Enriching Data Imputation with Extensive Similarity Neighbors

Shaoxu Song[§] Aoqian Zhang[§] Lei Chen[‡] Jianmin Wang[§]

[§]KLiss, MoE; TNList; School of Software, Tsinghua University, China
{sxsong, jimwang}@tsinghua.edu.cn zaq13@mails.tsinghua.edu.cn

[‡]The Hong Kong University of Science & Technology, Hong Kong leichen@cse.ust.hk

ABSTRACT

Incomplete information often occur along with many database applications, e.g., in data integration, data cleaning or data exchange. The idea of data imputation is to fill the missing data with the values of its neighbors who share the same information. Such neighbors could either be identified certainly by editing rules or statistically by relational dependency networks. Unfortunately, owing to data sparsity, the number of neighbors (identified w.r.t. value equality) is rather limited, especially in the presence of data values with *variances*. In this paper, we argue to extensively enrich similarity neighbors by similarity rules with tolerance to small variations. *More* fillings can thus be acquired that the aforesaid equality neighbors fail to reveal. To fill the missing values *more*, we study the problem of maximizing the missing data imputation. Our major contributions include (1) the NP-hardness analysis on solving and approximating the problem, (2) exact algorithms for tackling the problem, and (3) efficient approximation with performance guarantees. Experiments on real and synthetic data sets demonstrate that the filling accuracy can be improved.

1. INTRODUCTION

Incomplete data (a.k.a. null or missing data) are very prevalent, owing to incomplete entry, inaccurate extraction or heterogeneous schemas, e.g., in Web autonomous databases [18]. Existing techniques on imputing missing data are mainly in two categories, (1) statistical-based [12] and (2) rule-based [7]. Both techniques share a similar idea of filling the missing data with the values of its neighbors (identified by statistical relational dependencies/editing rules) who share the same information. The major problem in such data imputation, as indicated in [12], is the *sparsity* of data that many reasonable value combinations are not observed. In other words, no sufficient neighbors are available for imputing.

The *variety* of data (especially the Web data with various information formats) further prevents finding sufficient neighbors. We note that, in existing relational dependency networks [12] or editing rules [7], the neighbors are identified by value equality relationships, called *equality neighbors*. These existing notations fall short in capturing the neighbors whose data values are with small variations (very prevalent in Web data). The imputation upon such limited neighbors could barely be performed. (See examples, surveys below and more discussions in Section 7.)

In this study, we identify a more extensive class of *similarity neighbors*, with value similarity relationships identified by similarity rules [14]. By tolerance of small variations, the enriched (similarity) neighbors can fill more missing data that are not revealed by the very limited equality neighbors.

Table 1: Example of incomplete data

	Name	Street	Address
t_1	Susan Frank	Jordan Road	–
t_2	Susan L Frank	JR	–
t_3	Terry Michel	JR	–
t_4	Susan L Frank	Jordan Road	No402 Jordan Rd
t_5	Susan L Frank	Jordan Rd	#402 Jordan Rd
t_6	Terry K Michel	Jordan Rd	#531 Jordan Rd

Example 1. Table 1 illustrates an example of incomplete data. For instance, the value of t_1 on attribute Address is not available, i.e., a null cell denoted by t_1[Address] ='_'.

An editing rule [7] declared upon the functional dependency semantics, (Name, Street $\rightarrow$ Address), states that if two tuples t_i, t_j share equal Name and Street values, the missing t_j[Address] can be filled by the non-null t_i[Address].[1] As a neighbor of t_j, t_i is identified certainly (analogous to statistically below) by the editing rule, given their exactly equal Name and Street values. Unfortunately, none of the tuples in Table 1 share the same Name and Street values with t_1, i.e., t_1 has no equality neighbor and cannot be filled.

The statistical-based method considers the statistical distribution of values, e.g., between Name and Address in the relational dependency network [12]. It looks up the most probable fillings of null cells, referring to combination statistics. For instance, t_4, t_5 are identified statistically as the neighbors of t_2 referring to their equal Name. According to the statistics (on t_4, t_5), the most probable Address appearing together with Susan L Frank (in t_2[Name]) is either No402 Jordan Rd or #402 Jordan Rd. Again, since there is no combination relationship observed between Terry Michel on Name and any non-null value on Address, t_3 has no statistical-based equality neighbor, and t_3[Address] could not be filled.

[1] t_i with non-null t_i[Address] is regarded as reference data[7]

Proceedings of the VLDB Endowment, Vol. 8, No. 11

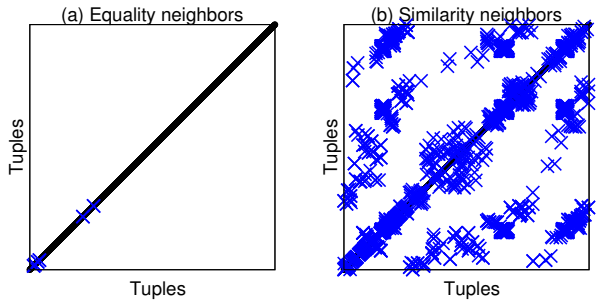

Figure 1: Tuple pairs in neighborhoods

Consider a similarity rule in the form of differential dependencies (DD) [14].

$$\text{DD}_1 : (\mathsf{Name}, \mathsf{Street} \rightarrow \mathsf{Address}, \langle [0,1], [0,9], [0,2] \rangle)$$

It states that if two tuples have similar Name (i.e., having Name value distance[2] in the range of $[0,1]$) and Street values (with distance in $[0,9]$), they must share similar Address values as well (having Address value distance in $[0,2]$).[3]

Tuples t_1 and t_4, sharing similar Name and Street values w.r.t. DD_1, are identified as similarity neighbors. No402 Jordan Rd of t_4 is thus suggested as a possible filling candidate to t_1[Address]. Similar filling candidate of t_3[Address] can also be obtained referring to the similarity neighborhood between t_3 and t_6 w.r.t. DD_1. □

Figure 1 reports a survey of equality and similarity neighborhoods, w.r.t. editing rules (based on equality) and similarity rules (DDs), respectively, in a real dataset Restaurant (see Section 6 for details). Each point in the figure denotes a tuple pair in neighborhood. As shown, the neighborhoods are significantly enriched, from equality to similarity. When some tuple value is missing, it is more likely to be filled from the more extensive similarity neighbors.

To explore similarity neighbors, existing method [19] computes a tuple-similarity which is indeed defined on value equality. Imputation in [19] or [20] considers only the most similar tuples with high tuple-similarities defined over all the attributes. In order to find and utilize more similarity neighbors, (1) DDs can address the value similarity with small value variations, which cannot be handled by value equality-based tuple-similarity; and (2) DDs could utilize more partial similarities on a subset of attributes between tuples (not necessarily the most similar ones that attributes are all similar). For example, $\text{DD}_2 : (\mathsf{Street} \rightarrow \mathsf{Address}, \langle [0,0], [0,3] \rangle)$ in Example 3 considers the partial similarities between tuples on a subset of Street and Address attributes, such as t_5 and t_6 in Table 1. Such similarity relationships are not considered by the tuple-similarity [19, 20], given the distinct Name values (not denoting the same people). Therefore, the similarity rule-based approach could utilize more similarity relationships between tuples. As mentioned, identifying more similarity neighbors is necessary for imputation, owing to data sparsity and variety. According to the experimental evaluation in Figure 9, applications such as entity reconciliation directly benefit from our similarity rule-based imputation.

[2]e.g., edit distance, see [13] for a survey of string similarity.
[3]Similarity rules with distance thresholds can either be specified by domain experts or discovered from data [15].

Challenges

While similarity neighbors bring more imputation opportunities, it comes with new challenges. (1) Different from the certain fixes by equality neighbors with editing rules [7], multiple candidates may be suggested by similarity neighbors for filling a cell. (2) The candidates for filling two null cells could be **incompatible** w.r.t. similarity rules, owing to the complex similarity relationships [4] (see examples on t_2 and t_3 in Example 3). Such incompatibility is not considered in the statistical-based imputation [12] either.

A practical question is thus whether all the null cells could be filled due to the incompatibility w.r.t. the similarity rules, namely *full filling* (Definition 3). If not, to what extend we can fill the null cells, known as *maximum filling* (Definition 2). Indeed, in light of the disability in imputing missing data (due to the aforesaid data sparsity and variety), it is naturally desirable to gain null-cell fillings as many as possible. As verified in the experiments in Section 6, the imputation performance (f-measure) could be significantly improved by gaining *more* fillings with the extensive similarity neighbors, compared to the equality-based barely-filled ones. Unfortunately, we find that maximizing the filling gain is Max-$\mathcal{SNP}$-hard, i.e., there exists $\varepsilon > 0$ such that achieving an approximation factor $(1-\varepsilon)$ for the maximum filling problem is NP-hard (see Sections 2 and 4 for details).

Contributions

In this paper, we propose a similarity rule based approach for imputing missing data. Our major contributions in this study are summarized as follows.

- We analyze the hardness of the similarity rule based imputation problem (Theorems 1 and 2) in Section 2.
- We present an approach for finding exact solutions in Section 3.
- We study the hardness of approximation (Theorem 3) and propose a heuristic for imputing in Section 4.
- We devise a randomized algorithm for returning maximal fillings, where an expected performance guarantee is obtained (Theorem 8) in Section 5.1.
- We improve the algorithm by ensuring a deterministic approximation factor instead of in expectation (Theorem 10) in Section 5.2.

Finally, we report an extensive experimental evaluation on both effectiveness and efficiency, over real and synthetic data sets, in Section 6. It is highlighted that the imputation accuracy is improved by considering similarity neighbors. We also demonstrate the improvement in the *record matching application*, after applying the proposed imputation.

