_i, r_j).$$

The distance sum is scaled down with $\frac{2}{(k-1)}$, since there are $\frac{k \cdot (k-1)}{2}$ numbers for distance sum, while only k numbers for cost sum. Here λ is a parameter defining a trade-off between repair cost and distance; preference to distance increases as λ increases. Observe the following. (1) As λ increases, it tends to introduce large-cost repairs for the optimization of $f_d()$, since their overhead in cost is outweighed by their distances to small-cost repairs. However, repair diversification aims to pick diversified repairs from low-cost ones; large-cost repairs that rarely occur in practice should be banned. (2) It is generally infeasible to avoid large-cost repairs by setting an upper bound for repair cost; for a given number a, it is NP-complete to determine whether there exists a repair r_i such that $c(r_i) \leq a$. Putting these together, we find λ used in $f_d()$ is typically a small number, *e.g.,* $\lambda \in$ [0, 0.3]. We will further discuss this through experiments (Section 5).

(2) **Diversification function** $f_m(S)$. Our second bi-criteria objective concerns the maximum difference between the cost and the distance of the selected set :

$$f_m(S) = (1 - \lambda) \max_{r_i \in S} c(r_i) - \lambda \min_{r_i, r_j \in S, i<j} d(r_i, r_j),$$

As shown in experiments, $f_m()$ can better adapt to different settings of $\lambda \in$ [0,1].

Example 5: When $\lambda = 0.3$, $(f_d(\{r_1, r_3, r_4\}) = 0.64) < (f_d(\{r_2, r_3, r_4\}) = 0.73)$, while $(f_m(\{r_1, r_3, r_4\}) = 0.46) > (f_m(\{r_2, r_3, r_4\}) = 0.4)$. □

Repair diversification problem. We next state the repair diversification problem. Given a positive integer k, a parameter λ, an inconsistent instance I *w.r.t.* a set Σ of FDs, a repair diversification objective function $f(S)$, it is to find a set S_k^* of k repairs, such that $S_k^* = argmin_{\substack{S \subseteq U_{(I,\Sigma)} \\ |S|=k}} f(S)$.

Specifically, we denote the proposed repair diversification problem by MindiffP (resp. MinmaxP) when $f(S) = f_d(S)$ (resp. $f_m(S)$).

Note that an optimal solution of MindiffP may be far from the optimum *w.r.t.* MinmaxP and vice versa. Also observe that neither $f_d(S)$ nor $f_m(S)$ satisfies the *stability* property [11]: $f_d(S)$ ($f_m(S)$) *cannot* guarantee that $S_k^* \subset S_{k+1}^*$. Therefore, the optimal solution cannot be constructed incrementally with the increase of k, which necessarily complicates the optimization of MindiffP (MinmaxP).

Not surprisingly, MindiffP (MinmaxP) is intractable; it includes optimal repair computation [13] as a special case.

Theorem 1: *The* MindiffP *(resp.* MinmaxP*) problem is NP-complete.* □

4 Finding diversified repairs

We present algorithms for diversification problems MindiffP and MinmaxP. In light of the intractability, our algorithms are necessarily heuristic. The first set of algorithms relate repair diversification problems to known techniques for facility dispersion problems (Section 4.1). These algorithms, however, may suffer from both effectiveness and efficiency. We then present the second set of algorithms (Section 4.2). These algorithms embed repair computation into the framework of repair diversification, and therefore have the early termination property.

4.1 Baseline algorithms

Algorithm BMindiff**.** Given a positive integer k, a parameter λ and an inconsistent relation I *w.r.t.* a set Σ of FDs, BMindiff is designed for problem MindiffP, aiming at picking k repairs to minimize the objective function $f_d(S)$ (Section 3). In a nutshell, BMindiff works in two steps. (1) It randomly computes a set S of n ($n \geq k$) repairs. This can be achieved by employing existing techniques, *e.g.,* repair sampling. (2) It selects a set S_k of k repairs from S. To do so, BMindiff relates the diversification objective $f_d()$ to the objective of the well-known *Maximum Sum Dispersion problem* (MAXSUMDISP) [12]; this is a technique commonly used for optimization problems [11]. Recall that the objective of MAXSUMDISP problem is to select k points from a set of n points, such that the sum of all pairwise distances between these k points is maximized.

