Enriching Data Imputation with Extensive Similarity Neighbors

Written by Shaoxu S., etc

What is Data Imputation?

- ***Data imputation*** is to fill the missing data with the values of its neighbors who share the same information.
- For example, in Table 1, the missing value "Address" of tuple t_1 can be filled by the the corresponding value of t_4 (t_4['Address']="No402 Joardan Rd").
- Such neighbors could either be identified certainly by rule-based and statistical-based methods.

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

Rule-based Method

An editing rule declared upon the functional dependency semantics, *(Name, Street −> 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].

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.

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

Statistical-based Method

The statistical-based method considers the statistical distribution of values, e.g., between Name and Address in the relational dependency network

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

For instance, t_4 , t_5 are identified statistically as the neighbors of t_2 referring to their equal Name, so t_2[Name] can be filled either by “No402 Jordan Rd” or “#402 Jordan Rd”.

The Problem and a Possible Solution

- **Problem**: *The major problem in such data imputation is the sparsity of data that many reasonable value combinations are not observed.*

- **Solution**: *In this study, we identify a more extensive class of similarity neighbors, with value similarity relationships identified by similarity rules (differential dependencies or DDs).*

Differential Dependencies (DDs)

Consider a similarity rule in the form of differential dependencies (dd)

dd_1 :(Name, Street —> Address, {[0,1],[0,9],[0,2]})

It states that if two tuples have similar Name (i.e., having Name value distance 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]).

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

For example, tuples t_1 and t_4, sharing similar Name (distance=1) and Street values (distance=0) 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.

Features of DDs

1) DDs can address the value similarity with small value variations, which cannot be handled by value equality-based tuple-similarity (values of attributes should be 100% same).

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, dd_2 : (Street —>Address, {[0, 0], [0, 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, given the distinct Name values).

3) The similarity rule-based approach could utilize more similarity relationships between tuples.

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

Object and Challenges

- **Object**: By using DDs, we find and utilize more similarity neighbors to impute the missing values of tuples' attributes as much as possible.

- **Challenges**:
  - 1) Different from the certain fixes by equality neighbors with editing rules, 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.

Contributions

- *1) We analyze the hardness of the similarity rule based imputation problem.*

- *2) We present an approach for finding exact solutions.*

- *3) We study the hardness of approximation and propose a heuristic for imputing.*

- *4) We devise a randomized algorithm for returning maximal fillings, where an expected performance guarantee is obtained.*

- *5) We improve the algorithm by ensuring a deterministic approximation factor instead of in expectation.*

Problem Notation

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

Similarity Rules

Similarity/distance metric ($d_A(a,b) >= 0$): $d_A(a,b) = d_A(b,a) = 0$ iff a = b, where $a,b \in dom_I(A)$ are values on attribute A.

A **differential function** [A] on attribute A specifies a distance restriction by a range of metric distances over A.

Two tuples t_1 , t_2 in a relation I are compatible w.r.t. the differential function [A], denoted by (**t1,t2**$\asymp\varnothing[A]$) or (t1[A],t2[A])$\asymp\varnothing[A]$, if the metric distance of t_1 and t_2 on attribute A is within the range specified by [A].

A differential function may also be specified on a set of attributes X , say $\varnothing$**[X]**.

We call $\varnothing[A]$ a projection on attribute A of $\varnothing$**[X]**,A ϵ X.

A differential dependency (dd) over R has a form (**X** $\rightarrow$ **A,[X A]**) where X $\subseteq$ R are **determinant attributes**, A $\subseteq$ R is the **dependent attribute**, and [X A] is a differential function on attributes X and A.

We use **I |= (X** $\rightarrow$ **A,[X A])** to represent a relation I of R satisfies a dd.

I |= Σ, if I satisfies each dd in Σ.

Rule-based data imputation

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

A null cell in a tuple t_i is denoted by **t_i[A] =_** (e.g. t1[A]=_). It is regarded to be compatible with any other data w.r.t. distance restrictions ($I \models \Sigma$).

A filling I' of I is also an instance of 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 (e.g. t'4[Address]=t4[Address]="No402 Jordan Rd").

2) Satisfaction of DDs is still retained, having $I' \models \Sigma$.

3) The filling should not be a random guess without any supporting neighbors.

Filling a null cell on attribute A (by any value) will never introduce violations to any D in the form of $(X \rightarrow B, [X\ B]), A \neq B$.

Rule-based Data Imputation

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

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

Consider again dd_1:

dd_1 :(Name, Street —> Address, {[0,1],[0,9],[0,2]})

The relation I in Table 1 satisfies this dd_1, since the values of any two tuples in relation I satisfies the distance restrictions of dd1.

For example, 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 $(t_4, t_5) \asymp \varnothing[A\text{ddress}]$, 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 [Name,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[Address]** in Table 1 is thus t_1[Address]="#402 Jordan Rd", which is suggested by the Address value of **its similarity neighbor t_5**.

Problem Statement

Target: Fill more null cells as much as possible.

Filling Gain: The total number of cells filled in I'_A for I_A, $|\Delta(I'_A, I_A)|$, where $\Delta(I'_A, I_A)$ is the difference on cells between I_A and its filling I'_A.

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

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

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

Problem Statement

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

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

Consider the imputation on attribute Address in Table 1, with $I_A = \{t_1, t_2, t_3\}$.

Another dd_2, in $\mathbf{\Sigma_A = \{dd_1, dd_2\}}$, states that two Address values a_1 and a_2 in the same Street should be similar (with at most 3 different digits, having $d_{Address}(a_1, a_2) \leq 3$), such as "#402 Jordan Rd" and "#531 Jordan Rd" in tuples t_5 and t_6 in Table 1.

dd_2 :(Street —> Address, {[0,0],[0,3]})

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.

If t_1[Address] ="#402 Jordan Rd" and t_2[Address] ="No402 Jordan Rd", t_3[Address] cannot be filled by any value w.r.t. {dd1 , dd2 }, since t_2[Address] = "No402 Jordan Rd" and t_3 [Address] = "#531 Jordan Rd", w.r.t. dd1 , have distance 5 not in the range [0,3] specified by dd2.

If t_1[Address] ="#402 Jordan Rd" and t_2[Address] ="#402 Jordan Rd", t_3[Address] can be filled by"#531 Jordan Rd", shown in Table 3.

I'_A and I''_A are both maximal filling, but I''_A is the maximum filling and full filling.

Problem Statement

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.

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

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

Computing exact solutions

1. Local Neighbor Restrictions

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

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

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

2. Filling Candidates

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

can(t_i[A]) is a set of candidates that are compatible with all the neighbors of t_i.