Table 2 lists the frequently used notations in this paper. Proofs of theoretic result are included in technique report [1].

2. PROBLEM STATEMENT

In this section, we formalize the problem of data imputation with similarity rules.

[4]In essence, unlike equality, transitivity is not applicable to the similarity relationships, where similar (a,b) and similar (b,c) do not necessary imply similar (a,c).

Table 2: Notation

Symbol	Description
$\mathsf{dom}_I(A)$	values on attribute A in a relation I
I_A	tuples with missing values on attribute A
m	number of tuples in I_A
$\Delta(I'_A, I_A)$	set of filled cells, with filling gain $\lvert\Delta(I'_A, I_A)\rvert$
$\phi[A](t_1, t_2)$	local neighbor restriction between two tuples
$\Phi[A]$	a set of local neighbor restrictions
$\mathsf{can}(t[A])$	candidates for filling a cell $t[A]$
$\mathbf{E}[W]$	expectation of filling gain W
c	the maximum candidate size of a tuple in I_A
b	the maximum neighbor size of a tuple in I_A

2.1 Preliminaries

Similarity Rules

Consider a relation I with scheme $\mathcal{R}$. Let $\mathsf{dom}_I(A)$ denote all the values of an attribute A in I, i.e., $\mathsf{dom}_I(A) = \Pi_A(I)$.

For each attribute $A \in \mathcal{R}$, we associate a *similarity/distance metric*, d_A, which satisfies non-negativity, $d_A(a, b) \geq 0$; identity of indiscernibles, $d_A(a, b) = 0$ *iff* $a = b$; symmetry, $d_A(a, b) = d_A(b, a)$; where $a, b \in \mathsf{dom}_I(A)$ are values on attribute A. For example, the metric on a numerical attribute can be the absolute value of difference, i.e., $d_A(a, b) = |a - b|$. For a text attribute, we can adopt string similarity, e.g., *edit distance* (see [13] for a survey).

A *differential function* $\phi[A]$ on attribute A specifies a distance restriction by a range of metric distances over A. We say that two tuples t_1, t_2 in a relation I are *compatible* w.r.t. the differential function $\phi[A]$, denoted by $(t_1, t_2) \asymp \phi[A]$ or $(t_1[A], t_2[A]) \asymp \phi[A]$, if the metric distance of t_1 and t_2 on attribute A is within the range specified by $\phi[A]$, a.k.a. *satisfy/agree* with the distance restriction $\phi[A]$. As the metric is symmetric, it is equivalent to write $(t_2, t_1) \asymp \phi[A]$.

A differential function may also be specified on a set of attributes X, say $\phi[X]$, which denotes a pattern of differential functions (distance ranges) on all the attributes in X. We call $\phi[A]$ a *projection* on attribute A of $\phi[X], A \in X$.

A *differential dependency* (DD) [14] over $\mathcal{R}$ has a form $(X \to A, \phi[XA])$ where $X \subseteq R$ are determinant attributes, $A \in R$ is the dependent attribute, and $\phi[XA]$ is a differential function on attributes X and A. It states that any two tuples from $\mathcal{R}$ satisfying the differential function $\phi[X]$ must satisfy $\phi[A]$ as well, $\phi[X]$ and $\phi[A]$ are the projections of differential function $\phi[XA]$ on X and A, respectively.

A relation I of $\mathcal{R}$ *satisfies* a DD, denoted by $I \vDash (X \to A, \phi[XA])$, if for any two tuples t_1 and t_2 in I, $(t_1, t_2) \asymp \phi[X]$ implies $(t_1, t_2) \asymp \phi[A]$. We say a relation I satisfies a set Σ of DDs, $I \vDash \Sigma$, if I satisfies each DD in Σ.

Rule-based Data Imputation

A *null* cell in a tuple $t_1 \in I$ on attribute $A \in \mathcal{R}$ is denoted by $t_1[A] =_-$. It is regarded to be compatible with any other data w.r.t. distance restrictions, i.e., always having $(t_1, t_2) \asymp \phi[A]$. We consider an input with null cells $I \vDash \Sigma$.[5]

[5]For potential errors existing in non-null cells so that $I \nvDash \Sigma$, a data repairing step [4] can be applied first on the non-null cells, which is out the scope of this study on filling null cells.

A *filling* I' of I is also an instance of $\mathcal{R}$ such that:
(1) Existing non-null cells do not change, i.e., $t'_i[A] = t_i[A]$ if $t_i[A] \neq_-$, where $t'_i[A]$ is the cell in I' corresponding to $t_i[A]$ in I.
(2) Satisfaction of DDs is still retained, having $I' \vDash \Sigma$.
(3) The filling should not be a random guess without any supporting neighbors. In other words, for each filled cell, $t'_i[A] \neq t_i[A]$, there should exist a $t_j \in I$ having $(t'_i, t_j) \asymp \phi[XA]$ for some $(X \to A, \phi[XA]) \in \Sigma$.

It is worth noting that filling a null cell on attribute A (by any value) will never introduce violations to any DD in the form of $(X \to B, \phi[XB]), A \neq B$. In data repairing [2], changing the value of $t[A], A \in X$, could make it possibly satisfy the condition on determinant attributes X of a dependency and thus introduce violations. This *never* occurs in data imputation, where filling a null cell could only make it no longer agree the restriction on X (recall that a null cell always agrees a distance restriction).

In other words, the imputation on attribute A could only introduce violations to DDs in the form of $(X \to A, \phi[XA])$. To ensure (2) the satisfaction of DDs in data imputation, it is sufficient to exam $I'_A \vDash \Sigma_A$ for each $A \in \mathcal{R}$, where $I_A \subseteq I$ is the set of tuples with null cells on attribute A, I'_A is the corresponding filling of I_A on attribute A, and $\Sigma_A \subseteq \Sigma$ denotes DDs in the form of $(X \to A, \phi[XA])$.

Since the imputation of an attribute A will not introduce violations in other attributes B, we can fix the attributes *one-by-one*. That is, for simplicity, we can focus on the imputation over one attribute A at a time: finding a filling I'_A for I_A such that $I'_A \vDash \Sigma_A$.

Example 2 (Example 1 continued)**.** Consider again DD_1 : $(\mathsf{Name}, \mathsf{Street} \to \mathsf{Address}, \langle[0, 1], [0, 9], [0, 2]\rangle)$ in Example 1. The relation I in Table 1 satisfies this DD_1, since for any two tuples, e.g., t_4 and t_5, having similar Name (distance equal to 0 in the range of $[0, 1]$) and similar Street (distance equal to 2 within $[0, 9]$), it always has $(t_4, t_5) \asymp \phi[\mathsf{Address}]$, i.e., Address distance of t_4 and t_5 (equal to 2) is in $[0, 2]$.

During data imputation, tuples t_1 and t_5, sharing similar Name and Street values, i.e., compatible w.r.t. the differential function $\phi[\mathsf{Name}, \mathsf{Street}]$ specified in DD_1, are identified as value similarity neighbors. As illustrated in I'_A in Table 3, a possible filling for the missing $t_1[\mathsf{Address}]$ in Table 1 is thus $t_1[\mathsf{Address}]$=#402 Jordan Rd, which is suggested by the Address value of its similarity neighbor t_5. □

2.2 Problem Statement

As discussed in the introduction, referring to the difficulty in filling a cell (owing to data sparsity and variety), we target on the fillings that can fill more null cells.

Given I_A with Σ_A, let $\Delta(I'_A, I_A) = \{t'_i[A] \mid t'_i[A] \neq_-, t_i[A] =_-, t_i \in I_A\}$ be the difference on cells between I_A and its filling I'_A w.r.t. Σ_A. We call the total number of cells filled in I'_A for I_A, $|\Delta(I'_A, I_A)|$, the *filling gain*.

Definition 1. *A filling I'_A is* maximal *if there does not exist any other filling I''_A of I_A, such that $\Delta(I'_A, I_A) \subset \Delta(I''_A, I_A)$.*

A result of '$_-$' in a maximal fix I'_A denotes that the cell cannot be filled with the values from $\mathsf{dom}_I(A)$. The reason is, as illustrated in the following example, filling candidates in different cells may conflict with each other.

Definition 2. *A filling I'_A is* maximum *if there does not exist any other filling I''_A of I_A, such that $|\Delta(I'_A, I_A)| < |\Delta(I''_A, I_A)|$.*

Table 3: Example of maximal and maximum fillings

I'_A	...	Address
t_1	...	#402 Jordan Rd
t_2	...	No402 Jordan Rd
t_3	...	–

I''_A	...	Address
t_1	...	#402 Jordan Rd
t_2	...	#402 Jordan Rd
t_3	...	#531 Jordan Rd

The set of maximum fillings is a subset of maximal fillings, according to the definitions.

Definition 3. *A filling I'_A is* full *if every null cell on attribute A in I_A is filled, i.e., $t'_i[A] \neq_-, \forall t'_i \in I'_A$.*

Example 3 (Example 1 continued)**.** Consider the imputation on attribute Address, with $I_A = \{t_1, t_2, t_3\}$. Another DD_2, in $\Sigma_A = \{\text{DD}_1, \text{DD}_2\}$, states that two Address values a_1 and a_2 in the same Street should be similar (with at most 3 different digits, having $d_{\text{Address}}(a_1, a_2) \leq 3$), such as #402 Jordan Rd and #531 Jordan Rd in tuples t_5 and t_6 in Table 1.

$$\text{DD}_2 : (\text{Street} \rightarrow \text{Address}, \langle [0,0], [0,3] \rangle)$$

Such partial similarities between tuples t_5 and t_6 on a subset of Street and Address attributes are not considered by the existing tuple-similarity methods [19, 20], given the distinct Name values (not denoting the same people). These enriched similarity relationships may increase the chance of missing data being filled, and preventing potential irrational filling.

