Constraint-Variance Tolerant Data Repairing

Shaoxu Song   Han Zhu   Jianmin Wang
Tsinghua National Laboratory for Information Science and Technology
KLiss, MoE; School of Software, Tsinghua University, China
{sxsong, zhuhan10, jimwang}@tsinghua.edu.cn

ABSTRACT

Integrity constraints, guiding the cleaning of dirty data, are often found to be imprecise as well. Existing studies consider the inaccurate constraints that are oversimplified, and thus refine the constraints via inserting more predicates (attributes). We note that imprecise constraints may not only be *oversimplified* so that correct data are erroneously identified as violations, but also could be *overrefined* that the constraints overfit the data and fail to identify true violations. In the latter case, deleting excessive predicates applies.

To address the oversimplified and overrefined constraint inaccuracies, in this paper, we propose to repair data by allowing a small variation (with both predicate insertion and deletion) on the constraints. A novel θ-tolerant repair model is introduced, which returns a (minimum) data repair that satisfies at least one variant of the constraints (with constraint variation no greater than θ compared to the given constraints). To efficiently repair data among various constraint variants, we propose a single round, sharing enabled approach. Results on real data sets demonstrate that our proposal can capture more accurate data repairs compared to the existing methods with/without constraint repairs.

Keywords

Data repairing; denial constraints

1. INTRODUCTION

Automatic repairing techniques are often employed for cleaning dirty data, under the supervision of integrity constraints [9]. Typical repairing approaches (minimally) modify the data towards conformance to the constraints [4]. The repair performance thus heavily relies on trustable constraints. Unfortunately, it has been highlighted that both integrity constraints and data could be variant [18]. Recent studies [5, 2] also indicate the fact of inaccurate integrity constraints and inaccurate data co-existence and work on repairing constraints and data at the same time.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-3531-7/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2882903.2882955

Existing studies on simultaneously repairing constraints and data [5, 2] consider imprecise constraints (mainly *functional dependencies*, FDs) that are oversimplified. With such oversimplified constraints, data repairing may erroneously identify correct data as violations (see examples below). The constraint repairing is thus to *refine* the oversimplified constraints, via inserting more predicates (attributes in FDs).

As one of the most direct motivations of this study, we first note that, besides oversimplified, the imprecise constraints could also be overrefined. That is, too many predicates (attributes) are specified in a constraint so that it overfits the data. Owing to such overfitting, the overrefined constraints fail to identify some truly dirty data (also see examples below). In this case, constraint *simplification*, by deleting excessive predicates (attributes), may apply.

In this study, we propose to address both constraint variances, *oversimplified* and *overrefined*. By *oversimplified*, we mean too few predicates specified in a constraint, such that many (correct) data will be identified as violations to the constraint, e.g., φ_1 in Example 1. On the other hand, by *overrefined*, we mean too many predicates specified in a constraint, such that no (incorrect) data could be identified as violations to the constraint, e.g., φ_3 in Example 1.

Example 1. *Consider an example Income relation in Figure 1(a). To support the order relationships on numeric values such as* Income, Tax, *we use a general notation of* denial constraints *(*DCs*) [7]. (See Example 3 for* DCs *on numerical attributes. As special cases,* FDs *can be represented by* DCs*.)*

Suppose that the following DC *is given.*

$$\varphi_1 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})$$

It states that for any $t_\alpha, t_\beta \in R$, they should not *have the same* Name *but different* CP *(Cellphone number), i.e., an* FD Name→CP*. This constraint, stating that* Name *can determine* CP*, is obviously imprecise. With this oversimplified φ_1, many data values (in red) need to be modified in Figure 1(b), most of which are indeed correct.*[1]

Referring to [5, 2], a (refinement) repair of constraint is performed by adding attributes to the left-hand-side of FDs*, i.e., inserting predicates in* DCs *as shown in φ_2 below.*

$$\varphi_2 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name} \wedge t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})$$

With the additional Birthday*, φ_2 becomes precise. Consequently, the truly dirty values (only a small proportion with*

[1]Assume that the data repair on CP has the minimum change in this example for simplicity. Repairs on other attributes may be performed in practice.