3. Maximum Filling

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

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

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

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

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

where $x_{ij} = 1$ if the j-th candidate $a_j \in$ can(t_i) is selected to fill the null cell t_i[A], otherwise 0.

w_{il} and v_{jk} are constants such that $w_{il} = 1$ if tuples t_i, $t_l \in$ IA are in neighborhood, otherwise 0;

$v_{jk} = 0$ if $(a_j,a_k) \asymp \varnothing[A]$ [A](t_i,t_l),$a_j \in$ can(t_i), $a_k \in$ can(t_l), otherwise 1.

Computing Exact Solutions

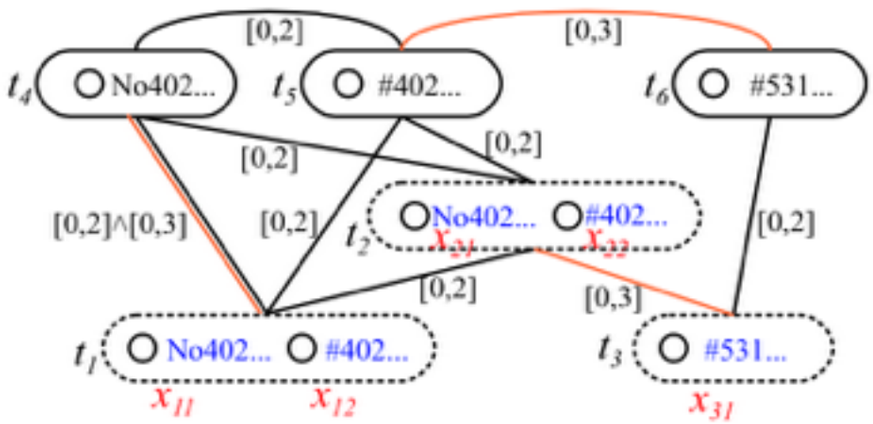


Figure 2: Filling candidates on attribute Address

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

1. 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., $(t_1, t_2) \asymp \varnothing[X]$, of dd_1 (in black).

In Figure 2 tuple pairs may have distinct local distance restrictions.

(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].

2. In Figure 2, solid line ellipses) suggest a set of all possible candidates (circles) for each null cell (dotted line ellipse)

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.

3. 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[Address] = t_2[Address] =#402. . . and t_3[Address] = #531. . . , which is exactly the full filling I''_A in Table 3.

Approximation Method

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

Referring to the hardness of approximation, 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 $0 \le x_{ij} \le 1$.

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
```

As shown in Lines 6-10 in Algorithm 1, a candidate a_j is assigned only if it is compatible with all the neighbors.

Approximation Method

Let the numbers attached to each candidate in Figure 3 denote a LP solution of Figure 2.

Especially, 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,

1) we first select #402. . . for t_2 and #531. . . for t_3, whose x_{ij} values are the highest 1.0.

2) 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.

3) 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.

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

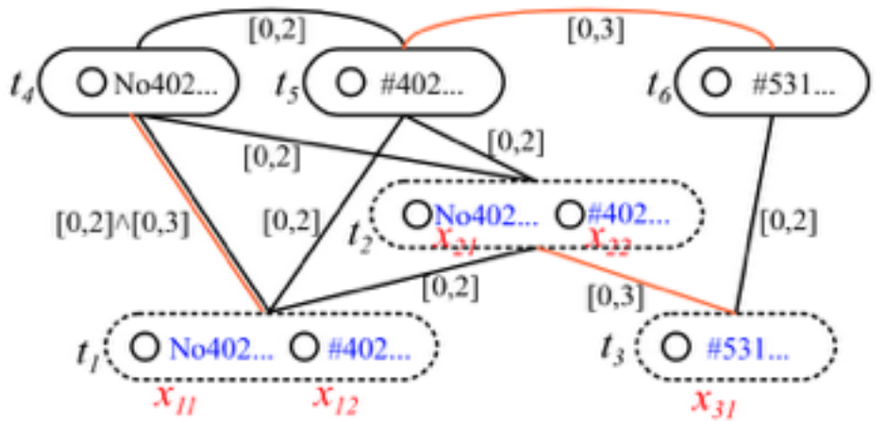


Figure 2: Filling candidates on attribute Address

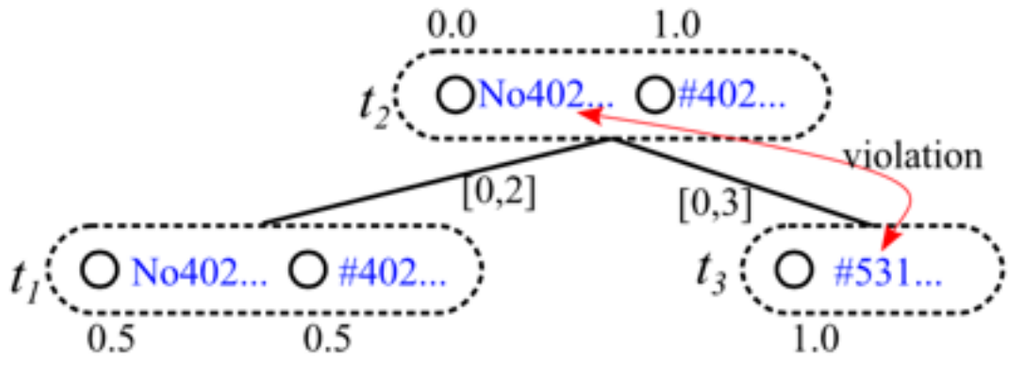


Figure 3: Filling by linear programming

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

Randomized Algorithm (Randomized Imputation)

For randomly sampling fillings, each candidate is associated with a probability of being selected as a filling. we may employ x_{ij} to interpret the probability of $a_j \in \text{can}(t_i)$ being selected.

where $\varepsilon > 0$ trades off the contribution of LP estimations.

When $\varepsilon = 0$, the probability is exactly proportional to x_{ij}.

If ε takes an extremely large positive, it approximately denotes equal probability having $\Pr[t'_i[A] = a_j] \approx 1/c_i$ for each $a_j \in \text{can}(t_i)$ and $\Pr[t'[A]=\_]=1/\varepsilon\, c_i$.

Algorithm 2 can find a full filling to any fully-fillable instance in expected time $O((1/p)^m)$, where p is the minimum probability of a candidate being selected.

All the possible maximal/maximum fillings are able to be sampled by the algorithm.

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

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

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

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