To illustrate example maximal/maximum fillings, consider possible filling candidates $t_2[\text{Address}] = \text{No402 Jordan Rd}$ and $t_3[\text{Address}] = \#531$ Jordan Rd, w.r.t. DD_1, having distance 5 not in the range $[0, 3]$ specified by DD_2. As shown in I'_A in Table 3, due to the choice of $t_2[\text{Address}]$ =No402 Jordan Rd, $t_3[\text{Address}]$ cannot be filled by any value w.r.t. $\{\text{DD}_1, \text{DD}_2\}$. That is, I'_A is already a maximal filling. Instead, I''_A in Table 3 is a full filling (also maximum and maximal) that fills all the null cells. □

Since the filling candidates suggested by different similarity neighbors may conflict w.r.t. the constraints, obtaining the maximum filling is non-trivial.

Problem 1. *The* full filling *problem is to determine whether there exists a full filling I'_A of I_A w.r.t. Σ_A.*

Problem 2. *The* maximum filling *problem is to find a maximum filling I'_A of I_A w.r.t. Σ_A.*

The existence of a full filling can be determined by finding the maximum filling and checking whether all the null cells are filled. Unfortunately, both problems are hard.

Theorem 1. *The full filling problem is* NP*-complete.*

Proof sketch. The proof is provided by constructing a reduction from the 3-CNF-SAT problem, which is one of Karp's 21 NP-complete problems [8]. In particular, we can show that a 3-CNF has a satisfying assignment if and only if the corresponding relation instance has a full filling. □

Referring to the hardness of determining full filling, it is not surprising to conclude the hardness of maximum filling. Indeed, if the maximum filling can be found (efficiently), it already determines whether a full filling exists or not.

Theorem 2. *The maximum filling problem is* NP*-hard.*

(See proofs of all theorems in technique report [1].)

3. COMPUTING EXACT SOLUTIONS

In this section, we present an approach for computing the maximum filling. Major steps include identifying local restrictions for specific tuple pairs, capturing filling candidates from neighbors, and transforming the problem to an integer linear programming (ILP) problem. (See Example 4 for instances of all these steps.)

Local Neighbor Restrictions

Two tuples t_1, t_2 in I are said *in neighborhood* if there exists at least one $(X \rightarrow A, \phi[XA]) \in \Sigma_A$ such that $(t_1, t_2) \asymp \phi[X]$.

Given multiple DDs in Σ_A, we may have specific distance restrictions for each individual tuple pair, according to the intersection inference rule for DDs implication [14].

Definition 4. *For any tuples t_1, t_2 in neighborhood, the* local neighbor restriction *$\phi[A](t_1, t_2)$ is an intersection ($\wedge$) of differential functions (distance ranges) of* DDs

$$\phi[A](t_1, t_2) = \bigwedge_{(X \rightarrow A, \phi_i[XA]) \in \Sigma_A, (t_1,t_2) \asymp \phi_i[X]} \phi_i[A].$$

Filling Candidates

For each tuple $t_i \in I_A$ with null cell on attribute A, we find all the tuples $t_c \in I$ that are in neighborhood with t_i w.r.t. Σ_A, i.e., neighbors. Each neighbor t_c with non-null $t_c[A]$ (from $I \setminus I_A$) suggests a set of candidates for filling the null cell $t_i[A]$, such that $(t'_i[A], t_c[A]) \asymp \phi[A](t_i, t_c)$.

We denote $\mathsf{can}(t_i[A])$, or simply $\mathsf{can}(t_i)$, the set of filling candidates for the null cell $t_i[A]$,

$$\mathsf{can}(t_i[A]) = \bigwedge_{t_c \in I \setminus I_A} \{a \in \mathsf{dom}_I(A) \mid (a, t_c[A]) \asymp \phi[A](t_i, t_c)\},$$

having $\mathsf{can}(t_i) \subseteq \mathsf{dom}_I(A)$. That is, $\mathsf{can}(t_i[A])$ is a set of candidates that are compatible with all the neighbors of t_i. As mentioned, a tuple without any neighbor is not able to be filled rationally and can be directly ignored.

Maximum Filling

Thus far, we have generated a set of filling candidates for each null cell, based on its non-null neighbors. The candidates can be represented as or-set relations [10], e.g., as shown in Table 4. As mentioned, not all the combinations of candidates are valid/compatible, owing to local neighbor restrictions between tuples in I_A inside.

Consider $I_A \subseteq I$ of m tuples. Let $c_i = |\mathsf{can}(t_i)|$. The maximum filling problem can be written as integer linear programming (ILP)

$$\max \sum_{i=1}^{m} \sum_{j=1}^{c_i} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j=1}^{c_i} x_{ij} \leq 1, \qquad 1 \leq i \leq m \tag{2}$$

$$x_{ij} \mathrm{w}_{il} \mathrm{v}_{jk} + x_{lk} \mathrm{w}_{il} \mathrm{v}_{jk} \leq 1, \quad 1 \leq i \leq m, 1 \leq j \leq c_i, \; 1 \leq k \leq c_l, 1 \leq l \leq m \tag{3}$$

$$x_{ij} \in \{0, 1\} \tag{4}$$

where $x_{ij} = 1$ if the j-th candidate $a_j \in \mathsf{can}(t_i)$ is selected to fill the null cell $t_i[A]$, otherwise 0. w_{il} and v_{jk} are constants such that $\mathrm{w}_{il} = 1$ if tuples $t_i, t_l \in I_A$ are in neighborhood, otherwise 0; and $\mathrm{v}_{jk} = 0$ if $(a_j, a_k) \asymp \phi[A](t_i, t_l), a_j \in \mathsf{can}(t_i), a_k \in \mathsf{can}(t_l)$, otherwise 1.

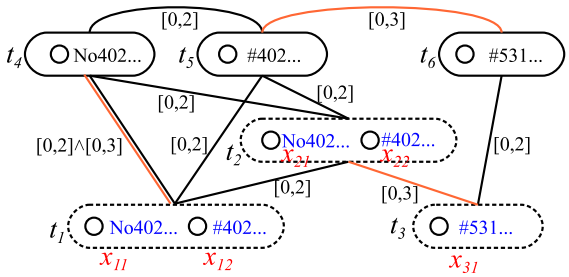

Figure 2: Filling candidates on attribute Address

Table 4: Example of filling candidates

$\mathsf{can}(I_A)$	...	**Address**
t_1	...	{#402 Jordan Rd, No402 Jordan Rd}
t_2	...	{#402 Jordan Rd, No402 Jordan Rd}
t_3	...	{#531 Jordan Rd}

Referring to formula (2), $\sum_{j=1}^{c_i} x_{ij} \leq 1$, there is at most one candidate say $a_j \in \mathsf{can}(t_i)$ that can be selected for filling a null cell $t_i[A]$, i.e., $x_{ij} = 1$. The second constraint formula (3) specifies that, for two tuples (t_i, t_l) in neighborhood ($\mathrm{w}_{il} = 1$), their corresponding selected assignments, say $a_j \in \mathsf{can}(t_i)$ and $a_k \in \mathsf{can}(t_l)$ having $x_{ij} = x_{lk} = 1$, must agree on $(a_j, a_k) \asymp \phi[A](t_i, t_l)$, or equivalently $\mathrm{v}_{jk} = 0$, in order to meet the requirement of $x_{ij}\mathrm{w}_{il}\mathrm{v}_{jk} + x_{lk}\mathrm{w}_{il}\mathrm{v}_{jk} \leq 1$.

Example 4 (Example 3 continued)**.** Figure 2 illustrates the neighborhoods of tuples in Table 1. Each ellipse denotes a tuple. The edge between two tuples, e.g., t_1 and t_2, denotes their neighborhood, i.e., $(t_1, t_2) \asymp \phi[X]$ of DD_1 (in black).

Different tuple pairs may have distinct local distance restrictions, e.g., (t_2, t_3) requires distance within $[0, 3]$ according to DD_2 (in orange). For (t_1, t_4), multiple restrictions are declared by DD_1 and DD_2. It requires distances not only in the range of $[0, 3]$ but also $[0, 2]$, whose intersection is $[0, 2]$.

The neighbors from $I \setminus I_A$ with non-null cells on A (solid line ellipses) suggest a set of all possible candidates (circles) for each null cell (dotted line ellipse), as indicated in Table 4. Note that not all the combinations of candidates are valid fillings, owing to the distinct distance restrictions between tuples in I_A. For instance, No402 Jordan Rd for t_2 and #531 Jordan Rd for t_3, with distance 5 not in the range of $[0, 3]$, are not compatible and cannot make up a filling.

By transforming the problem to integer linear programming, each candidate is associated with a variable x_{ij}. There is only one pair of candidates No402... and #531... in violation to the local restriction $[0, 3]$ between t_2 and t_3. We put $x_{21} + x_{31} \leq 1$. A solution of ILP can be $x_{12} = x_{22} = x_{31} = 1, x_{11} = x_{21} = 0$. It leads to a maximum filling $t_1[\mathsf{Address}] = t_2[\mathsf{Address}] =$#402... and $t_3[\mathsf{Address}] =$ #531..., which is exactly the full filling I''_A in Table 3. □

4. APPROXIMATION METHOD

Recognizing the hardness of imputation with DDs, we focus on approximation approaches below. Unfortunately, approximation of the imputation problem is also hard.