	Name	Birthday	CP	Year	Income	Tax
t_1	Ayres	8-8-1984	322-573	2007	21k	0
t_2	Ayres	5-1-1960	***-389	2007	22k	0
t_3	Ayres	5-1-1960	564-389	2007	22k	0
t_4	Stanley	13-8-1987	868-701	2007	23k	3k
t_5	Stanley	31-7-1983	***-198	2007	24k	0
t_6	Stanley	31-7-1983	930-198	2008	24k	0
t_7	Dustin	2-12-1985	179-924	2008	25k	0
t_8	Dustin	5-9-1980	***-870	2008	100k	21k
t_9	Dustin	5-9-1980	824-870	2009	100k	21k
t_{10}	Dustin	9-4-1984	387-215	2009	150k	40k

(a)

	$\cdots$ CP
t_1	$\cdots$ 564-389
t_2	$\cdots$ 564-389
t_3	$\cdots$ 564-389
t_4	$\cdots$ 930-198
t_5	$\cdots$ 930-198
t_6	$\cdots$ 930-198
t_7	$\cdots$ 824-870
t_8	$\cdots$ 824-870
t_9	$\cdots$ 824-870
t_{10}	$\cdots$ 824-870

(b)

	$\cdots$ CP
t_1	$\cdots$ 322-573
t_2	$\cdots$ 564-389
t_3	$\cdots$ 564-389
t_4	$\cdots$ 868-701
t_5	$\cdots$ 930-198
t_6	$\cdots$ 930-198
t_7	$\cdots$ 179-924
t_8	$\cdots$ 824-870
t_9	$\cdots$ 824-870
t_{10}	$\cdots$ 387-215

(c)

	$\cdots$ CP
t_1	$\cdots$ 322-573
t_2	$\cdots$ 564-389
t_3	$\cdots$ 564-389
t_4	$\cdots$ 868-701
t_5	$\cdots$ ***-198
t_6	$\cdots$ 930-198
t_7	$\cdots$ 179-924
t_8	$\cdots$ ***-870
t_9	$\cdots$ 824-870
t_{10}	$\cdots$ 387-215

(d)

Figure 1: (a) dirty data, and possible repairs w.r.t. (b) oversimplified φ_1, (c) precise φ_2, (d) overrefined φ_3

*hidden digits ***) are repaired in Figure 1(c). (Note that predicates may not be inserted arbitrarily, see Section 2.2 for a detailed discussion on inserting meaningful predicates.)*

On the other hand, a given DC *could also be overrefined.*

$$\varphi_3 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name} \wedge t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year} \wedge t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})$$

The Year *information is not necessary to identify a person's* CP*. As illustrated in Figure 1(d), φ_3 overfits the data and fails to identify the truly dirty data in t_5 and t_8. By eliminating the excessive predicates on* Year *from φ_3, the modified constraint (i.e., φ_2 exactly) addresses all the dirty* CP *values.*

Compared to the existing approach [2] (with the trust parameter τ that controls the portion of trusted data), our proposal differs in the following aspects: (1) We employ both predicate insertion and deletion for repairing constraints, rather than only refining constraints (by insertion) without addressing overfitting ones; (2) We consider the advanced refinement with order relationships on numerical values in addition to the (in)equality relationship, e.g., $>$ refines $\neq$, which is not considered in the existing FD-based methods (see more examples in Example 4); (3) Instead of pre-setting a parameter τ ahead on the portion of trusted data, we can try several constraint variations (with different thresholds θ) and select the repair result having a moderate amount of changed data, with neither oversimplified nor overrefined constraints (see more discussions in Section 5.1).

In this paper, we propose a novel *θ-tolerant* repair model, with tolerance on constraint variances: Given a set Σ of constraints over a data instance I, the *θ-tolerant* repair model finds a minimum data repair I' that satisfies at least one of the constraint variants Σ' whose variation upon the given Σ (via predicate insertion and deletion) is within θ.

Challenges. Owing to the inherent hardness of data repairing problems [14, 16], it is not surprising to understand the challenge of the θ-tolerant repair problem (in Section 2.3).

To tackle the θ-tolerant repair problem, a straightforward approach is to compute the data repair results for each constraint *variant* Σ' with variation no greater than θ. A potentially large number of possible constraint repairs (variants) naturally lead to three questions: (1) how to *prune* the constraint variants, (2) how to enable the *sharing* of data repair results among different constraint variants, and (3) how to choose the constraint-variance tolerance threshold θ and the corresponding repair results.