We show that an instance of MindiffP can be transformed to an instance of MAXSUMDISP. Given a set S of n repairs, we construct a set S' of n points, in which each point v_i represents a repair $r_i \in S$. The objective function of MAXSUMDISP is $f'_d(S') = \sum_{v_i, v_j \in S', i<j} d'(v_i, v_j)$, where $d'(v_i, v_j)$ is the distance between v_i, v_j. Herein, for two points $v_i, v_j \in S'$, we define the distance between them: $d'(v_i, v_j) = \frac{2 \cdot \lambda}{(k-1)} d(r_i, r_j) - \frac{1-\lambda}{(k-1)} (c(r_i) + c(r_j))$, where $d(\cdot,\cdot)$, $c(\cdot)$ is the distance and cost function defined on repairs (Section 3). For a set S'_k of k points, it can be verified that $f'_d(S'_k) = \frac{2 \cdot \lambda}{(k-1)} \sum_{r_i, r_j \in S_k, i<j} d(r_i, r_j) - (k-1) \cdot \frac{1-\lambda}{(k-1)} \sum_{r_i \in S_k} c(r_i) = -f_d(S_k)$; each $c(r_i)$ is counted exactly k -1 times in the sum. Therefore, S'_k is the set of points that maximizes the objective function of

MAXSUMDISP if and only if S_k is the set of repairs that minimizes the objective function of MindiffP. Given this reduction, we can employ known techniques for MAXSUMDISP in the second step of BMindiff. Although MAXSUMDISP is proved to be NP-complete, various heuristic and approximation algorithms have been proposed to solve MAXSUMDISP (see *e.g.,* [12, 14]).

Remark. (1) The bottleneck of BMindiff in terms of effectiveness lies in its first step, in which n candidate repairs are generated by considering neither cost nor diversity. Although BMindiff may simulate the known best algorithms for MAXSUMDISP in its second step, it still suffers from this significant limitation. (2) To improve the effectiveness of overall algorithm, BMindiff has to pick a $n \gg k$ in its first step, however, at the cost of efficiency.

Algorithm BMinmax**.** Taking the same input as BMindiff, BMinmax deals with problem MinmaxP, aiming to select k repairs that minimize the objective function $f_m(S)$ (Section 3). Just as BMindiff, BMinmax first randomly generates n repairs from the repair space, and then select k repairs from these repairs. To do so, BMinmax links the objective of MinmaxP to *Maximum Min Dispersion problem* (MAXMINDISP) [14]. For space limitation, we omit the details here.

4.2 Early termination heuristics

We are about to present algorithms Mindiff and Minmax for problems MindiffP and MinmaxP, respectively. These algorithms combine repair computation into repair diversification: instead of random repair generation adopted in BMindiff (BMinmax), Mindiff (Minmax) employs diversification objective to guide repair generation. This helps find repairs that are beneficial to diversification, and simultaneously avoids unnecessary computation. To facilitate repair generation, we first present a repairing algorithm, referred to as Genrepair. Genrepair is based on the sampling algorithm presented in [4], with non-trivial improvement by incremental techniques. It can produce different repairs of an inconsistent instance I, by taking as input different sorting of cells in I. As will be seen later, a tricky technique concerns sorting cells in Mindiff (Minmax) to produce different inputs for Genrepair, such that desirable repairs can be generated.

Algorithm Genrepair**.** Genrepair is a common basis of Mindiff and Minmax. It takes as input a list of cells of an inconsistent instance I *w.r.t.* a set Σ of FDs, and produces a repair of I after processing all cells one by one in the list order.

Genrepair has to detect conflicts that are among cells processed so far, and that are deduced by FD reasoning. To this end, Genrepair employs the notion of *equivalence class* (EC) [3, 4]. Each EC is associated with an attribute: an EC e^A on attribute A is a set of cells of the form $t_i[A]$; these cells have a same value in the repair generated by Genrepair. We use the following notations. (1) Any cell c belongs to exactly one EC at any time, denoted by $ec(c)$; cell c is initially in the EC $\{c\}$, *i.e.,* a singleton set containing itself. (2) Any EC e^A is associated with a value that is assigned to all cells in e^A in the generated repair, denoted by $val(e^A)$; $val(e^A)$ is initially NULL, which implies that the value is not yet determined. (3) We denote by ξ the set containing all equivalence classes (ECs).