Theorem 3. *The maximum filling problem is Max-$\mathcal{SNP}$-hard. That is, there exists an $\varepsilon > 0$ such that $(1-\varepsilon)$-approximation of the maximum filling is* NP*-hard.*

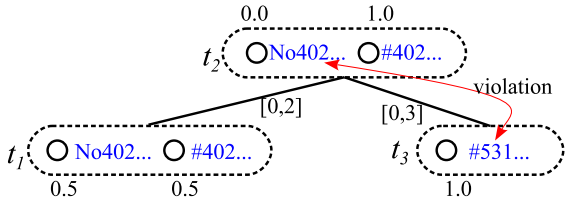

Figure 3: Filling by linear programming

Referring to the hardness of approximation, we study heuristics for constructing feasible solutions that might achieve a high filling gain.

A natural intuition is to consider linear programming (LP) relaxation of ILP. That is, change the requirement of $x_{ij} \in \{0, 1\}$ in formula (4) to $0 \leq x_{ij} \leq 1$.

Since the outputs x_{ij} of LP are real numbers, we need to round the solution into integers in order to form fillings. Intuitively, we may greedily pick a x_{ij} with the maximum value to fill in each step, i.e., Line 3 in Algorithm 1.

We say an assignment $t'_i[A] = a_j$ is *compatible* with its neighbor assignment $t'_l[A]$ in I'_A, if $(t'_i[A] = a_j, t'_l[A]) \asymp \phi[A](t_i, t_l)$. When the assignment is *compatible with all the neighbors*, i.e., $(t'_i[A], t'_l[A]) \asymp \phi[A](t_i, t_l), \forall \phi[A](t_i, t_l) \in \Phi[A], t'_l \in I'_A$, we denote $(t'_i[A] = a_j, I'_A) \asymp \Phi[A]$. As shown in Lines 6-10 in Algorithm 1, a candidate a_j is assigned only if it is compatible with all the neighbors (Line 7).

Algorithm 1 ROUND$(I_A, \Phi[A])$

Input: I_A with candidate sets $\mathsf{can}(I_A)$ and local neighbor restrictions $\Phi[A]$
Output: A filling I'_A

```
1: let x be a solution of linear programming
2: I'_A := I_A
3: while max_{x_ij ∈ x} x_ij ≥ 0 and |Δ(I'_A, I_A)| < |I_A| do
4:     let x_ij be arg max_{x_ij ∈ x} x_ij
5:     set x_ij to negative
6:     t'_i[A] := a_j the j-th candidate in can(t_i)
7:     if (t'_i[A] = a_j, I'_A) ≍ Φ[A] then
8:         set x_il to negative for each a_l ∈ can(t_i)
9:     else
10:        t'_i[A] :=_
11: return I'_A
```

The correctness of the ROUND algorithm is obviously ensured that none of the remaining null cells can further be filled over the current assignments.

Proposition 4. ROUND *always returns a maximal filling.*

For each possible assignment $t'_i[A]$, we need to check whether it violates with its neighbor. Finding the max x_{ij} in Line 3 can be done in constant time by amortizing all the x_{ij} values over an integer domain of $[0, d]$, e.g., by $\lfloor d * x_{ij} \rfloor$, where d is a large integer. Considering all the candidates of each tuple, the complexity of ROUND algorithm is $O(m^2 c)$, where c is the maximum candidate size of a tuple in I_A.

Example 5 (Example 4 continued)**.** Let the numbers attached to each candidate in Figure 3 denote a LP solution of x_{ij} in Figure 2. According to $\sum_{j=1}^{c_i} x_{ij} \leq 1$, the summation of each tuple is no greater than 1, e.g., $0.5 + 0.5 \leq 1$ in t_1.

Moreover, as mentioned in Example 4, the violation between candidates No402... and #531... (denoted by red arrows in Figure 3) requires $x_{21} + x_{31} \leq 1$, i.e., $0.0 + 1.0 \leq 1$.

During rounding, we first select #402... for t_2 and #531... for t_3, whose x_{ij} values are the highest 1.0. For the next largest x_{ij}, suppose that #402... with $x_{12} = 0.5$ is selected for t_1. Since it is a valid assignment compatible with all neighbors, Line 8 in Algorithm 1 sets other candidates of t_1 to negative. Finally, all the x_{ij} in I_A have been considered, and the generated filling is exactly the full filling I'' in Table 3. □

Unfortunately, we do not get a theoretical performance guarantee for the ROUND approximation algorithm, although this simple heuristic, based on LP relaxation, performs well in practice (as shown in the experiments).

5. RANDOMIZED ALGORITHM AND DE-RANDOMIZATION

To ensure the quality in terms of filling gain, in this section, we devise algorithms that return maximal fillings, with certain approximation guarantees w.r.t. the optimal maximum filling.

5.1 Randomized Imputation

For randomly sampling fillings, each candidate is associated with a probability of being selected as a filling. The solution x_{ij} of LP indicates a heuristic of constructing a filling with larger gain. Intuitively, we may employ x_{ij} to interpret the probability of $a_j \in \mathsf{can}(t_i)$ being selected.

$$\Pr[t'_i[A] = a_j] = \frac{x_{ij} + \epsilon}{c_i\epsilon + 1 + \sum_{j=1}^{c_i} x_{ij}} \tag{5}$$

$$\Pr[t'_i[A] = \_] = \frac{1}{c_i\epsilon + 1 + \sum_{j=1}^{c_i} x_{ij}} \tag{6}$$

where $\epsilon \geq 0$ trades off the contribution of LP estimations. When $\epsilon = 0$, the probability is exactly proportional to x_{ij}. On the other hand, if ϵ takes an extremely large positive, it approximately denotes equal probability having $\Pr[t'_i[A] = a_j] \approx \frac{1}{c_i}$ for each $a_j \in \mathsf{can}(t_i)$ and $\Pr[t'_i[A] = \_] \approx \frac{1}{c_i\epsilon}$.

Proposition 5. *We have*

$$\Pr[t'_i[A] = \_] + \sum_{a_j \in \mathsf{can}(t_i)} \Pr[t'_i[A] = a_j] = 1.$$

The conclusion is straightforward according to the definitions in formulas (5) and (6).

Proposition 6. *There exists a randomized algorithm which finds a full filling to any fully-fillable instance in expected time $\mathcal{O}((1/p)^m)$, where p is the minimum probability of a candidate being selected.*

Algorithm 2 works in a pay-as-you-go way, i.e., repeats ℓ steps of random filling. The more repeats are conducted, the more likely it returns the maximum filling. Note that there is no need to consider assignments with violations to restrictions. They are discarded by the algorithm in Line 6.

The correctness of the RANDOM algorithm is obvious by showing that all the returned fillings are maximal. Most importantly, all the possible maximal/maximum fillings are able to be sampled by the algorithm.

Algorithm 2 RANDOM($I_A, \Phi[A]$)

Input: I_A with candidate sets $\mathsf{can}(I_A)$ and local neighbor restrictions $\Phi[A]$
Output: A filling I'_A
1: initialize probabilities
2: **repeat**
3: **for** each $t_i \in I_A$ **do**
4: randomly draw a value $a_j \in \mathsf{can}(t_i)$ with probability $\Pr[t'_i[A] = a_j]$ for $t'_i[A]$ or leave it as null with probability $\Pr[t'_i[A] = \_]$
5: **if** $t'_i[A] = a_j$ is in violation with any $t'_l[A]$ **then**
6: $t'_i[A] := \_$
7: **if** I'_A is not maximal **then**
8: fill null cell by any valid candidate (not in violation)
9: rank I'_A in the top-k list K
10: **until** ℓ times
11: **return** the top-1 filling in K with the highest filling gain

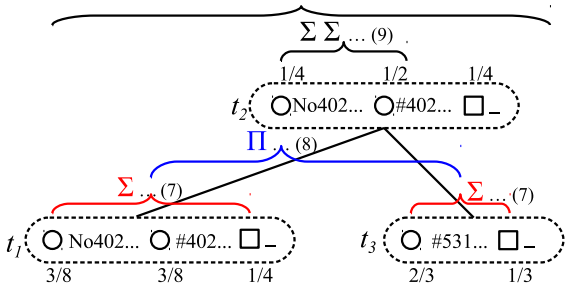

Figure 4: Random filling with probability

Proposition 7. *All the possible maximal/maximum fillings could be generated by the* RANDOM *algorithm.*

Example 6 (Example 5 continued)**.** Let the number of each candidate in Figure 4 denote its probability (e.g., computed from the LP solution in Figure 3 with $\epsilon = 1$). Value '_' denoted by square also has a probability of being selected.

RANDOM algorithm randomly draws a candidate with the given probability for each tuple, e.g., t_1[Address] =#402..., t_2[Address] =_ and t_3[Address] = #531.... Since the remaining null cell in t_2 can be further filled, the current filling is not maximal. According to Line 8 in Algorithm 2, the remaining null cell should be filled by any valid candidate, if exists, e.g., t_2[Address] =#402.... Finally, a maximal filling is constructed. □

Performance Analysis on Expected Filling Gain $\mathbf{E}[W]$

We study the performance guarantee of the RANDOM algorithm w.r.t. the maximum filling (optimum solution). Let random variable W denote the filling gain, i.e., $|\Delta(I'_A, I_A)|$, having $0 \leq W \leq |I_A|$. In the following, we analyze the bound of expected W, i.e., $\mathbf{E}[W]$, the number of null cells filled by random filling in expectation.

Consider an assignment $t'_i[A] = a_j, a_j \in \mathsf{can}(t_i)$, of any tuple $t_i \in I_A$.