Existing holistic data repair [8] eliminates only the currently observed violations, which may introduce new violations to others; and thus needs multiple rounds of repairing till no new violations coming up. Directly applying the holistic data repair falls short in the following aspects: (1) the multi-round repairing could be inefficient and with possibly unbounded repair cost (so that pruning is not applicable); (2) the ad-hoc repairing in distinct rounds prevents sharing the repair computation among different constraint variants.

Contributions. Our major contributions in this paper are summarized as:

(1) We propose a novel θ-tolerant repair, the first repair model with tolerance of constraint variances on both predicate insertion and deletion.

(2) We devise efficient pruning of possible constraint variants Σ'. By revealing the refinement relationships between constraints, we show that those non-maximal constraint variants w.r.t. refinement are not necessary to be considered. Upper and lower bounds of data repair cost further advance the pruning of candidate constraint variants.

(3) We present a violation-free algorithm for repairing data given a set of constraints (variants). The proposed method guarantees one round repairing without introducing new violations, by putting not only violations but also the suspect data (that may introduce violations after repairing) into the repair context. This one round data repairing also enables sharing repair results among different constraint variants.

(4) Finally, experimental results on real data sets demonstrate that our constraint-variance tolerant (CVtolerant) repair has significantly higher data repair accuracy than the state-of-the-art Unified [5] and Relative [2] approaches. In addition, our proposal shows about 2 orders of magnitude improvement in time costs compared to Relative. Owing to the effective pruning techniques, the time cost of CVtolerant is even comparable to the pure Holistic data repair [8] (which does not consider constraint variants).

2. PROBLEM STATEMENT

Consider a relation scheme R with attributes $attr(R)$. Let predicate space $\mathbb{P}$ be a set of predicates P in the form of $v_1\phi v_2$ or $v_1\phi c$ with $v_1, v_2 \in t_x.A$, $x \in \{\alpha, \beta\}$, $t_\alpha, t_\beta \in R$, $A \in attr(R)$, c is constant, and $\phi \in \{=, <, >, \leq, \geq, \neq\}$ is a built-in operator[2]. A *denial constraint* (DCs) [7]

$$\varphi : t_\alpha, t_\beta, \cdots \in R, \neg(P_1 \wedge \cdots \wedge P_m)$$

[2]The determination of meaningful predicates in $\mathbb{P}$ over the same attributes or joinable and comparable attribute pairs has been studied in [7], which is not the focus of this study.

states that for any tuples $t_\alpha, t_\beta, \dots$ from R, all the predicates $P_i \in pred(\varphi), i = 1, \dots, m$, should not be true at the same time.

For an instance I of R, we say that a list of tuples $t_i, t_j, \dots \in I$ *satisfies* φ, denoted by $\langle t_i, t_j, \dots \rangle \vDash \varphi$, if there exists at least one $P \in pred(\varphi)$ that is not satisfied, e.g., having $I(t_i.A)\overline{\phi}I(t_j.B)$ for a $P : t_\alpha.A\phi t_\beta.B$. Here $I(t_i.A)$ denotes the value of cell $t_i.A$ in I, and $\overline{\phi}$ is the inverse of ϕ (see Table 1). If $\langle t_i, t_j, \dots \rangle$ satisfies all the predicates in $pred(\varphi)$, it is a violation of φ.

We say that I satisfies φ, written as $I \vDash \varphi$, if all the distinct tuple lists in I satisfies φ. For a set Σ of φ, $I \vDash \Sigma$ if and only if $\forall \varphi \in \Sigma$, $I \vDash \varphi$.

Example 2. *For φ_1 in Example 1, the tuple pair t_1, t_2 in Figure 1(a) violates φ_1 since $t_1.\mathsf{Name} = t_2.\mathsf{Name}$ whereas $t_1.\mathsf{Income} \neq t_2.\mathsf{Income}$, i.e., all predicates in $pred(\varphi_1)$ are true. Instead, tuple pair t_1, t_4 satisfies φ_1, i.e., $\langle t_1, t_4 \rangle \vDash \varphi_1$, given $t_1.\mathsf{Name} \neq t_4.\mathsf{Name}$. In conclusion, I in Figure 1(a) does not satisfy $\Sigma = \{\varphi_1\}$ in Example 1.*

2.1 Data Repair via Value Modification

Data repairing is to find a modification I' of I such that all the violations w.r.t the constraints Σ are eliminated, having $I' \vDash \Sigma$. Referring to the minimum change principle [4], the data repair I' is expected to be as close as possible to the original I, i.e., minimizing the data repair cost below.