```
initialize probabilities
repeat
    for each t_i ∈ I_A do
        randomly draw a value a_j ∈ 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] = _]
        if t'_i[A] = a_j is in violation with any t'_l[A] then
            t'_i[A] := _
    if I'_A is not maximal then
        fill null cell by any valid candidate (not in violation)
    rank I'_A in the top-k list K
until ℓ times
return the top-1 filling in K with the highest filling gain
```

Randomized Algorithm (Randomized Imputation)

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

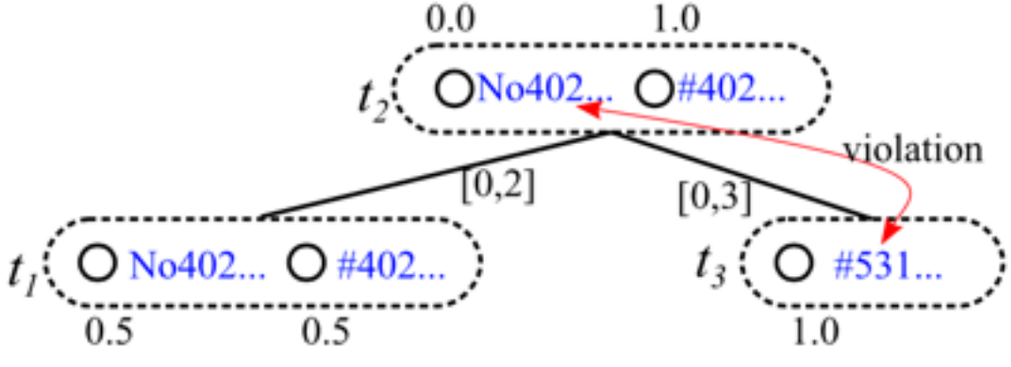


Figure 3: Filling by linear programming

1) 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. . . .

2) Since the remaining null cell in t_2 can be further filled, the current filling is not maximal.

3) 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.

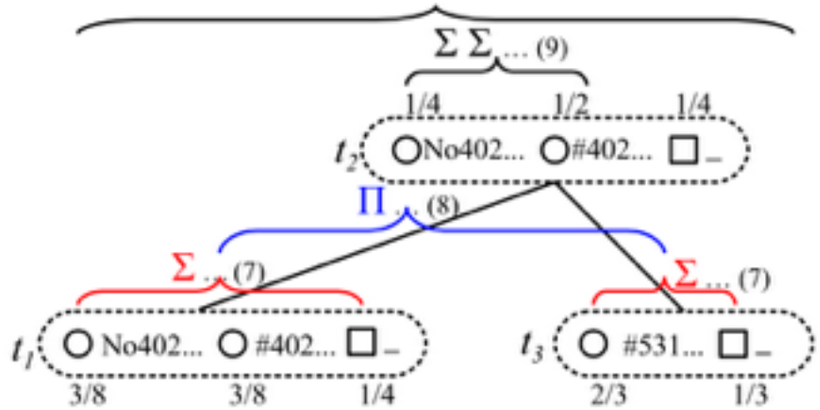


Figure 4: Random filling with probability

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

Expected Filling Gain E[W]

We analyze the bound of expected W , i.e., E[W], the number of null cells filled by random filling in expectation.

For any tuple t_l in I_A in neighborhood with t_i, i.e., $t_l \in I_A$, $(t_i, t_l \asymp \varnothing[A])$, the probability of $t_i[A] = a_j$ compatible with $t'_l[A]$ is shown in Equation 7.

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

The probability of $t'_i[A] = a_j$ compatible with all the tuples t_l in neighborhood with t_i is shown in Equation 8.

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

Considering all the non-null assignments $a_j \in \mathrm{can}(t_i)$ of each $t_i \in I_A$, with probability $\Pr[t_i[A] = a_j]$, E[W] can be computed by Equation 9.