For any tuple t_l in I_A in neighborhood with t_i, i.e., $t_l \in I_A, \phi[A](t_i, t_l) \in \Phi[A]$, the probability of $t'_i[A] = a_j$ compat-

ible with $t'_l[A]$ is

$$\Pr[(t'_i[A] = a_j, t'_l[A]) \asymp \phi[A](t_i, t_l)] \quad (7)$$
$$= \Pr[t'_l[A] = \_] + \sum_{a_k \in \mathsf{can}(t_l), (a_j, a_k) \asymp \phi[A](t_i,t_l)} \Pr[t'_l[A] = a_k].$$

Consequently, the probability of $t'_i[A] = a_j$ compatible with all the tuples t_l in neighborhood with t_i is

$$\Pr\left[(t'_i[A] = a_j, I'_A) \asymp \Phi[A]\right] \quad (8)$$
$$= \prod_{t_l \in I_A, \phi[A](t_i,t_l) \in \Phi[A]} \Pr[(t'_i[A] = a_j, t'_l[A]) \asymp \phi[A](t_i, t_l)].$$

Considering all the non-null assignments $a_j \in \mathsf{can}(t_i)$ of each $t_i \in I_A$, with probability $\Pr[t'_i[A] = a_j]$, we have

$$\mathbf{E}[W] = \sum_{i=1}^{m} \sum_{j=1}^{c_i} \Pr[t'_i[A] = a_j] \Pr\left[(t'_i[A] = a_j, I'_A) \asymp \Phi[A]\right]. \quad (9)$$

Example 7 (Example 6 continued)**.** We illustrate the computation of $\mathbf{E}[W]$ for the example in Figure 4. Consider the first candidate No402... (a_j in formula (7)) of t_2 (i.e., t_i) and its neighbor t_1 (i.e., t_l). The candidate is not in violation with any candidate of t_1. The probability of $t'_2[A] =$ No402... compatible with $t'_1[A]$ is

$$\Pr[(t'_2[A] = \mathsf{No402}\ldots, t'_1[A]) \asymp \phi[A](t_2, t_1)] = \frac{1}{4} + \frac{3}{8} + \frac{3}{8} = 1.$$

For the neighbor t_3, however, No402... is in violation with #531..., as discussed in Example 4, Figure 3. The probability of $t'_2[A] =$ No402... compatible with $t'_3[A]$ is

$$\Pr[(t'_2[A] = \mathsf{No402}\ldots, t'_3[A]) \asymp \phi[A](t_2, t_3)] = \frac{1}{3}.$$

By considering all the neighbors $\{t_1, t_3\}$ of t_2 in formula (8), we have the probability of $t'_2[A] =$ No402... compatible with all neighbors

$$\Pr\left[(t'_2[A] = \mathsf{No402}\ldots, I'_A) \asymp \Phi[A]\right] = 1 * \frac{1}{3}.$$

For each candidate of all tuples, we can compute such a probability, e.g., $\Pr\left[(t'_2[A] = \#402\ldots, I'_A) \asymp \Phi[A]\right] = 1$, $\Pr\left[(t'_3[A] = \#531\ldots, I'_A) \asymp \Phi[A]\right] = \frac{3}{4}$, etc.

Finally, by the weighted aggregation in formula (9), we have $\mathbf{E}[W] = \frac{3}{8} * 1 + \frac{3}{8} * 1 + \frac{1}{4} * \frac{1}{3} + \frac{1}{2} * 1 + \frac{2}{3} * \frac{3}{4} = \frac{11}{6}$. That is, a number of $\frac{11}{6}$ null cells are filled by random filling in expectation. □

Theorem 8. *The* RANDOM *algorithm returns a solution with the expected filling gain* $\mathbf{E}[W] \geq \left(\frac{1}{c\epsilon+2}\right)^{b+1} OPT$*, where* OPT *is the optimal filling gain of the maximum filling,* c *is the maximum candidate size of a tuple in* I_A*, and* b *is the maximum number of neighbors of a tuple in* I_A*.*

When taking $\epsilon = 0$, i.e., the probability is proportional to x_{ij}, we have $\mathbf{E}[W] \geq \left(\frac{1}{2}\right)^{b+1} OPT$.

5.2 Derandomization

Theorem 8 gives only the performance guarantee in expectation of the RANDOM algorithm. In the following, we advance the approach by giving a deterministic bound of approximation rather than expectation. The idea is to guide the filling via *conditional expectation* instead of assigning randomly in each step. (More specifically, each step chooses an assignment that maximizes the conditional expectation.)

5.2.1 Conditional Expectation

Consider the conditional expected filling gain, given a number of tuples $t_1, \ldots, t_{i-1}$ that have been filled,

$$\mathbf{E}[W \mid I_A^{i-1}] = \mathbf{E}[W \mid t'_1[A] = a_1, \ldots, t'_{i-1}[A] = a_{i-1}],$$

where the assignments of $t_1, \ldots, t_{i-1}$ have been determined, denoted as I_A^{i-1}. Intuitively, $\mathbf{E}[W \mid I_A^{i-1}]$ denotes the number of filled cells in $t_1, \ldots, t_{i-1}$ plus the expectation of cells that can be filled in the remaining $t_i, \ldots, t_m$. We have $\mathbf{E}[W \mid I_A^{\prime 0}] = \mathbf{E}[W]$ initially, and $\mathbf{E}[W \mid I_A^{\prime m}]$ is the exact gain of the filling $I_A^{\prime m}$.

Let $t'_i[A] = a_i, a_i \in \mathsf{can}(t_i) \cup \{\_\}$ be the next assignment. We study the computation of the conditional expectation $\mathbf{E}[W \mid I_A^{\prime i}]$. There is no need to compute it from scratch. Instead, it can be calculated incrementally from $\mathbf{E}[W \mid I_A^{\prime i-1}]$ by considering the following possible cases.

Case 1. If $t'_i[A] = a_i$ is in violation with any $t'_1[A] = a_1, \ldots, t'_{i-1}[A] = a_{i-1}$, no valid solution can be generated, i.e., $\mathbf{E}[W \mid I_A^{\prime i}] = 0$.

Case 2. For $t'_i[A] = a_i$ compatible with existing assignments, we further consider three sub-cases for updating the compatible probability in formula (8) of other tuples.

Case 2.1. For any t_l not in neighborhood with t_i, the compatible probability will not change, having

$$\Pr\left[(t'_l[A] = a_k, I'_A) \asymp \Phi[A] \mid I_A^{\prime i}\right]$$
$$= \Pr\left[(t'_l[A] = a_k, I'_A) \asymp \Phi[A] \mid I_A^{\prime i-1}\right].$$

Case 2.2. For t_l in neighborhood with t_i, we have

$$\Pr\left[(t'_l[A] = a_k, I'_A) \asymp \Phi[A] \mid I_A^{\prime i}\right] = 0,$$

if $(a_i, a_k) \not\asymp \phi[A](t_i, t_l)$, for any $a_k \in \mathsf{can}(t_l)$.

Case 2.3. For t_l in neighborhood with t_i, and $a_k \in \mathsf{can}(t_l)$ such that $(a_i, a_k) \asymp \phi[A](t_i, t_l)$, we have

$$\Pr\left[(t'_l[A] = a_k, I'_A) \asymp \Phi[A] \mid I_A^{\prime i}\right] \quad (10)$$
$$= \frac{\Pr\left[(t'_l[A] = a_k, I'_A) \asymp \Phi[A] \mid I_A^{\prime i-1}\right]}{\Pr\left[(t'_l[A] = a_k, t'_i[A]) \asymp \phi[A](t_l, t_i) \mid I_A^{\prime i-1}\right]}.$$

Case 3. If the next assignment is $t'_i[A] = \_$, for any t_l in neighborhood with t_i, it belongs to the above Case 2.3.

Finally, by the weighted summation in formula (9) of compatible probabilities updated in the aforesaid cases, $\mathbf{E}[W \mid I_A^{\prime i}]$ is computed.

Example 8 (Example 7 continued)**.** Suppose that $t'_3[A] =$ #531... is the first assignment in Figure 4. This assignment will not change the compatible probability of t_1 which is not in neighborhood with t_3, i.e., Case 2.

Since No402... is in violation with #531..., as discussed in Example 4 Figure 3, i.e., Case 2.2, we have

$$\Pr\left[(t'_2[A] = \mathsf{No402}\ldots, I'_A) \asymp \Phi[A] \mid t'_3[A] = \#531\ldots\right] = 0.$$

For $\Pr\left[(t'_2[A] = \#402\ldots, I'_A) \asymp \Phi[A]\right] = 1$, we update it by dividing $\Pr[(t'_2[A] = \#402\ldots, t'_3[A]) \asymp \phi[A](t_2, t_3)] = 1$ according to Case 2.3. It still has

$$\Pr\left[(t'_2[A] = \#402\ldots, I'_A) \asymp \Phi[A] \mid t'_3[A] = \#531\ldots\right] = 1.$$

Finally, since $t'_3[A]$ is not '_' and will contribute 1 in the conditional expectation, we have $\mathbf{E}[W \mid t'_3[A] = \#531\ldots] = \frac{3}{8} * 1 + \frac{3}{8} * 1 + \frac{1}{4} * 0 + \frac{1}{2} * 1 + 1 = \frac{9}{4}$. □

5.2.2 Filling Guided by Conditional Expectation

Algorithm 3 presents the filling guided by conditional expectation. Instead of random assignment, in each step, we choose an assignment that can maximize the expected filling gain of the remaining unassigned cells.

$$t'_i[A] = \underset{a_j \in \mathsf{can}(t_i) \cup \{\_\}}{\arg\max} \mathbf{E}[W \mid t'_1[A] = a_1, \ldots, t'_i[A] = a_j] \tag{11}$$

It is found by searching $E_{\max}$, the maximum $\mathbf{E}[W \mid I'^{i}_A]$, from Lines 4 to 10 in Algorithm 3. Procedure ConExp($\mathbf{E}[W \mid I'^{i-1}_A], t_i, a_i$) computes $\mathbf{E}[W \mid I'^{i}_A]$ from $\mathbf{E}[W \mid I'^{i-1}_A]$ by considering all the aforesaid Cases.