After initializing the set ξ of ECs (line 1), Genrepair processes cells one by one. (1) when the current cell $t_i[B]$ has already been put into an EC containing other cells, and the value of that EC is determined, Genrepair modifies $t_i[B]$ if its value is different from the value chosen for that EC (lines 4-5). (2) Otherwise, Genrepair has to determine the value for the EC that $t_i[B]$ belongs to, and maintains ξ for further processing (lines 6-11). Genrepair first tries to use the value of $t_i[B]$ as the value of its EC (line 7), while this may cause its EC to be merged with other EC having the same value (line 8). To address this issue, Genrepair calls function Merge (line 9). If the merging fails (NULL is returned), Genrepair turns to introduce a new value as the value of $t_i[B]$ and its EC (line 10). Otherwise, Merge returns a modified set of ECs, accepted by Genrepair (line 11).

Algorithm 1: Genrepair

```
input : a list $L$ of cells of inconsistent instance $I$ w.r.t. a set $\Sigma$ of FDs.
output: a repair of $I$.
1  initialize $\xi$: each cell $c$ is in the EC $\{c\}$;
2  while $L$ is not empty do
3      remove the first cell $c$ from $L$; suppose $c = t_i[B]$ and $ec(t_i[B]) = e^B$;
4      if $|e^B| \neq 1$ and $val(e^B) \neq NULL$ then
5          if $t_i[B] \neq val(e^B)$ then $t_i[B] := val(e^B)$;
6      if $(|e^B| \neq 1$ and $val(e^B) = NULL)$ or $(|e^B| = 1)$ then
7          $val(e^B) := t_i[B]$;
8          if there exists $e'^B \in \xi$ such that $val(e'^B) = t_i[B]$ then
9              $\xi' :=$ merge($e^B, e'^B, \xi, I$) ;
10             if $\xi' = NULL$ then introduce a new value to $t_i[B]$ and $val(e^B)$ in $\xi$ ;
11             else $\xi := \xi'$ ;
```

Function Merge

```
input : ECs $e^B, e'^B$, set $\xi$ of ECs and instance $I$
output: a modified $\xi$ if merging of $e^B, e'^B$ succeeds, or NULL
1  put ($e^B$, $e'^B$) into an empty list $K$ ;
2  while $K$ is not empty do
3      remove the first element, say ($e^C$, $e'^C$), from $K$;
4      if $val(e'^C) \neq NULL$ and $val(e^C) \neq NULL$ and $val(e'^C) \neq val(e^C)$ then
       return NULL;
5      remove $e^C$, $e'^C$ from $\xi$, and add to $\xi$ a new EC $e''^C := e^C \cup e'^C$, where
       $val(e''^C) := val(e'^C)$ if $val(e^C)$ = NULL, otherwise $val(e''^C) := val(e^C)$ ;
6      while there exists (1) $X \to A \in \Sigma$ and (2) tuples $t_1, t_2$ in $I$ such that (1)
       $C \in X$, (2) $t_1[C] \in e^C$, $t_2[C] \in e'^C$, (3) $\forall D \in X$, $ec(t_1[D]) = ec(t_2[D])$ and
       (4) $ec(t_1[A]) \neq ec(t_2[A])$ do
7          put ($ec(t_1[A])$, $ec(t_2[A])$) into $K$ if ($ec(t_1[A])$, $ec(t_2[A])$) $\notin K$;
8  return $\xi$;
```

The difficulty of function Merge arises from the fact that merging of two ECs may cause mergings of other ECs iteratively. To this end, Merge maintains a list K of pairs of ECs to be merged, and continues until this list is empty. The merging of two ECs fails when the values of these ECs are not NULL, and are different (line 4). Otherwise, Merge merges two ECs by replacing them with a new EC that is equal to their union (line 5). Merge then collects in K all pairs of ECs to be merged (lines 6-7). To improve the efficiency, Genrepair provides an *incremental* version of the *chase* approach. Specifically, given two ECs on attribute C, Merge only checks tuples t_1, t_2 when $t_1[C], t_2[C]$ originate from the two ECs respectively, and applies FDs that have C as left-hand side attribute. Merge returns a modified ξ when all merging succeeds (line 8).