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

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.

Randomized Algorithm

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

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

For the neighbor t_3, however, No402. . . is in violation with #531. . . , the probability of $t'[A]$ = No402. . . compatible with $t'[A]$ is

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

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

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

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. □

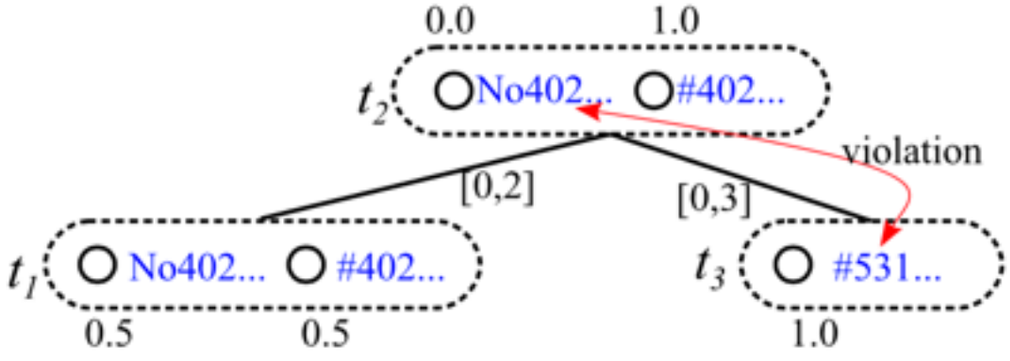


Figure 3: Filling by linear programming

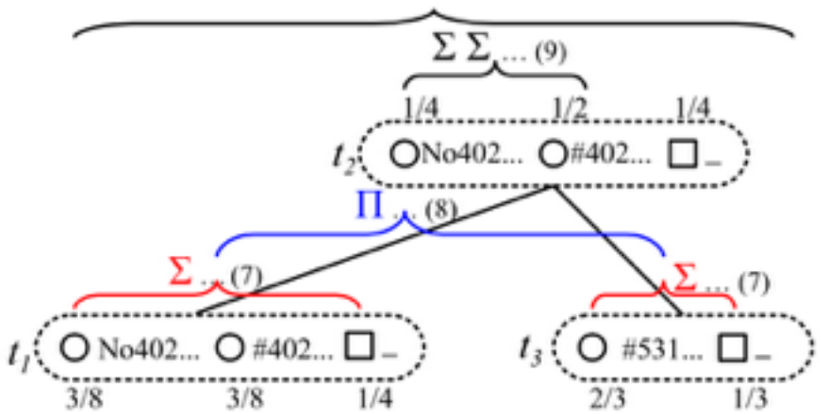


Figure 4: Random filling with probability

Derandomization — Conditional Expectation

Theorem 8 gives only the performance guarantee in expectation of the Random algorithm. Next, we advance the approach by giving a deterministic bound of approximation rather than 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 $\mathbf{E}[W \mid I_A'^{0}] = \mathbf{E}[W]$ initially, and $\mathbf{E}[W \mid I_A'^{m}]$ is the exact gain of the filling $I_A'^{m}$.

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 $\mathbf{E}[W \mid I_A'^{i}]$. There is no need to compute it from scratch. Instead, it can be calculated incrementally from $\mathbf{E}[W \mid I_A'^{i-1}]$ 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., $\mathbf{E}[W \mid I_A'^{i}] = 0$.

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

$$\begin{aligned}&\Pr\left[(t_l'[A] = a_k, I_A') \asymp \Phi[A] \mid I_A'^{i}\right]\\ =&\Pr\left[(t_l'[A] = a_k, I_A') \asymp \Phi[A] \mid I_A'^{i-1}\right].\end{aligned}$$

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