Definition 1. *Let I' denote a repair of I by modifying the attribute values without adding or deleting tuples. The data repair cost is evaluated by the distance between I and I'*

$$\Delta(I, I') = \sum_{t \in I, A \in attr(R)} w(t.A) \cdot dist(I(t.A), I'(t.A))$$

where $dist(I(t.A), I'(t.A))$ is the distance between two values on cell $t.A$ in I and I', and $w(t.A)$ is the weight of cell $t.A$.

The data repair cost could be simply the *count* of cells $t.A$ such that $I(t.A) \neq I'(t.A)$, i.e., with $dist(I(t.A), I'(t.A)) = 1$, while $dist(I(t.A), I'(t.A)) = 0$ if $I(t.A) = I'(t.A)$. Alternatively, the absolute value of difference on numerical values or the edit distance [17] for string values could be considered. The weight $w(t.A)$ can either be the confidence of the original value in cell $t.A$, or simply an equal 1 if no advanced knowledge is provided [14].

Note that there may not exist a repair I' with values from the currently known values $dom(A)$ of each attribute $A \in attr(R)$ that eliminates all the violations. Following the same line in [14], an assignment of *fresh variable* (*fv*) out of the currently known domain $dom(A)$ is necessary. As stated in [8], a fresh variable *fv* is a value not in the currently known domain $dom(A)$ and does not satisfy any of the predicates, i.e., ensures eliminating violations. It is worth noting that a denial constraint requires all the predicates should not be true at the same time. That is, tuples do not satisfy a *denial* constraint, e.g., φ_4 below, if they satisfies all the predicates specified in φ_4. Therefore, in order to make the tuples satisfy φ_4, we need some modification on the tuples such that they do not satisfy at least one predicate in φ_4. By specifying that "an fv does not satisfy any of the predicate", it ensures the repaired tuples satisfy the denial constraints. Usually, a cell is repaired to *fv* only when there is no other alternative from $dom(A)$, and is thus associated with higher cost, having $dist(a, b) < dist(a, fv), a, b \in dom(A)$.

Example 3. *Consider another φ_4 over the relation Income:*

$$\varphi_4 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Income} > t_\beta.\mathsf{Income} \wedge t_\alpha.\mathsf{Tax} \leq t_\beta.\mathsf{Tax})$$

which states that higher income should pay more tax. Tuples $t_1 - t_7$ violate φ_4 with each other, for instance, having $t_2.\mathsf{Income} > t_1.\mathsf{Income}$ and $t_2.\mathsf{Tax} \leq t_1.\mathsf{Tax}$, i.e., $\langle t_2, t_1 \rangle \nvDash \varphi_1$. Suppose that there is a data repair I' with the following modifications, on attribute Tax of t_2, t_3, t_5, t_6, t_7. (As we will see soon in Example 4, this is not a rational repair owing to the imprecise φ_4.)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Tax	*0*	fv_1	fv_2	3k	fv_3	fv_4	fv_5	21k	21k	40k

Note that $I(t_1.\mathsf{Tax}) = 0$ requires $I(t_2.\mathsf{Tax}) > 0$ referring to φ_4, while $I(t_4.\mathsf{Tax}) = 3k$ indicates $I(t_2.\mathsf{Tax}) < 3k$. However, there is no value in the current $dom(\mathsf{Tax}) = \{0, 3k, 21k, 40k\}$ which is > 0 and $< 3k$. Therefore, the repair $I'(t_2.\mathsf{Tax}) = fv_1$ is assigned by a fresh variable fv_1 (out of $dom(\mathsf{Tax})$, e.g., denoting $\mathsf{unknown}$ or $\mathsf{not\ applicable}$).

Suppose that the count cost is used, having $dist(a, a) = 0, dist(a, b) = 1, dist(a, fv) = 1.1, a \neq b, a, b \in dom(A)$ and fv is a fresh variable. According to the cost function, the total modification cost by this I' is $\Delta(I, I') = 5.5$.

2.2 Constraint Variation via Insertion/Deletion

We consider two types of reasonable constraint variances, via predicate insertion and deletion, respectively.

2.2.1 Predicate Insertion

First, following the same line of adding attributes for refining FDs [5, 2], by adding additional predicates (from predicate space $\mathbb{P}$) into a constraint, some tuples may no longer violate the modified constraint. However, we should not insert two types of predicates that make the DCs useless, (1) insertion leading to trivial DCs, and (2) insertion of predicates with constants, which is not an issue for FDs.