Example 6: Recall repairs in Fig 1. Genrepair produces repair r_1 when taking as input cells first sorted in tuples t_1, t_2, t_3 and then in attributes $ID, Name, Zip, City$, while r_2 is generated when sorting cells first in attributes $ID, Name, City, Zip$, and then in tuples t_1, t_2, t_3. □

Complexity. The number of iterations in Genrepair is equal to the number of cells, *i.e.,* $m \cdot n$, where m is the number of attributes and n is the number of tuples. In each iteration, Genrepair may call Merge to merge ECs when necessary, and the worst case complexity of Merge is $O(m \cdot n \log n)$. Therefore, Genrepair takes $O(m^2 \cdot n^2 \log n)$ in the worst case. Note that (1) cells not involved in any FD can be safely skipped; and (2) ECs are incrementally maintained in Genrepair and chase is incrementally conducted in Merge. The actual complexity of Genrepair is hence much better than its worst case complexity.

Remark. Compared to [4], Genrepair improves equivalence class techniques by maintaining all ECs in an incremental way, while [4] requires to rebuild ECs from scratch when new cell is processed. Combining this with the incremental chase adopted in Merge, Genrepair outperforms [4] in the efficiency of producing a single repair, as will be verified by experiments (Section 5).

Algorithm Mindiff. We then present algorithm Mindiff for MindiffP. Combining diversification methods, this algorithm is based on the following observations. (1) Genrepair postpones resolution of FD violations until it has to do so. For example, suppose cells $t_1[A], t_1[B], t_2[A], t_2[B]$ violate an FD $A \rightarrow B$, Genrepair will resolve this violation by modifying the value of the last cell among this four cells in the input list. (2) Genrepair uses as the value of each EC the value of the first cell of that EC in the input list, when this does not cause merging of ECs to fail. (3) It is preferable *not* to modify cells that have been modified by other repairs, when producing a repair r for diversity. Intuitively, such modifications increase the cost of r, while may fail to simultaneously increase the distances between r and other repairs, and hence hinder repair diversification.

Taken together, these tell us the following. (1) When calling Genrepair with an input list L, it is better to put at the front of L those cells that we prefer not to modify, while put at the rear of L those cells that we prefer to modify. (2) When generating diversified repairs, cells modified in former repairs should be adjusted towards the front of the input list, to reduce the possibility of modifying these cells in future repairs.

Example 7: We adjust the list for producing r_1 in Example 6, by putting at its front the three cells that are modified in r_1, *i.e.,* $t_2[City]$, $t_3[Name]$, $t_3[City]$, Genrepair generates r_3 when taking this adjusted list as its input. □

We present details of Mindiff. It employs Genrepair to compute S_k of k repairs by providing different list L as input (lines 1-8). Specifically, function $w'(c)$

measures the preference of modifying a cell c, say $t[A]$, in the current repair (line 3). In $w'(c)$, $cnt(S_k, t[A])$ denotes the number of repairs in S_k that have modified cell $t[A]$, and $vio(t[A])$ is computed as the number of tuples violating t on A. Intuitively, it is preferable to modify cell c with a small $w'(c)$ value, *i.e.,* cell with a small cost, modified by less repairs, and involved in more FD violations; this follows the semantics of repair diversification: finding diversified repairs among low-cost ones. In addition to this idea, Mindiff also uses parameter $\alpha \in [0, 1]$ to follow a greedy randomized approach, in producing input list L for Genrepair (lines 5-7). Note that max (resp. min) in line 6 may require recomputation after cells are removed from P. When $\alpha = 0$, cells are ordered in L by decreasing $w'()$ values; when $\alpha = 1$, L is purely constructed at random; otherwise, each time a random cell, among cells whose $w'()$ values are within a range, is added to list L. Complementary to greedy strategies, randomization is introduced to Mindiff by following this approach; this allows for trying different sorting of cells.