$$\Pr\left[(t_l'[A] = a_k, I_A') \asymp \Phi[A] \mid I_A'^{i}\right] = 0,$$

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

$$\begin{aligned}&\Pr\left[(t_l'[A] = a_k, I_A') \asymp \Phi[A] \mid I_A'^{i}\right] \qquad (10)\\ =&\frac{\Pr\left[(t_l'[A] = a_k, I_A') \asymp \Phi[A] \mid I_A'^{i-1}\right]}{\Pr\left[(t_l'[A] = a_k, t_i'[A]) \asymp \phi[A](t_l, t_i) \mid I_A'^{i-1}\right]}.\end{aligned}$$

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, $\mathbf{E}[W \mid I_A'^{i}]$ is computed.

Derandomization — Conditional Expectation

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.

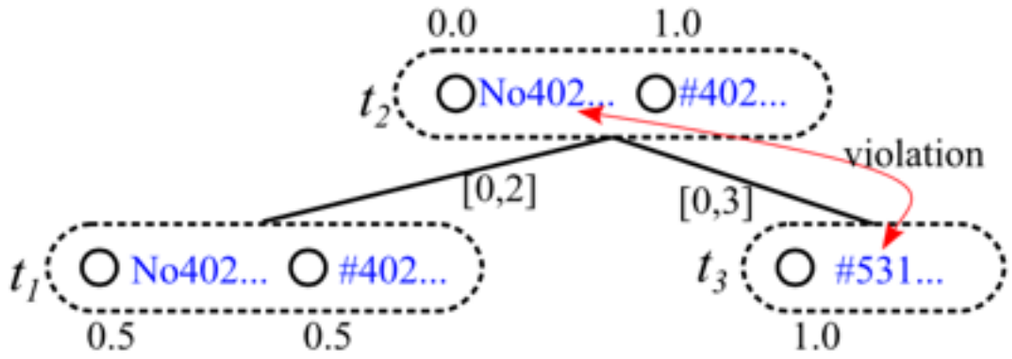


Figure 3: Filling by linear programming

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

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

For $\Pr\left[(t'_2[A] = \#402\ldots, I'_A) \asymp \Phi[A]\right] = 1$, we update it by dividing $\Pr[(t'_2[A] = \#402\ldots, t'_3[A]) \asymp \phi[A](t_2, t_3)\big] = 1$ 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 $\mathbf{E}[W \mid t'_3[A] = \#531\ldots] = \frac{3}{8} * 1 + \frac{3}{8} * 1 + \frac{1}{4} * 0 + \frac{1}{2} * 1 + 1 = \frac{9}{4}$. □

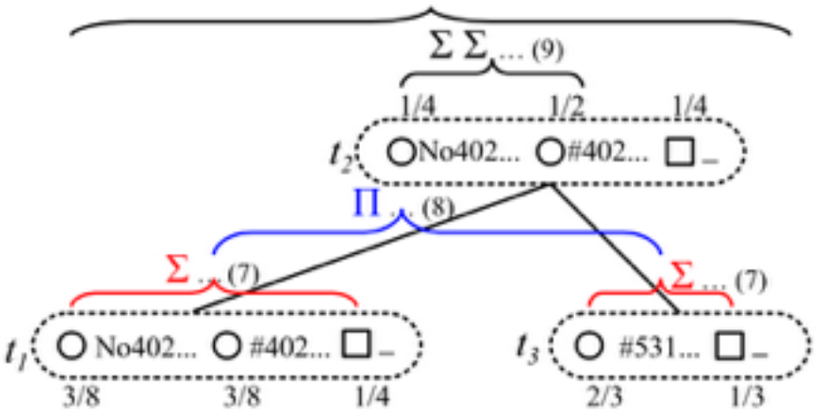


Figure 4: Random filling with probability

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.

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 $O(mc^2b)$.

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 $t_i \in I_A$ **do**
3: $\quad t'_i[A] :=\_$ {construct I'^{i}_{A}}
4: $\quad E_{\max} :=$ ConExp($\mathbf{E}[W \mid I'^{i-1}_{A}], t_i, \_$)
5: $\quad$ **for** each $a_j \in \mathsf{can}(t_i)$ **do**
6: $\quad\quad E :=$ ConExp($\mathbf{E}[W \mid I'^{i-1}_{A}], t_i, a_j$)
7: $\quad\quad$ **if** $E > E_{\max}$ **then**
8: $\quad\quad\quad E_{\max} := E$