Algorithm 3 Derand($I_A, \Phi[A]$)

Input: I_A with candidate sets $\mathsf{can}(I_A)$ and local neighbor restrictions $\Phi[A]$
Output: A filling I'_A

```
1: initialize probabilities
2: for each t_i ∈ I_A do
3:    t'_i[A] :=_                                   {construct I'^i_A}
4:    E_max := ConExp(E[W | I'^{i-1}_A], t_i, _)
5:    for each a_j ∈ can(t_i) do
6:       E := ConExp(E[W | I'^{i-1}_A], t_i, a_j)
7:       if E > E_max then
8:          E_max := E
9:          t'_i[A] := a_j
10:   E[W | I'^i_A] := E_max
11: if I'_A is not maximal then
12:    fill null cell by any valid candidate (not in violation)
13: return I'_A
```

Procedure ConExp(E, t_i, a_i)

Input: E denotes $\mathbf{E}[W \mid I'^{i-1}_A]$ and assignment a_i of $t_i[A]$
Output: $\mathbf{E}[W \mid t'_1[A] = a_1, \ldots, t'_{i-1}[A] = a_{i-1}, t'_i[A] = a_i]$

```
1: E:=E-∑_j Pr[t'_i[A] = a_j] Pr[(t'_i[A] = a_j, I'_A) ≍ Φ[A] | I'^{i-1}_A]
2: if a_i ≠_ then
3:    E := E + 1
4: for each t_l in neighborhood with t_i, l = i+1, ..., m do
5:    for each a_k ∈ can(t_l) do
6:       E:=E - Pr[t'_l[A]=a_k] Pr[(t'_l[A]=a_k, I'_A) ≍ Φ[A] | I'^{i-1}_A]
7:       if (a_i, a_k) ≍ φ[A](t_i, t_l) then
8:          update Pr[(t'_l[A]=a_k, I'_A) ≍ Φ[A] | I'^i_A] by formula (10)
9:          E:=E + Pr[t'_l[A]=a_k] Pr[(t'_l[A]=a_k, I'_A) ≍ Φ[A] | I'^i_A]
10: return E
```

Example 9 (Example 8 continued)**.** Consider another assignment of t_3, i.e., $t'_3[A] =\_$. We compute its $\mathbf{E}[W \mid t'_3[A] =\_] = \frac{3}{8} * 1 + \frac{3}{8} * 1 + \frac{1}{4} + \frac{1}{2} * 1 + 0 = \frac{3}{2}$ in Lines 3-4 in Algorithm 3. As shown in Example 7, $\mathbf{E}[W \mid t'_3[A] = \#531\ldots] = \frac{9}{4}$ is larger. Therefore, $t'_3[A] = \#531\ldots$ is assigned.

Similarly, for the next tuple t_2, we compute

$$\mathbf{E}[W \mid t'_2[A] = \#402\ldots, t'_3[A] = \#531\ldots] = \frac{11}{4},$$
$$\mathbf{E}[W \mid t'_2[A] = \mathsf{No}402\ldots, t'_3[A] = \#531\ldots] = 0,$$
$$\mathbf{E}[W \mid t'_2[A] =\_, t'_3[A] = \#531\ldots] = \frac{7}{4}.$$

The first one is larger and selected.

Finally, it leads to $\mathbf{E}[W \mid t'_1[A] = \#402\ldots, t'_2[A] = \#402\ldots, t'_3[A] = \#531\ldots] = 3$, where all null cells are filled. □

5.2.3 Correctness and Performance Analysis

First, the output filling is always maximal, referring to Line 12. Moreover, we can show that the maximum conditional expectation (including the final $\mathbf{E}[W \mid I'^{m}_A]$) is non-decreasing, i.e., no less than the initial $\mathbf{E}[W \mid I'^{0}_A] = \mathbf{E}[W]$.

Lemma 9. *There always exists an assignment* $t'_i[A] = a_i, a_i \in \mathsf{can}(t_i) \cup \{\_\}$ *such that* $\mathbf{E}[W \mid I'^{i}_A] \geq \mathbf{E}[W \mid I'^{i-1}_A]$ *, i.e.,*

$$\mathbf{E}[W \mid t'_1[A] = a_1, \ldots, t'_{i-1}[A] = a_{i-1}, t'_i[A] = a_i]$$
$$\geq \mathbf{E}[W \mid t'_1[A] = a_1, \ldots, t'_{i-1}[A] = a_{i-1}]$$

Given $\mathbf{E}[W \mid I'^{0}_A] = \mathbf{E}[W]$, it is sufficient to conclude the bound of $\mathbf{E}[W \mid I'^{m}_A]$.

Theorem 10. *The* Derand *algorithm guarantees to output a solution with filling gain* $\geq \mathbf{E}[W]$.

Referring to Theorem 8, it is not surprising that the filling gain by derandomization is at least $\left(\frac{1}{c\epsilon+2}\right)^{b+1} OPT$. When $\epsilon = 0$, we have approximation bound $\left(\frac{1}{2}\right)^{b+1} OPT$.

As shown in Procedure ConExp, the conditional expectation can be computed by considering all the neighbors of t_i and their corresponding candidates with complexity $O(cb)$. Considering all the m tuples in I_A and their candidates, the complexity of Algorithm 3 is $O(mc^2b)$.

6. EXPERIMENTS

This section reports the experimental evaluation on both effectiveness and efficiency of the proposed approaches. All programs are implemented in Java and the experiments were performed on a PC with 3.4 GHz CPU and 16 GB RAM.

6.1 Experimental Settings

We employ several real and synthetic datasets, including the Restaurant[6] dataset with name, address, type and city information of 864 restaurants, and a synthetic dataset generated by the UIS database generator[6].

Following the same line of evaluating data repairing techniques by artificially injecting errors [2], we randomly remove values as missing data. Imputation approaches are then applied to fill/recover these removed values.

Let truth be the set of removed cell values and found be the set of filling results returned by imputation algorithms (not including the null cells that are failed to fill). The recall accuracy is given by $recall = \frac{|\mathsf{truth} \cap \mathsf{found}|}{|\mathsf{truth}|}$, i.e., the proportion of null cells that are accurately filled/recovered; the precision accuracy is $precision = \frac{|\mathsf{truth} \cap \mathsf{found}|}{|\mathsf{found}|}$, denoting the proportion of filled cells (non-null) that are correct; and $f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$ represents an overall accuracy [16]. In particular, while the null cells that are failed to fill (leave '_' unchanged) count negatively toward the recall accuracy, these null cells are not necessary to be considered in the precision accuracy.

Enlightened by discovering FDs from the complete part of incomplete data [18], DDs used in the experiments are also obtained by applying the discovery techniques [14, 15] (with manual verification). Specifically, to avoid "*finding accidental rules which may not always be dependable nor correct*", the discovery approaches [14, 15] use support and confidence guarantees. To avoid accidental undependable

[6] http://www.cs.utexas.edu/users/ml/riddle/data.html

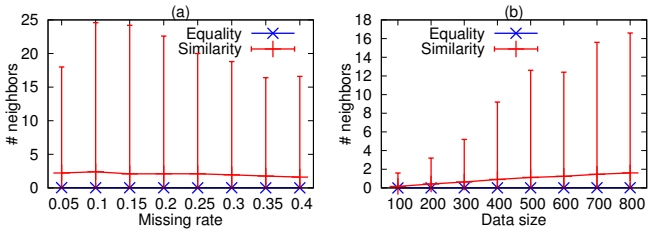

Figure 5: Number of neighbors (Restaurant)

rules, the support measures the number of tuple pairs that can be covered by the distance thresholds on determinant attributes X of rules. To ensure correctness, the confidence evaluates the proportion of the aforesaid covered tuple pairs that are also covered by the distance threshold on dependent attribute A of the rule. The discovery approaches [14, 15] return high confidence (≥ 0.95 in our experiments) rules, ranked by their supports in decreasing order. (Rules with redundant semantics are excluded from the results, see [14, 15] for more details.) We manually exam the returned rules by discarding those low support, ranked far behind (unreliable) rules, which may overfit the data. Owing to the limited space, we present the full list of DDs obtained and used in the experiments, in the long version technique report [1].

To vary the distance in rules, e.g., by tightening towards 0, it turns similarity rules to equality rules. The precision of imputation may slightly increases, as the equality-based Certain illustrated in Figure 6(b), while the corresponding recall drops dramatically in Figure 6(a). The overall f-measure is significantly worse. On the other hand, if we relax the distance which makes almost all the data pass the examination of a rule, it becomes useless. Such variations of rule distance have been reported in the preliminary study [15].

6.2 Comparison with Existing Techniques

This experiment compares our proposed method to Certain fixes with editing rules [7] on equality neighbors, ERACER [12] based on statistics over equal values, MIBOS [19] with tuple similarity defined on value equality, and CMI [20] considering imputation between most similar tuples.

First of all, Figure 5 observes the number of equality and similarity neighbors of tuples (with null cells), w.r.t. editing rules and similarity rules, respectively. Three points from higher to lower of each bar denote the maximum, average, minimum numbers of neighbors. The missing rate in sub-figure (a), e.g., 0.4, means that 40% of the tuples have null cells, while the data size in sub-figure (b) denotes the number of all tuples in a test. As illustrated, the (average) number of similarity neighbors is greater than that of equality neighbors. It verifies our claim (in Figure 1) that similarity rules can identify more neighbors than equality-based ones.