Algorithm 2: Mindiff

```
   input  : k, $\lambda$ and inconsistent relation I w.r.t. a set $\Sigma$ of FDs.
   output: a set $S_k$ of k repairs for diversification objective $f_d()$.
1  $S_k := \emptyset$;
2  while $|S_k| \le k$ do
3      compute $w'(c)$ for each cell c, say $t[A]$, where $w'(c) = (w(t,A) \cdot (1-\lambda) + \frac{\lambda \cdot cnt(S_k, t[A]) \cdot w(t,A)}{(|S_k|-1)})/(vio(t[A])+1)$ if $|S_k| > 1$; otherwise $w'(c) = \frac{w(t,A)}{vio(t[A])+1}$ ;
4      suppose P is a set containing all cells in I; initialize an empty list L;
5      for i:=1; $i \le |P|$; i++ do
6          randomly pick c from P such that $w'(c) \ge max - \alpha \cdot (max - min)$, where max (resp. min) is the maximum (resp. minimum) $w'()$ value in P ;
7          remove c from P, and add c to L;
8      r := Genrepair (L); add r to $S_k$ if $r \notin S_k$ ;
9  repeat
10     pick repair r such that $f'_d(r)$ is maximum among all $f'_d()$ values of unlabeled repairs in $S_k$, where $f'_d(r) = (1-\lambda) \cdot c(r) - \frac{\lambda}{(k-1)} \sum_{r, r_i \in S_k, r_i \ne r} d(r, r_i)$ ;
11     old := $f_d(S_k)$ ; $S_k := S_k \backslash \{r\}$ ;
12     compute and add a repair r' to $S_k$, by following the same way as lines 2-8 ;
13     if old $\le f_d(S_k)$ then replace r' by r in $S_k$ and label r;
14     else label r' and remove all other labels in $S_k$.
15 until termination condition is satisfied;
```

Finally, Mindiff performs swaps between repairs in S_k and some newly generated repairs, in order to improve objective function $f_d()$ (lines 9-15). Specifically, Mindiff measures the contribution of repair r to $f_d(S_k)$ as $f'_d(r)$, and tests repairs in decreasing order of $f'_d()$ values (line 10). When trying to replace a repair r, Mindiff generates a new repair r' by running methods for producing S_k on the set $S_k \backslash \{r\}$, *i.e.,* employing lines 2-8 by setting the initial S_k as $S_k \backslash \{r\}$. Repair r' that improves diversification objective is kept in S_k; this will cause $f'_d(\cdot)$ to be recomputed for all repairs in line 10. For each repair whose testing does not change S_k, Mindiff attaches a label to it, such that it will not be tested again until S_k changes; such label is also attached to each new repair added to S_k. The termination condition (line 15) can be set based on the number of new repair computations, or based on the gain of swaps, *e.g.,* terminates when the average improvement in $f_d(\cdot)$ of the last n swaps is below some threshold.

Complexity. Computing $w'(\cdot)$ for all cells (line 3) and then sorting cells based on $w'(\cdot)$ (lines 5-7) take $O(m \cdot n)$ and $O(m \cdot n \log(m \cdot n))$, respectively, where m is the number of attributes and n is the number of tuples. The cost of a repair modifying p cells can be computed in $O(p)$, and the distance between two repairs modifying p and q cells respectively can be computed in $O(p+q)$. Then, suppose l is the maximum number of modified cells in any repair, computations of $f'_d(\cdot)$ take $O(k^2 \cdot l)$ (line 10). Typically, $l \ll n \cdot m$. Therefore, the overall complexity of Mindiff is governed by the running times of line 8 and line 12 that compute repairs, with the worst case complexity of $O(m^2 \cdot n^2 \log n)$.

Algorithm Minmax. Taking the same input as Mindiff, Minmax is designed for problem MinmaxP, aiming to optimize objective $f_m()$. Minmax follows the same framework as Mindiff, we therefore only highlight the differences between them.

(1) Minmax computes the value of $\frac{w(t,A)}{vio(t[A])+1}$ for each cell $t[A]$, and fixes top $\beta\%$ cells at the front of the list L when calling Genrepair. Intuitively, this reduces the possibility of modifying these cells and hence avoids generating repairs with large costs. In our experiments, we set $\beta\% = 10\%$.