Specifically, if a $P_j : x\phi_j y$ is inserted, such that there exist $P_i : x\phi_i y \in pred(\varphi)$ having $\overline{\phi_i} \in Imp(\phi_j)$ as defined in Table 1, the repaired φ' becomes a *trivial* DC [7]. Since P_i, P_j always cannot be true at the same time, φ' is trivially satisfied. In other words, no violations can be detected by such a trivial DC. Similarly, inserting a predicate $t_\alpha.K = t_\beta.K$ over a declared key attribute K makes any DC trivially satisfied as well, and thus should be avoided. For the same reason, we do not consider inserting predicates with constants either. For instance, inserting a predicate $t_\alpha < 0$ makes all DCs fail to detect any violation in the Income relation in Figure 1.

2.2.2 Predicate Deletion

On the other hand, besides the oversimplified DCs (need predicate insertion), the constraints may already overfit the data (have to remove unnecessary predicates).

However, if too many predicates are eliminated from a DC, the constraint changes from overrefined to oversimplified. Indeed, the more the predicates are deleted, the higher the data repair cost will be (as stated in the following Lemma 1). Hence, a low cost data repair with moderated predicate deletion is preferred (see details in the problem definition below).

2.2.3 Constraint Variation Cost

It is worth noting that the variation of a constraint may need both predicate insertion (for fixing over-simplification) and predicate deletion (for fixing over-refinement).

For example, consider another φ_7.

$$\varphi_7 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name} \wedge t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})$$

To fix this constraint, we need to remove $t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year}$, and insert $t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday}$ in the variant φ'_7 of φ_7. While predicate insertion reduces unnecessary data repairs and the corresponding data repair cost, predicate deletion identifies new violations and increases the data repair cost. In this sense, the "common effect" between predicate insertion and deletion is that, the changed data will be smaller by inserting more predicates (deleting fewer predicates) than that of inserting fewer predicates (deleting more predicates). The data repair cost function consider the effect of predicate insertion and deletion holistically in φ'_7. Similarly, we need a constraint variation cost function to holistically represent the effect of predicate insertion and deletion in φ'_7 as well: (1) To avoid inserting too many predicates such that the constraints overfit the data (*smaller* data change), the constraint variation cost counts *positively* the predicate insertion and should be bounded. (2) To reward the contribution of identifying new violations (*larger* data change), we count *negatively* the predicate deletion towards the cost of constraint variation.

We call Σ' a variant of Σ if all $\varphi' \in \Sigma'$ are obtained by inserting/deleting predicates of a corresponding $\varphi \in \Sigma$. While $\Delta(I, I')$ denotes the cost on modifying data values, we use $\Theta(\Sigma, \Sigma')$ to represent the modification of constraints, in order to avoid potential confusion between data and constraint modifications.

Definition 2. *For a variant Σ' of Σ, the constraint variation cost is defined as*

$$\Theta(\Sigma, \Sigma') = \sum_{\varphi \in \Sigma} edit(\varphi, \varphi')$$

where φ' is a variant of φ, and $edit(\varphi, \varphi')$ is the corresponding cost.

The cost of changing φ into φ' by inserting/deleting predicates is given by

$$edit(\varphi, \varphi') = \sum_{P \in \varphi \setminus \varphi'} c(P) + \lambda \sum_{P \in \varphi' \setminus \varphi} c(P) \qquad (1)$$

where $c(P)$ denotes the weighted cost of predicate P, and λ is a weight of a predicate deletion relative to a predicate insertion, having $-1 \leq \lambda \leq 0$, e.g., $\lambda = -0.5$. It is not suggested to count predicate deletion by $\lambda = -1$ so that predicate substitution may have 0 cost.

Example 4 (Example 3 continued). *Note that φ_4 is imprecise, since low-income people do not have to pay tax (equal 0). To address this scenario, we refine φ_4 by substituting predicate $t_\alpha.\mathit{Tax} \leq t_\beta.\mathit{Tax}$ with $t_\alpha.\mathit{Tax} < t_\beta.\mathit{Tax}$, having*

$$\varphi'_4 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathit{Income} > t_\beta.\mathit{Income} \wedge t_\alpha.\mathit{Tax} < t_\beta.\mathit{Tax}).$$

By deleting and inserting a predicate in φ_4, we have the constraint variation cost $edit(\varphi_4, \varphi'_4) = c(\{t_\alpha.\mathit{Tax} < t_\beta.\mathit{Tax}\}) - \frac{1}{2}c(\{t_\alpha.\mathit{Tax} \leq t_\beta.\mathit{Tax}\}) = \frac{1}{2}$, where each predicate has a unit cost $c(P) = 1$ for simplicity and $\lambda = -\frac{1}{2}$.