9: $\quad\quad\quad t'_i[A] := a_j$
10: $\quad \mathbf{E}[W \mid I'^{i}_{A}] := E_{\max}$
11: **if** I'_A is not maximal **then**
12: $\quad$ 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: $E{:=}E{-}\sum_j \Pr[t'_i[A] = a_j] \Pr\left[(t'_i[A] = a_j, I'_A) \asymp \Phi[A] \mid I'^{i-1}_{A}\right]$
2: **if** $a_i \neq \_$ **then**
3: $\quad E := E + 1$
4: **for** each t_l in neighborhood with $t_i, l = i+1, \ldots, m$ **do**
5: $\quad$ **for** each $a_k \in \mathsf{can}(t_l)$ **do**
6: $\quad\quad E{:=}E - \Pr[t'_l[A]{=}a_k] \Pr\left[(t'_l[A]{=}a_k, I'_A) \asymp \Phi[A] \mid I'^{i-1}_{A}\right]$
7: $\quad\quad$ **if** $(a_i, a_k) \asymp \phi[A](t_i, t_l)$ **then**
8: $\quad\quad\quad$ update $\Pr[(t'_l[A]{=}a_k, I'_A) \asymp \Phi[A] \mid I'^{i}_{A}]$ by formula (10)
9: $\quad\quad\quad E{:=}E + \Pr[t'_l[A]{=}a_k] \Pr\left[(t'_l[A]{=}a_k, I'_A) \asymp \Phi[A] \mid I'^{i}_{A}\right]$
10: **return** E

Filling Guided by Conditional Expectation

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

$$\mathbf{E}[W \mid t'_2[A] = \#402\ldots, t'_3[A] = \#531\ldots] = \frac{11}{4},$$
$$\mathbf{E}[W \mid t'_2[A] = \mathsf{No402}\ldots, t'_3[A] = \#531\ldots] = 0,$$
$$\mathbf{E}[W \mid t'_2[A] =\_, t'_3[A] = \#531\ldots] = \frac{7}{4}.$$

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. □

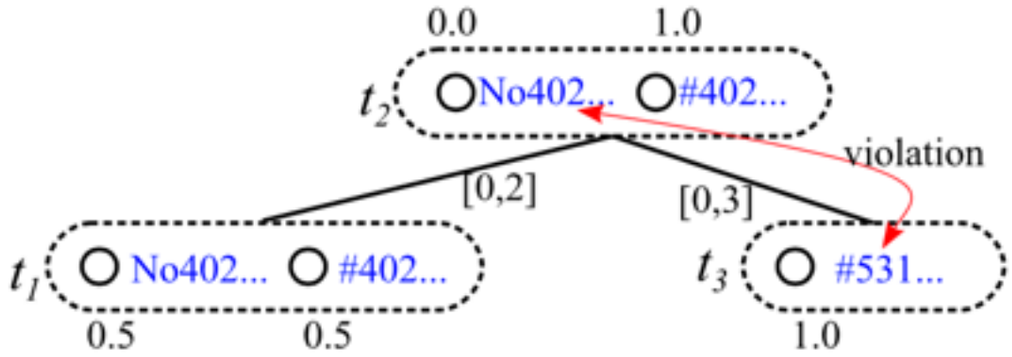


Figure 3: Filling by linear programming

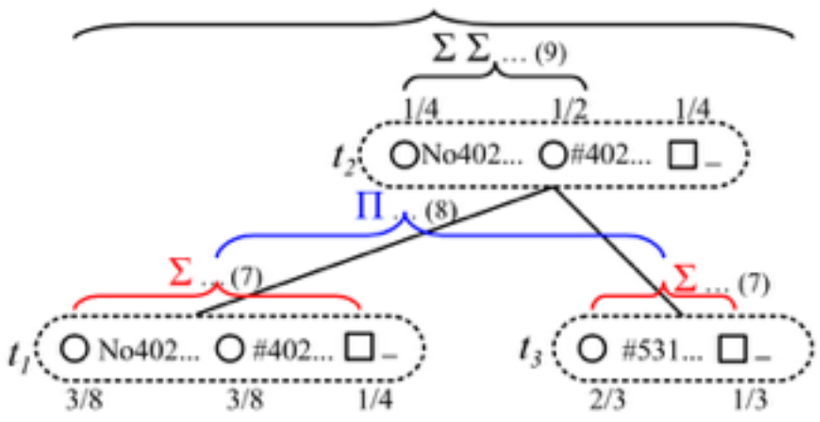


Figure 4: Random filling with probability

Experimental Setting

Let **truth** be the set of removed cell values.

Found be the set of filling results returned by imputation algorithms (not including the null cells that are failed to fill).

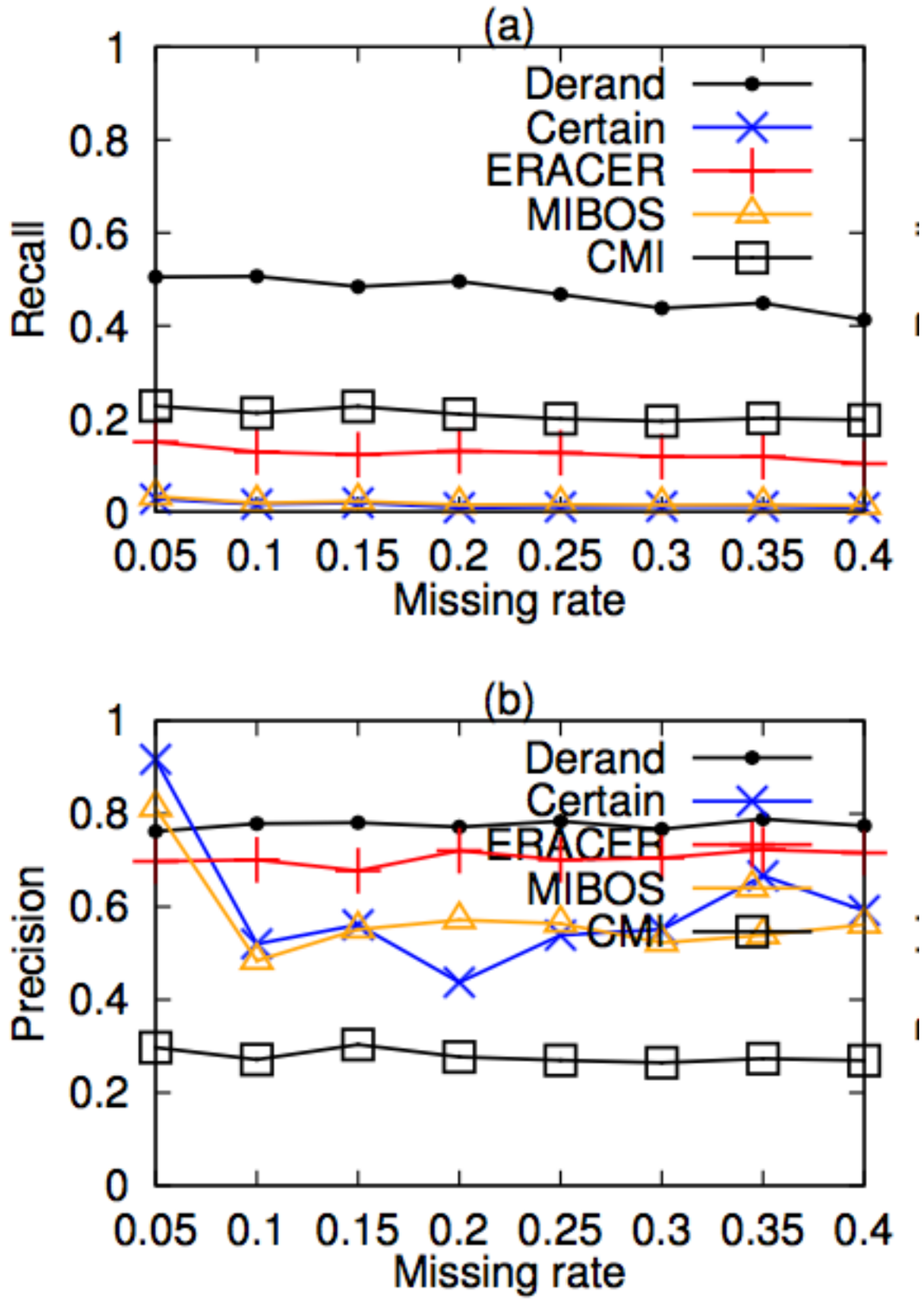