Figure 6 illustrates that a significantly higher f-measure of imputation is achieved by similarity neighbors (using our proposed Derand approach, see detailed results below). Possibly, the precision of our Derand could be a bit lower than that of equality-based Certain [7] in some tests, e.g., with missing rate 0.05 in Figure 6(b), which however is not stably observed in all the tests. This is largely because the found set of filled values by Certain is extremely small. Indeed, in most tests in Figure 6(b)(e), the precision of Derand is comparable to (higher than) that of the equality-based methods.

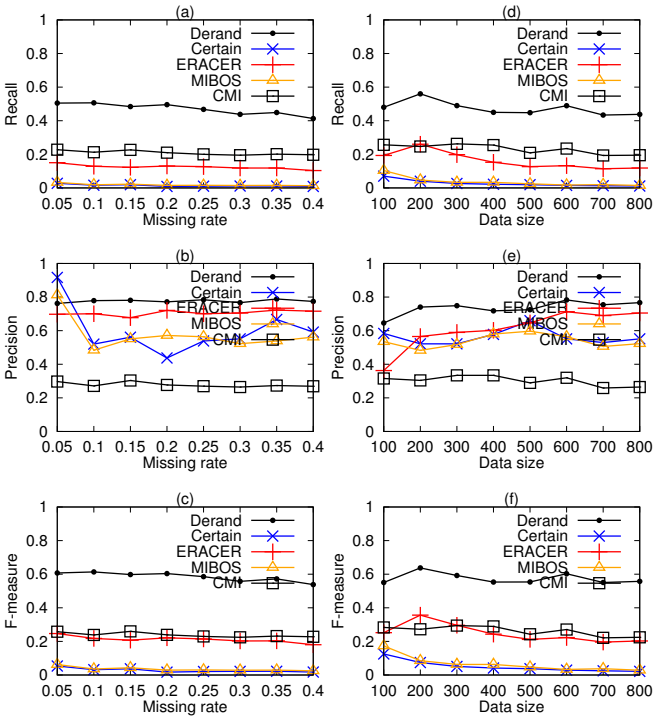

Figure 6: Imputation accuracy (Restaurant)

By filling more null cells, Derand shows a clearly higher recall in Figure 6(a)(d), and higher overall f-measure accuracy.

For MIBOS [19], whose tuple similarity is still defined on value equality, it shows very similar results to the equality neighbor-based Certain [7] with editing rules. In particular, as shown in Figure 6, given the very limited number of filled missing values (extremely low recall), the corresponding precision observations are not stable. Nevertheless, our DDs-based Derand shows comparable precision to the equality-based methods, but significantly higher recall owing to the successful identification of value similarity neighbors.

The CMI [20] method considers similarities over all the attributes, and thus has more opportunities in filling missing values with higher recall in Figure 6(a). However, strictly considering similarities over all the attributes, including those irrelevant ones, limits the power of finding similarity neighbors by CMI. Instead, by explicitly using only the (subset of) attributes specified in DDs, our Derand can identify and utilize more (partial) similarity neighborhoods between tuples that do not belong to the same cluster. Thereby, as shown in Figure 6(a), the recall of Derand is even higher.

Figure 7 reports the results on various duplication rates. The Restaurant dataset originally contains at most two tuples (versions) for denoting one entity. To introduce higher duplication rate, we enrich more tuples for an entity by randomly choosing two alternative values (versions) on each attribute. A duplication rate, e.g., 6, denotes that there are six tuples (versions generated) for denoting an entity, while the minimum duplication rate 1 denotes no duplicates. For duplication rate 1, it is not surprising that the CMI method [20] can hardly perform, since no duplicates exist in such a scenario. Remarkably, our maximum filling-based

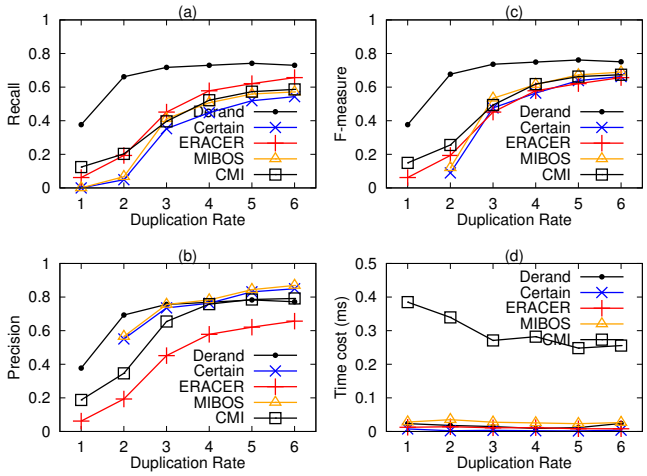

Figure 7: Varying duplication rates (Restaurant)

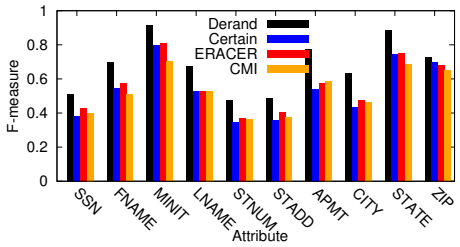

Figure 8: Comparison on various attributes (UIS)

Derand still shows a very significant improvement in imputation accuracy. Moreover, almost all the approaches benefit from higher duplication rates. Our proposed Derand already achieves considerably high accuracy when duplication rate is small, such as 2. The results verify again that our similarity rule-based method can successfully utilize the similarity relationships between tuples, even they are not duplicates (denoting the same entity). For the efficiency under various duplication rates (with the same data size), as shown in Figure 7(d), most methods are not affected clearly by duplication rates, except CMI. The reason is that the K-means clustering algorithm in CMI converges more quickly under a higher duplication rate.

Figure 8 presents the imputing results on specific attributes. As illustrated, our proposed Derand with similarity rules shows a clear improvement of f-measure on all the attributes, from 0.4-0.8 (by equality neighbors) to 0.5-0.9 (with similarity neighbors). MIBOS [19] fails to fill any missing value in all the tests and thus does not appear in Figure 8.

6.3 Application in Record Matching

To further validate the effectiveness of applying imputation in real applications, we consider the accuracy of record matching application [5]. An existing rule-based method [6] is directly implemented and performed over the Missing data without imputation, the filled data by Certain [7], ERACER [12], MIBOS [19], CMI [20], and our proposed Derand.

As shown in Figure 9, the record matching accuracy is generally related to the filling accuracy in Figures 6 and 8. With imputation, some missing data could be filled and the

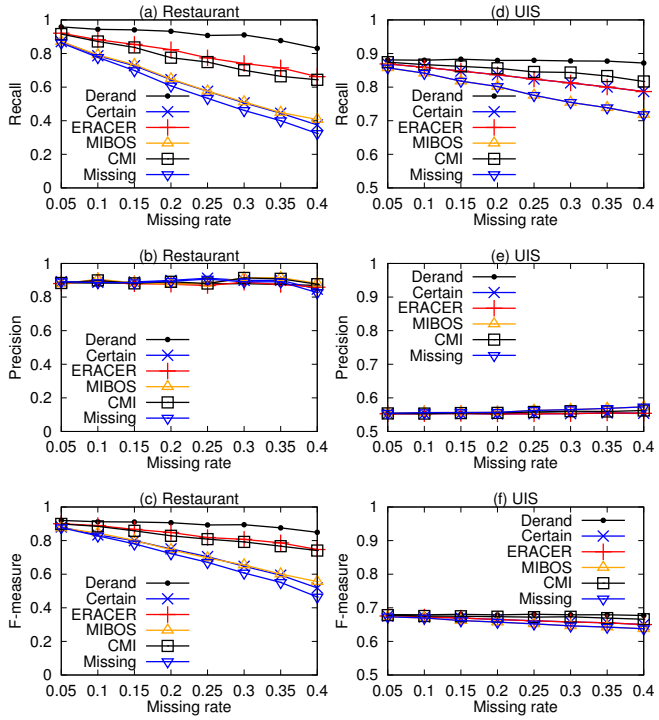

Figure 9: Application in record matching

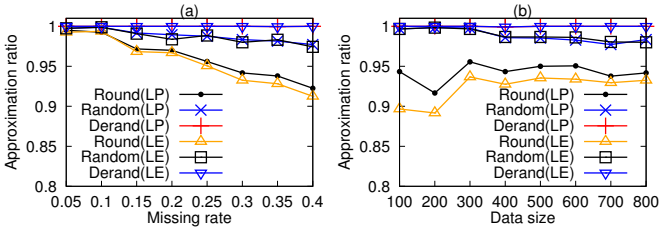

Figure 12: Approximation performance (Restaurant)

matching accuracy is improved compared to Missing without imputation. Our Derand with high imputation recall also leads to significantly higher recall accuracy of record matching application. Meanwhile, the record matching precision by applying Derand is comparable to that of Certain, since the imputation of Derand is as accurate as Certain.

6.4 Performance of Proposed Methods

In general, the imputation accuracy could be improved by given more reasonable rules. However, redundant semantics may exist among rules, which will not further improve the chance or accuracy of imputation. As shown in Figure 10, by given more than 4 DDs, the imputation accuracy cannot be further improved, which verifies the aforesaid justification. In practice, we can apply the existing techniques [14, 15] for reasoning about rules to eliminate redundant semantics, which is out the scope of this study.