Finally, with this repaired φ'_4, the irrational data repair I' in Example 3 can be avoided. Indeed, the minimum data repair w.r.t. φ'_4 only needs to modify one cell, $I'(t_4.\mathit{Tax}) = 0$.

Besides unit cost, it is possible to employ the distribution information to distinguish the contribution of attributes (predicates) when evaluating the constraint variance of adding and removing predicates. Consider a weighted cost $c(P)$ of a predicate P w.r.t. a constraint φ

$$c(P) = |\Pr(P) - \Pr(\varphi)|. \qquad (2)$$

Intuitively, $\Pr(\varphi)$ estimates the proportion of tuple pairs satisfying φ, while $1-\Pr(P)$ denotes the proportion of tuple pairs not satisfying P, or equivalently the proportion of tuple pairs that will satisfy the constraint by adding P into φ (referring to the denial semantics in the constraint). The more similar the distributions of tuple pairs satisfying P and φ (leading to more similar $\Pr(P)$ and $\Pr(\varphi)$ as well), the more likely the constraint will be true after adding P into φ, i.e., higher contribution or lower cost of P.

For instance, consider φ_1 with predicate $t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP}$ in Example 1. A predicate $P : t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday}$ on attribute Birthday, which has a distribution coinciding better with CP, will leads to more similar $\Pr(P : t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday})$ and $\Pr(\varphi : \neg(\cdots \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP}))$. That is, the predicate P on attribute Birthday has higher contribution to φ and lower cost when added into φ.

In contrast, $t_5[\mathsf{Year}] \neq t_6[\mathsf{Year}]$ but with similar CP values (or more precisely $t_5[\mathsf{CP}] = t_6[\mathsf{CP}]$ in truth data) would contribute negatively in observing similar $\Pr(t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year})$ and $\Pr(\varphi : \neg(\cdots \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP}))$ values. In other words, the predicate on attribute Year has lower contribution to φ and higher cost when added into φ.

We have weighed costs $c(P : t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday}) \leq c(t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year})$. The weighted cost similarly applies to predicate removing, following the intuition that removing a higher contribution predicate should have higher cost. Referring to the cost function, for an overrefined φ_3 in Example 1, removing a predicate $P : t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday}$ on attribute Birthday, which has a distribution coinciding better with CP, will leads to higher removing cost, i.e., $\lambda c(t_\alpha.\mathsf{Year} = t_\beta.\mathsf{Year}) \leq \lambda c(P : t_\alpha.\mathsf{Birthday} = t_\beta.\mathsf{Birthday})$.

2.3 θ-tolerant Repair Model

Let Φ_i denote the set of possible variants of a constraint φ_i by inserting/deleting predicates. Consider $\mathbb{D} = \Phi_1 \times \cdots \times \Phi_{|\Sigma|}$, where each $\Sigma' \in \mathbb{D}$ corresponds to a possible variant of Σ. By θ-tolerant, we mean allowing a small variation on the constraints Σ, i.e., having constraint variation cost no greater than θ, $\Theta(\Sigma, \Sigma') \leq \theta$.

We take data repairing cost as objective function (to minimize) and constraint repairing cost as constraint θ, following two intuitions: (1) to avoid over-refinement, the constraint variation (cost on inserting and deleting predicates in Definition 2) should not exceed a threshold θ; (2) to avoid over-simplification, i.e., identifying too many correct data as violations, the repairing follows the minimum change principle[3] over data.

Problem 1. *Given a set Σ of constraints and an instance I, the θ-tolerant repair problem is to find a data repair I' of I such that (1) $\Delta(I, I')$ is minimized, and (2) $I' \models \Sigma'$, for some constraint variant Σ' of Σ with $\Theta(\Sigma, \Sigma') \leq \theta$.*

[3]The minimum change principle is widely considered in data cleaning [4], under the assumption that people and machines always try to make mistakes as few as possible.