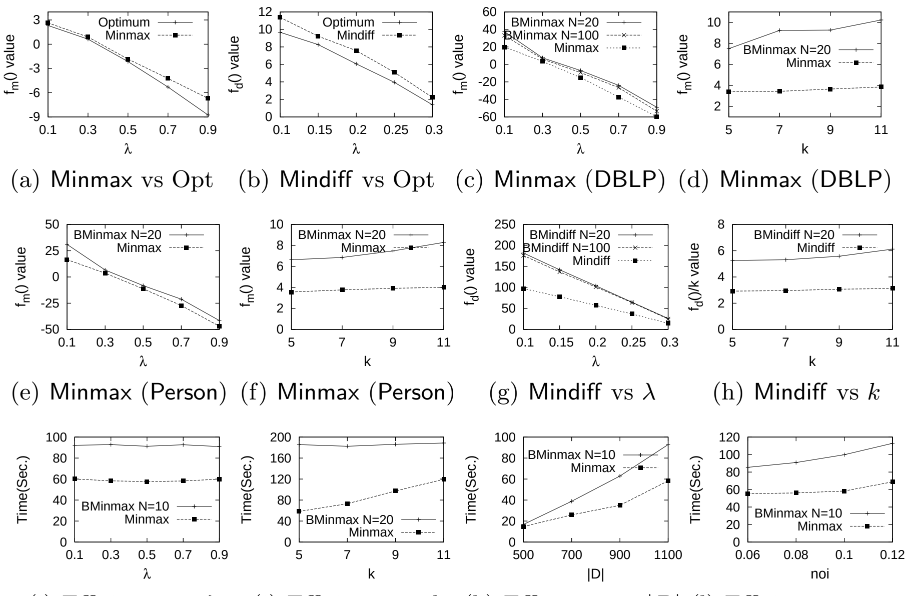
(2) Minmax uses function $f'_m(r)$ to measure the contribution of repair r to $f_m(S_k)$ in the swap stage, and tests repairs in decreasing order of $f'_m()$ values. $f'_m(r) = max((1-\lambda) \cdot (c(r) - max_c(S_k \backslash \{r\})), \lambda \cdot (min_d(S_k \backslash \{r\}) - min_{r' \in S_k, r' \neq r} d(r, r')))$, where $max_c(S) = max_{r_i \in S} c(r_i)$, and $min_d(S) = min_{r_i, r_j \in S, i<j} d(r_i, r_j)$. Intuitively, repair r with the maximum $f'_m(r)$ value is the "worst" repair in terms of the optimization of $f_m(S_k)$.

5 Experimental study

Experimental setting. We use a PC with 3.3GHz Intel Duo CPU, 4GB memory and Windows 7. All experiments report the average over five runs.

Data. (1) DBLP data is extracted from dblp bibliography. (2) Synthetic Person data extends the relation in Fig. 1, and we populate the relation using a data generator. Each dataset in the experiments is produced by introducing noises, controlled by two parameters: (a) $|D|$: the number of tuples; and (b) $noi\%$: the noise rate, which is the ratio of the number of violating cells *w.r.t.* FDs to the total number of cells in the dataset. We randomly assign a weight to each cell.

Algorithms. We implement the following algorithms for problems MindiffP and MinmaxP, all in C++: (1) baseline algorithms BMindiff and BMinmax; and (2) Mindiff and Minmax. BMindiff (resp. BMinmax) randomly computes a set S of repairs, from which k repairs is then found. We implement repair sampling technique [4] to generate S, with a parameter N=$|S|$. To select k repairs from S in BMindiff (resp. BMinmax), we implement the facility dispersion technique p-resented in [12] (resp. [14]). In Mindiff and Minmax, we set the randomization parameter α=0.3, and compute k new repairs in repair swapping phase.