Recall accuracy is given by $\text{recall} = \frac{|\text{truth} \cap \text{found}|}{|\text{truth}|}$, i.e., the proportion of null cells that are accurately filled/recovered;

The **precision accuracy** is $\text{precision} = \frac{|\text{truth} \cap \text{found}|}{|\text{found}|}$, denoting the proportion of filled cells (non-null) that are correct;

f-measure$= 2 \cdot (\text{precision} \cdot \text{recall}) / (\text{precision} + \text{recall})$ represents an overall accuracy.

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

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 b, while the correspond- ing recall drops dramatically in Figure a.



Comparison with Existing Techniques

This experiment compares the proposed method to Certain fixes with **editing rules on equality neighbors**, **ER- ACER** based on statistics over equal values, **MIBOS** with tuple similarity defined on value equality, and **CMI** considering imputation between most similar tuples.

Figure 5 verifies 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).

In most tests in Figure 6(b)(e), the precision of Derand is comparable to (higher than) that of the equality-based methods.

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

Compared to CMI, as shown in Figure 6(a), the recall of Derand is even higher by explicitly using only the (subset of) attributes specified in dds.

As shown in Figure 6, our dds-based Derand shows comparable precision to the equality-based methods, but significantly higher re-call owing to the successful identification of value similarity.

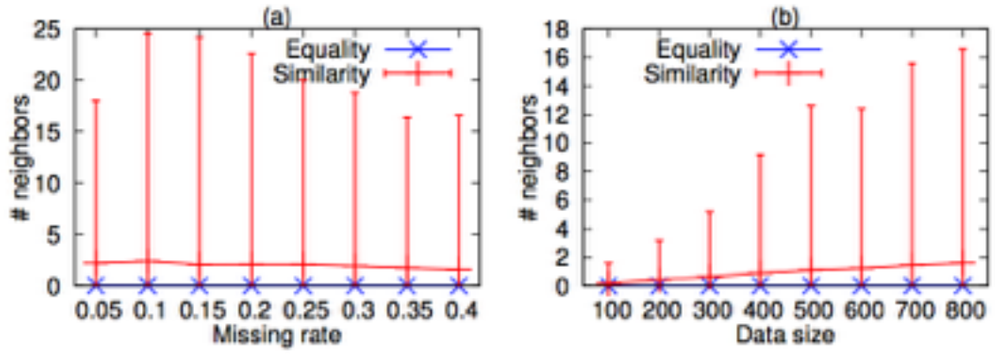


Figure 5: Number of neighbors (Restaurant)

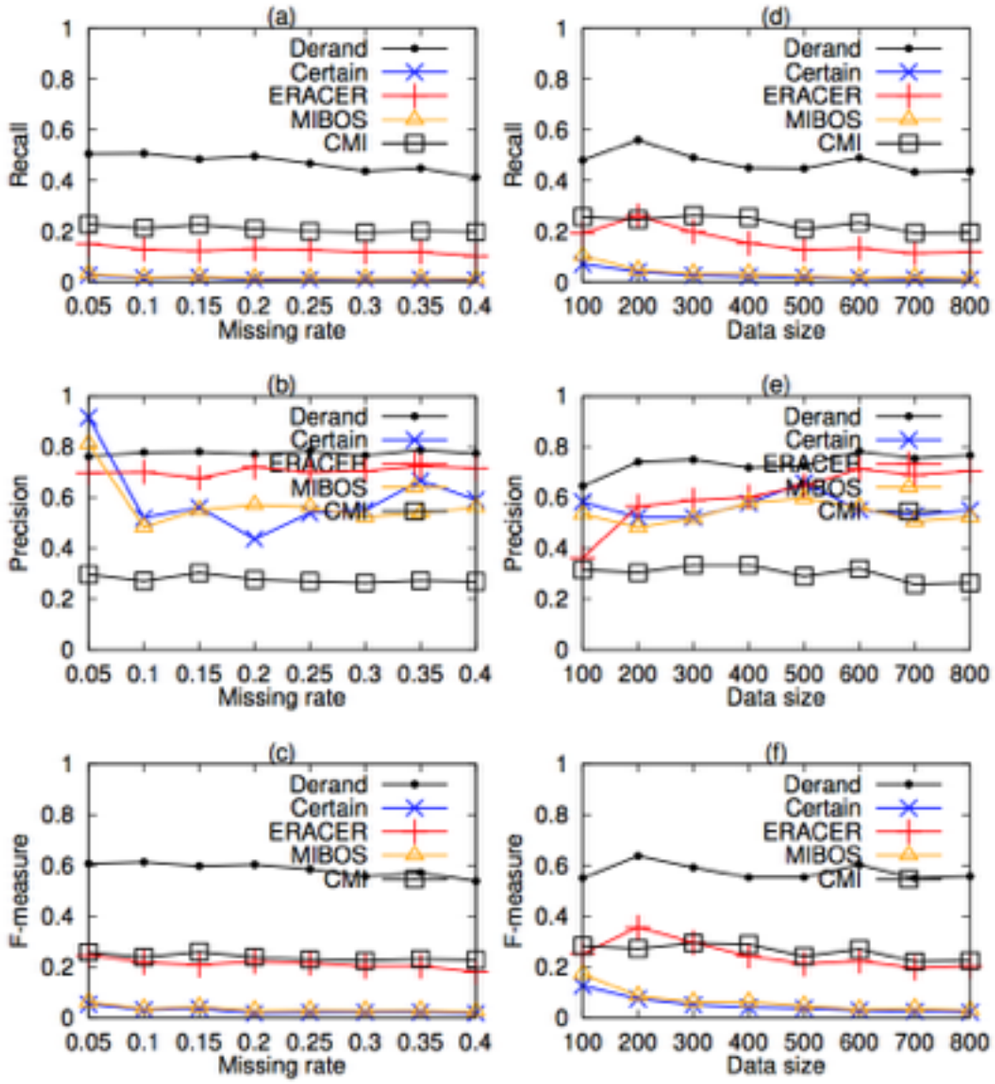


Figure 6: Imputation accuracy (Restaurant)

Comparison with Existing Techniques