The second experiment evaluates the performance of different techniques in our proposal. Figure 11 provides an overview of all the proposed approaches. Integer linear programming (ILP) can be implemented by branch and bound, and linear programming (LP) employs simplex. When com-

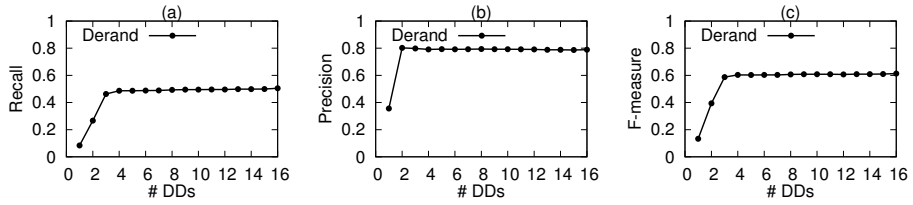

Figure 10: Varying the number of used DDs (Restaurant)

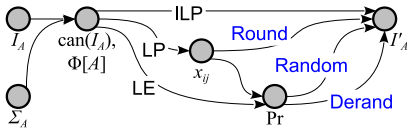

Figure 11: Overview of approaches

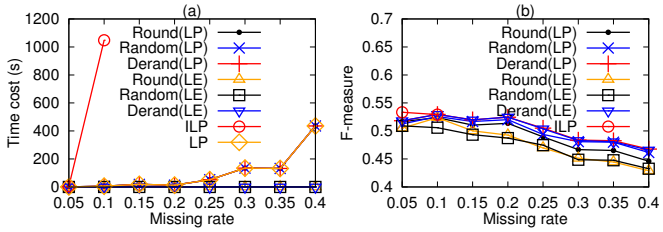

Figure 13: Varying missing rates (Restaurant)

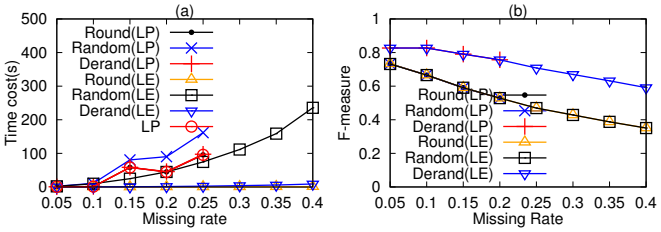

Figure 15: Varying missing rates (UIS)

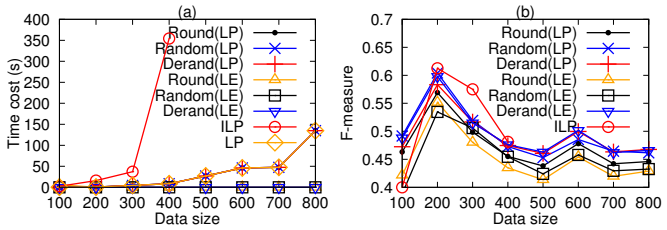

Figure 14: Scalability (Restaurant)

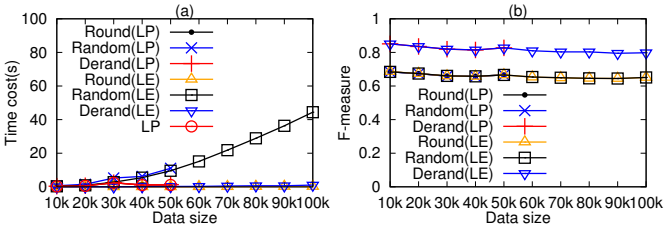

Figure 16: Scalability (UIS)

puting Pr by formula (5), as introduced at the beginning of Section 5.1, for a large ϵ, each $a_j \in \mathsf{can}(t_i)$ has a similar probability $\frac{1}{c_i}$ of being selected. We consider particularly this large ϵ case (namely LE), as its Pr can be directly computed without calling the costly LP solver.

Figure 12 observes (in Restaurant) the approximation ratio by comparing the filling gain of approximate fillings to the maximum filling (OPT). Derand with either LP or LE can achieve filling gains almost the same as the maximum filling. As mentioned, we do not obtain any performance guarantee for the Round approach, whose approximation ratio is low in the test. If the simple LE is used instead of LP, the approximation ratio of Round(LE) is even worse.

Next, we study the time and accuracy performance of imputation in Figures 13 and 14 with various missing rates and data sizes. Referring to the hardness of the filling problem, it is not surprising that the exact approach ILP takes extremely high costs. For approximation, Derand with both LP and LE can achieve the best accuracy in most tests. With the increase of missing rates, the time cost of LP-encapsulated approaches increases heavily in Figure 13(a). The rationale is that the LP step takes most of the time cost in the approximate filling approach, e.g., Round(LP). When the simple LE is applied instead of LP, the time cost is significantly lower. Meanwhile, with the increase of missing rate, it is not surprising that the accuracy drops. Owing to the distinct relationships in tuple pairs, the accuracy performance may be different in various sets of tuples, as show in Figure 14(b). Nevertheless, similar results of approaches are observed and the accuracy is higher than 0.4 in most tests. Figure 14(a) shows that the time cost increases as the increase of data sizes.

7. RELATED WORK

The idea of imputing missing values based on other available information is widely adopted in solving data imputation problems, and has been successfully applied in important areas such as analysis of variance of planned experiments, survey sampling, multivariate analysis, and so on [11]. Most preliminary studies dedicated to data imputation, e.g., tuple similarity-based [19] or clustering-based [20], also utilize the existing non-null values (assumed to be correct without modification) for imputation. Following the same line of imputing missing data under the constraints of certain rules, a.k.a. rule-based imputation [7], in this study, we also focus on filling the missing values on the A (dependent attribute) according to the X (determinant attributes) values. Detecting and modifying the existing incorrect non-null values, which is known as the data repairing problem, is out the scope of this study.

It is worth noting that any attribute could be the dependent attribute A, i.e., the imputation can be applied to any attribute in a relation. For example, Figure 8 shows the experimental results of imputation on all the 10 attributes in

the UIS dataset. Therefore, rather than imputation only on one attribute, we can practically fill the missing values over all the attributes.

The studied rule-based imputation of missing data is different from the constraint-based data repairing for eliminating violations [2, 9, 4] mainly in two aspects: (1) Data repairing often finds a repair that is closest to the initial values, known as the minimum change principle. For data imputation, however, there are no initial values (they are missing) to compare with, and thus the imputation relies only on the neighbors. (This is also the reason why the filling should not be a random guess of a value from the domain without any supporting neighbors, as indicated in Section 2.1.) In this sense, it is more urgent to study the extensive similarity neighbors for imputing missing data. (2) Repairing a value in an attribute may introduce violations to others. For data imputation, luckily, filling a null cell in an attribute will never introduce violations to other attributes, as analyzed at the end of Section 2.1. This promising property enables imputing attributes one-by-one.

Nevertheless, to practically address the potential incorrect values (in some antecedent attributes X) in the existing non-null data, a natural idea is to incorporate the data repairing and imputation techniques. We believe that after correcting the existing non-null values more accurately by the effective repairing techniques such as [3, 4, 17], the missing data imputation certainly benefits (not only our proposal but also all the other imputation methods). Indeed, after filling some null values, new inconsistencies may be uncovered for further repairing. In this sense, data imputation and repairing techniques are complementary. We may gradually improve the quality of data by iteratively applying the imputation and repairing approaches in turn. As an interesting future study, it is promising to simultaneously perform data repairing and imputation.

8. CONCLUSIONS

Imputing missing values of a tuple replies on others (neighbors) sharing the same information. Such filling could barely be done in the presence of data variety, owing to the very limited number of equality neighbors. It is essential to explore the extensive similarity neighbors, in order to fill *more* missing values (maximizing the data imputation). In this paper, we propose exact and approximate approaches for computing the maximum/maximal fillings. In particular, efficient approximation algorithms are devised with certain performance guarantees.

To support and utilize the DDs that might not be exactly correct, i.e., approximate DDs, a natural extension is to further rank the filling results according to the accuracy (confidence) of DDs rules that the filling violates. Intuitively, a filling violates those high confidence DDs should probably be ranked lower. We believe that there is a room to further improve the filling performance by such imperfect DDs rules. Since it leads to a new research problem, more comprehensive studies are expected in the future.

Acknowledgement

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; China NSFC under Grants 61325008, 61202008, 61328202 and 61370055; National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200; Hong Kong SRFDP&RGC ERG Joint Research Scheme M-HKUST602/12; Microsoft Research Asia Gift Grant and Google Faculty Award 2013.

9. REFERENCES

[1] Full version. *http://ise.thss.tsinghua.edu.cn/sx-song/doc/incomplete.pdf*.

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

[3] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.

[4] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[6] W. Fan, J. Li, X. Jia, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2009.

[7] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[8] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[9] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[10] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In *PODS*, pages 37–48, 1993.

[11] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[12] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD Conference*, pages 75–86, 2010.

[13] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[14] S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16, 2011.

[15] S. Song, L. Chen, and H. Cheng. Parameter-free determination of distance thresholds for metric distance constraints. In *ICDE*, pages 846–857, 2012.

[16] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[17] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD Conference*, pages 457–468, 2014.

[18] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati. Query processing over incomplete autonomous databases. In *VLDB*, pages 651–662, 2007.

[19] S. Wu, X. Feng, Y. Han, and Q. Wang. Missing categorical data imputation approach based on similarity. In *SMC*, pages 2827–2832, 2012.

[20] S. Zhang, J. Zhang, X. Zhu, Y. Qin, and C. Zhang. Missing value imputation based on data clustering. *Trans. on Computational Science*, 1:128–138, 2008.