(a) Minmax vs Opt (b) Mindiff vs Opt (c) Minmax (DBLP) (d) Minmax (DBLP)

(e) Minmax (Person) (f) Minmax (Person) (g) Mindiff vs λ (h) Mindiff vs k

(i) Efficiency vs λ (j) Efficiency vs k (k) Efficiency vs $|D|$ (l) Efficiency vs noi

Fig. 2. Experimental Results

Exp-1. We verify the effectiveness of Minmax (resp. Mindiff) against the optimal result minimizing $f_m()$(resp. $f_d()$). Due to the intractability, we find the optimal result by enumerating all possible ones. Using Person data, fixing $|D| = 100$, $noi\% = 6.5\%$ and $k = 5$, it takes about 8 minutes to find the optimal result; this approach does not scale with data size. Fig. 2(a) shows results of $f_m()$, by varying λ from 0.1 to 0.9. Note that $f_m()$ values decrease with increasing λ; as λ increases, the minuend decreases while the subtrahend increases in $f_m(S)$ for any given set S. We find gaps between $f_m()$ values of Minmax and the optimal ones are about [15%, 35%] of the optimal ones, and Minmax only takes 2 seconds.

In the same setting, Fig. 2(b) shows results of $f_d()$ by varying λ from 0.1 to 0.3. We see gaps between $f_d()$ values of Mindiff and optimal ones are about [17%, 32%] of the optimal ones. Results on larger λ are omitted, because we find when $\lambda = 0.4$, the maximum repair cost in the optimal solution increases sharply, more than 150% of that when $\lambda = 0.3$. This shows that it favors large-cost repairs in minimizing $f_d()$; their overhead in cost sum is outweighed by their large distances to small-cost repairs in distance sum as λ increases. However, this significant rise in repair cost implies that an optimal solution for $f_d()$ deviates from a good result for repair diversification. To avoid such disturbance, λ has to be a small number when used in $f_d()$.

Exp-2. We verify the effectiveness of Minmax against BMinmax, using DBLP ($|D| = 1100$ and $noi\% = 10\%$). In Fig. 2(c), we fix $k = 5$, vary λ, and use N=20, 100 for BMinmax. We see the following. (1) Minmax significantly outperforms BMinmax even when $N = 100$, which requires more repair computations than Minmax by orders of magnitude. Specifically, Minmax reduces $f_m()$ values by 10% to 55%, compared to BMinmax ($N = 100$). (2) As expected, the performance of BMinmax improves as N increases; however, the improvement is minor. The increased sample number is still too small compared to the exponential number of repairs, although it is already very costly to compute these repairs. (3) The gap between $f_m()$ values for Minmax and BMinmax decreases as λ increases. This is because preference to distance increases as λ increases, and it is relatively easy to pick a small number (k=5) of dissimilar repairs from an exponential repair space at random. Nevertheless, Minmax still outperforms BMinmax by about 10% when λ=0.9, and the cost sum of the k repairs in Minmax is about 75% of that in BMinmax (not shown). This demonstrates that Minmax is effective in finding diversified repairs from small-cost ones.

By fixing $\lambda = 0.3$ and varying k, Fig. 2(d) shows $f_m()$ values. Minmax consistently outperforms BMinmax. Indeed, the performance of Minmax is stable, while BMinmax degrades as k increases. The solution is incrementally constructed in BMinmax by its employed facility dispersion technique, *i.e.,* $S^*_k \subset S^*_{k+1}$, where S^*_k is the solution with k repairs. However, $f_m()$ does not have this property. Minmax addresses this issue by introducing randomization and by performing swaps between repairs, which is proved to be effective.

We then verify the effectiveness of Minmax using Person ($|D| = 1000$ and $noi\%$ = 6.5%). The results are reported in Fig. 2(e) and Fig. 2(f), in the same setting as Fig. 2(c) and Fig. 2(d), respectively. These results confirm our observations on DBLP data. For instance, Minmax outperforms BMinmax in reducing $f_m()$ value by 9% to 45%, as shown in Fig. 2(e).

Exp-3. Using DBLP ($|D| = 1100$ and $noi\% = 10\%$), the effectiveness of Mindiff is verified against BMindiff (N=20, 100), shown in Fig. 2(g) (k=5) and 2(h) (λ=0.3). Note that (1) λ is varied from 0.1 to 0.3 in Fig. 2(g). Mindiff still outperforms BMindiff for large λ; however, as noted earlier, those settings of λ deviate from the purpose of repair diversification; and (2) we show $f_d()/k$ values in Fig. 2(h), *i.e.,* the difference between the average cost and the average distance of repairs; these values are comparable for different k. The results show that Mindiff performs well. $f_d()$ values of Mindiff are only about [50%, 58%] of those of BMindiff; the performance of Mindiff is stable as k increases.