Figure 7 reports the results on various duplication rates.

The proposed Derand already achieves considerably high accuracy when duplication rate is small, such as 2.

Besides, our similarity rule-based method can successfully utilize the similarity relationships between tuples, even they are not duplicates (denoting the same entity).

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.

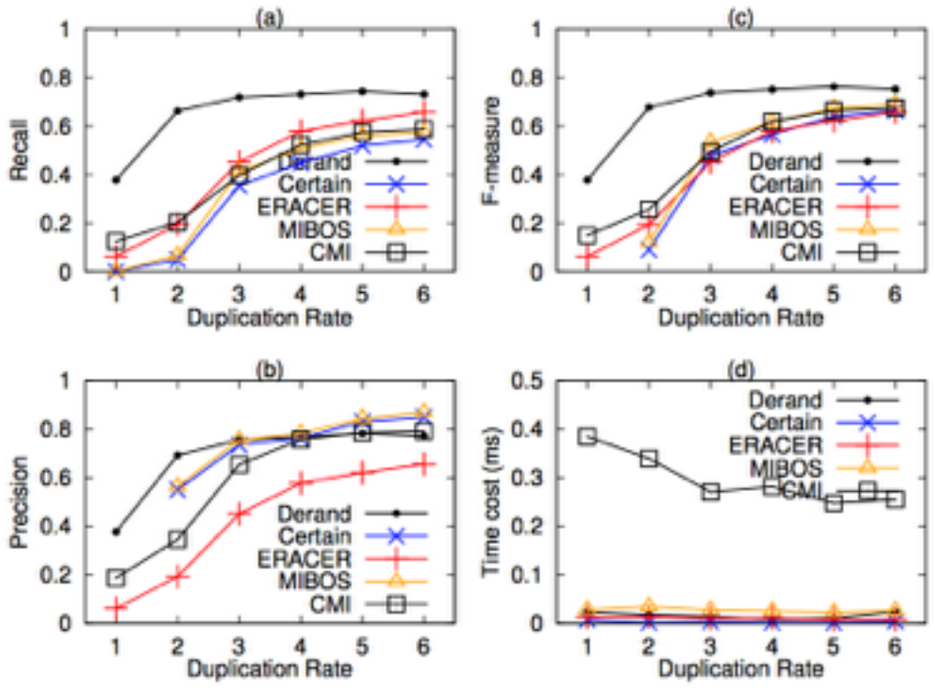


Figure 7: Varying duplication rates (Restaurant)

Figure 8 presents that 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).

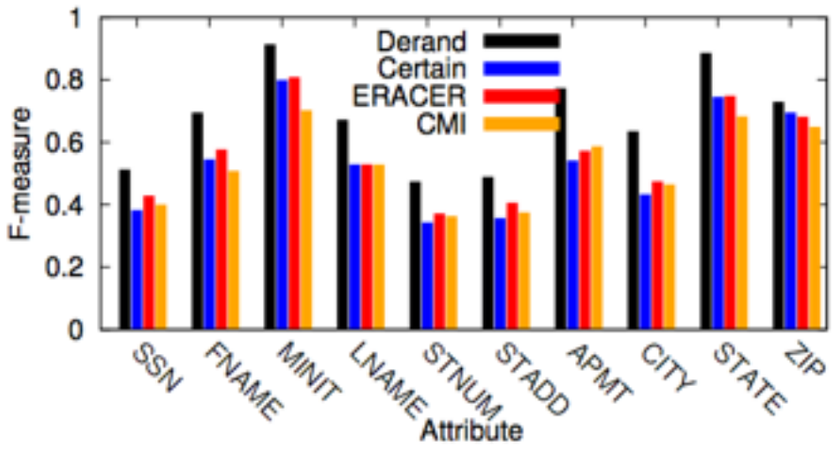


Figure 8: Comparison on various attributes (UIS)

Application in Record Matching

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 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.

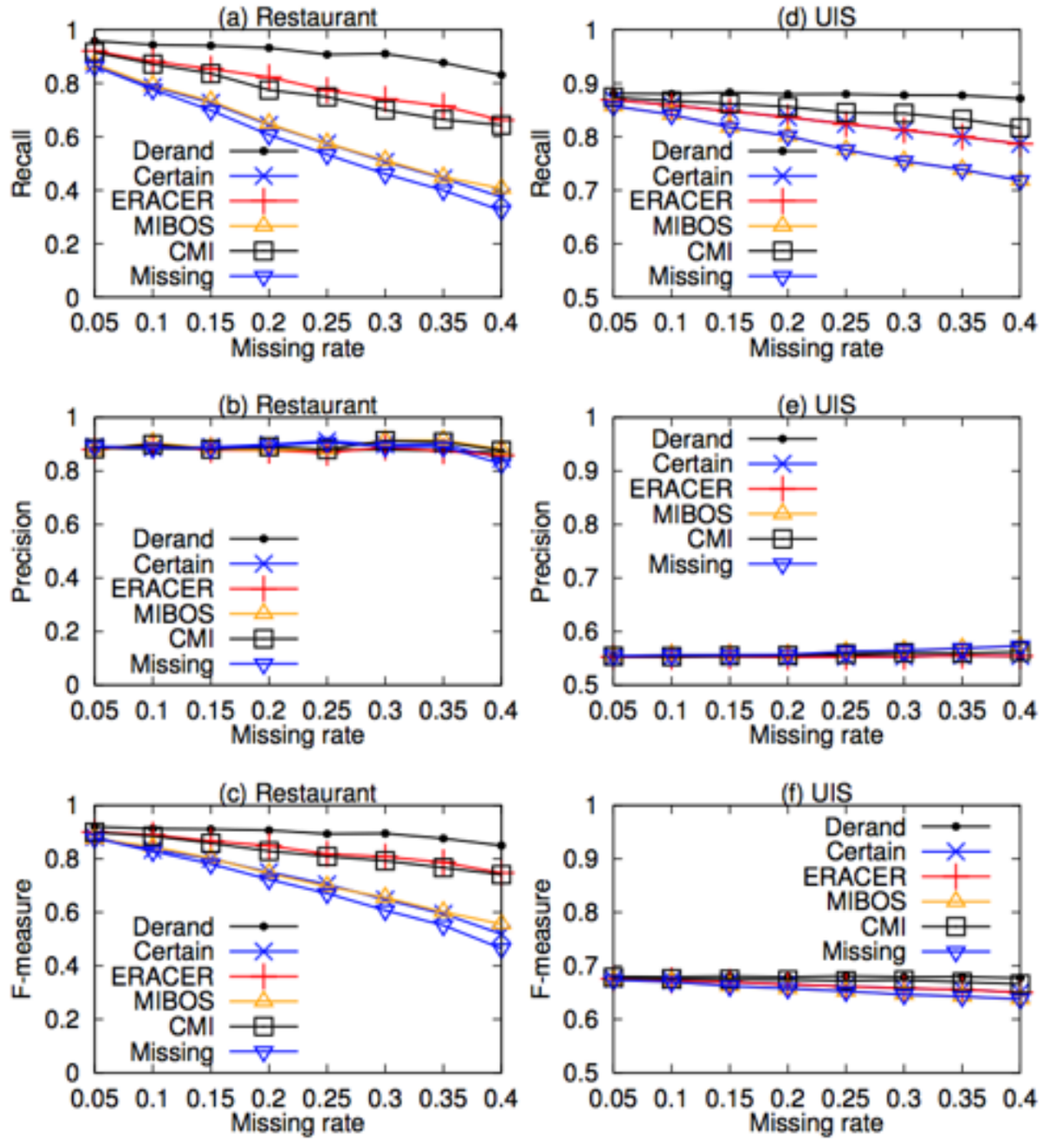


Figure 9: Application in record matching

Performance of Proposed Methods

As shown in Figure 12, Derand with either LP or LE can achieve filling gains almost the same as the maximum filling.

With the increase of missing rates, the time cost of LP-encapsulated approaches increases heavily in Figure 13(a).

Figure 14(a) shows that the time cost increases as the increase of data sizes.

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).

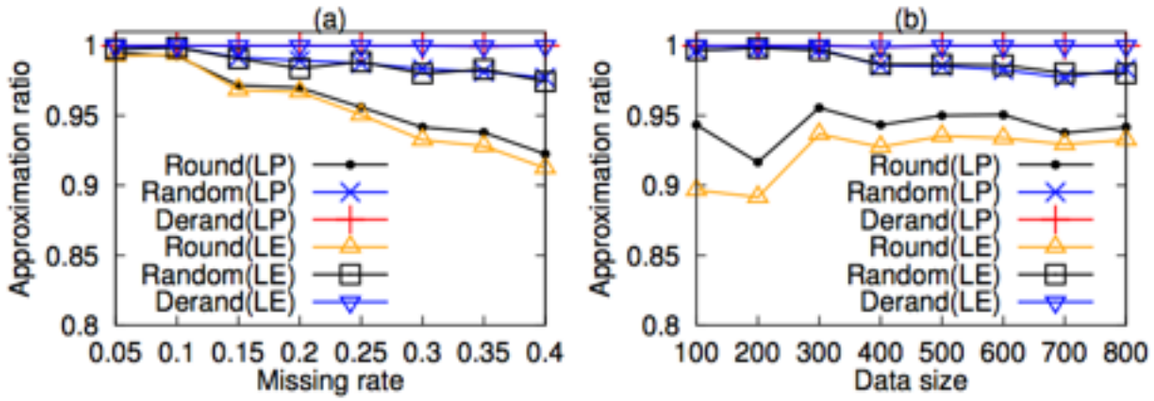


Figure 12: Approximation performance (Restaurant)

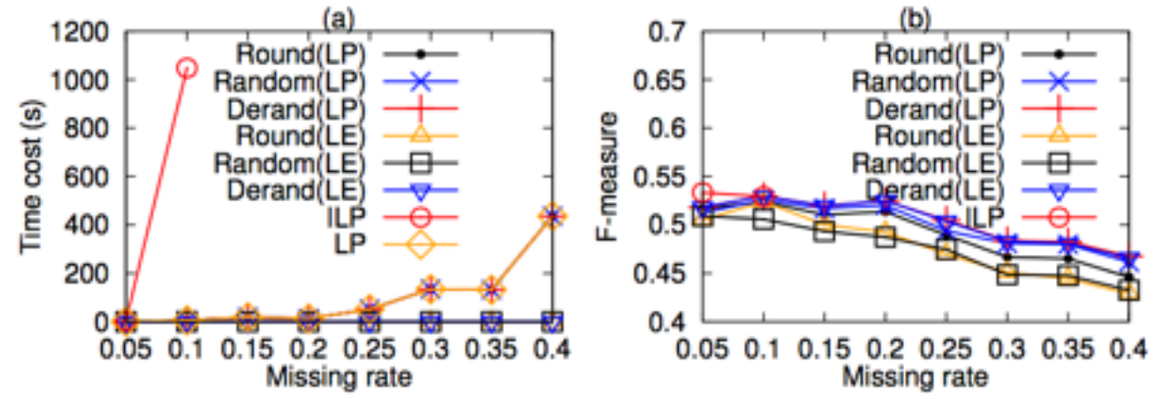


Figure 13: Varying missing rates (Restaurant)

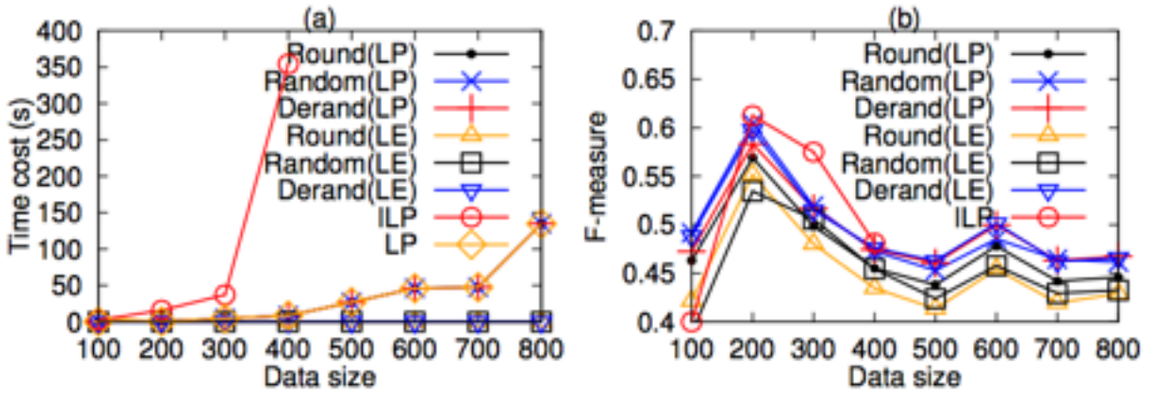


Figure 14: Scalability (Restaurant)

Conclusion

Imputing missing values of a tuple replies on others (neighbors) sharing the same information.

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.

The End

Thanks for watching and criticizing