Exp-4. We evaluate the efficiency and scalability of Minmax against BMinmax (N=10, 20) on DBLP. We set λ=0.3, k=5, $|D|$= 1100 and $noi\%$= 10% by default, and vary one parameter in each of Fig. 2(i), 2(j), 2(k) and 2(l), respectively. For space limitation, we omit the results of Minmax on other dataset, and omit the results of Mindiff, since they are similar. We find Minmax outperforms BMinmax in terms of efficiency and scalability. (1) Fig. 2(i) shows that running times of all algorithms are not sensitive to λ, as expected. In this setting, numbers of repair computations conducted in Minmax (initial phase and swap phase) and BMinmax are the same. The results show that the time for Minmax is about 70% of that for BMinmax; this implies that Minmax is more efficient in computing a single repair, due to incremental computation techniques adopted in Genrepair. (2) Fig. 2(j) shows that running times of BMinmax are the same as k increases, since it always computes N=20 repairs and the time for selecting k repairs from these repairs is trivial. The running time of Minmax is almost linear in k; k controls the number of repair computations conducted in Minmax and as stated in the complexity analysis, time for repair computations governs the overall time. (3) Running times of all algorithms in Fig. 2(k) increase as $|D|$ increases, as expected. We see Minmax scales better with $|D|$ than BMinmax. (4) The increase of $noi\%$ also has a negative impact on the running times in Fig. 2(l), as expected. We find that Minmax scales well with $noi\%$.

6 Conclusions

We have presented a formal framework for repair diversification problems, established the complexity and developed algorithms for these problems, and experimentally verified our approach. We are currently experimenting with more real-life data sets to test the usefulness of the proposed algorithms, exploring different diversification objective functions, and studying optimization techniques to further improve our algorithms.

References

1. M. Arenas, L. Bertossi, J. Chomicki. Consistent query answers in inconsistent databases. PODS, 1999.
2. L. Bertossi. Database repairing and consistent query answering. Morgan & Claypool Publishers, 2011.
3. P. Bohannon, W. Fan, M. Flaster, R. Rastogi. A cost based model and effective heuristic for repairing constraints by value modification. SIGMOD, 2005.
4. G. Beskales, I. Ilyas, L. Golab, Sampling the repairs of functional dependency violations under dard constraints, VLDB, 2010.
5. G. Cong, W. Fan, F. Geerts, X. Jia, S. Ma. Improving data quality: Consistency and accuracy. VLDB, 2007.
6. X. Chu, I. Ilyas, P. Papotti. Holistic data cleaning: Putting violations into context. ICDE, 2013.
7. M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. Ilyas, M. Ouzzani, N. Tang. NADEEF: a commodity data cleaning system. SIGMOD, 2013.
8. M. Drosou, E. Pitoura. Search result diversification. SIGMOD Record 39(1): 41-47, 2010.
9. W. Fan, F. Geerts. Foundations of data quality management. Morgan & Claypool Publishers, 2012.
10. W. Fan, J. Li, S. Ma, N. Tang, W. Yu. Towards certain fixes with editing rules and master data. VLDB, 2010.
11. S. Gollapudi, A. Sharma. An axiomatic approach for result diversification. WWW, 2009.
12. R. Hassin, S. Rubinstein, A. Tamir. Approximation algorithms for maximum dispersion. Operations Research Letters, 21(3): 133-137, 1997.
13. S. Kolahi, L. Lakshmanan. On approximating optimum repairs for functional dependency violations. ICDT, 2009.
14. S. Ravi, D. Rosenkrantz, G. Tayi. Heuristic and special case algorithms for dispersion problems. Operations Research, 42(2): 299-310, 1994.
15. M. Yakout, A. Elmagarmid, J. Neville, M. Ouzzani, I. Ilyas. Guided data repair. VLDB, 2011.