Table 1: Operator inverse and implication

ϕ	$=$	$\neq$	$>$	$<$	$\geq$	$\leq$
$\overline{\phi}$	$\neq$	$=$	$\leq$	$\geq$	$<$	$>$
Imp(ϕ)	$=, \geq, \leq$	$\neq$	$>, \geq, \neq$	$<, \leq, \neq$	$\geq$	$\leq$

Similar to data value modification that does not allow adding or deleting tuples, the constraint modification does not consider inserting or removing entire constraints either. The definition of denial constraints requires at least one predicate. Therefore, removing all predicates is not supported. Indeed, we can hardly imagine the meaning of $\neg()$ without any predicate.

While turning everything into a fresh variable definitely forms a feasible repair, it is not necessary. Instead, turning one attribute value into a fresh variable is sufficient to eliminate the violation to a denial constraint (recall that an fv does not satisfy any of the predicate). Referring to the target of minimizing the repair cost in Problem 1, such a repair of turning everything into a fresh variable will never be returned as an optimal repair.

Finding the θ-tolerant minimum repair is highly non-trivial. Indeed, for a fixed set of DCs, the minimum repair problem has already been known to be NP-hard [16]. Owing to this inherent hardness (in the case without constraint modification), it is not surprising to recognize the challenge in the θ-tolerant repair problem. A straightforward approach is to obtain all the possible constraint variants Σ' with $\Theta(\Sigma, \Sigma') \leq \theta$, and find the minimum data repair I' w.r.t. each Σ', respectively.

In the remainder of this paper, we show that not all the constraint variants need to be considered (in Section 3), and sharing of data repairs can be enabled among different constraint variants (in Section 4). For the determination of the constraint variance tolerant threshold θ, we introduce some guideline on the number of repaired cells (in Section 5.1).

3. CONSTRAINT VARIATION

It is worth noting that not all the constraint variants $\Sigma' \in \mathbb{D}$ with cost $\Theta(\Sigma, \Sigma') \leq \theta$ are necessarily to be considered. In the following, we illustrate that some constraint variant may refine another. It enables the pruning of constraint variants that will not generate minimum data repairs for sure.

3.1 Maximal Constraint Variants

To capture the refinement relationship among constraints, we consider the implication between operators [7]. For any two values a and b, if $a\phi_1 b$ always implies $a\phi_2 b$, it is said that $\phi_2 \in Imp(\phi_1)$. The $Imp(\phi)$ of all ϕ are presented in Table 1. For example, we have $\leq \in Imp(<)$ since any $a < b$ always implies $a \leq b$. As restrictions, $<$ is stronger than $\leq$. Given the denial semantics, $<$ refines the constraint with $\leq$, which belongs to $Imp(<)$. The concept of closure of a set of predicates, w.r.t. a denial constraint set, is defined and calculated upon the notation of $Imp(\phi)$ in [7], which is out the scope of this study.

Definition 3. *We call φ_2 a* refinement *of φ_1, denoted by $\varphi_1 \preceq \varphi_2$, if for each $P : x\phi_1 y \in pred(\varphi_1)$, there exists a $Q : x\phi_2 y \in pred(\varphi_2)$ such that $\phi_1 \in Imp(\phi_2)$.*

By inserting additional predicates, each variant φ' of φ is a refinement of φ.

Definition 4. *We call Σ_2 a* refinement *of Σ_1, denoted by $\Sigma_1 \preceq \Sigma_2$, if for each $\varphi_2 \in \Sigma_2$, there exists a $\varphi_1 \in \Sigma_1$ such that $\varphi_1 \preceq \varphi_2$.*

Intuitively, the more the predicates are inserted, the more likely the constraints overfit the data. In other words, the more the constraints are refined (via predicate insertion or substitution), the less the data need to be changed. For instance, two tuples with $t_\alpha.A = t_\beta.A$ violating $\neg(t_\alpha.A \leq t_\beta.A)$ will no longer violate the refinement $\neg(t_\alpha.A < t_\beta.A)$.

Lemma 1. *Given two constraint variants Σ_1, Σ_2 of Σ such that Σ_2 is also a refinement of Σ_1, having $\Sigma \preceq \Sigma_1 \preceq \Sigma_2$, it always has $\Delta(I, I_1) \geq \Delta(I, I_2)$, where I_1 and I_2 are the minimum data repairs w.r.t. Σ_1 and Σ_2, respectively.*

As a result, given possibly higher data repair cost $\Delta(I, I_1)$, the corresponding constraint variant Σ_1 can be ignored.

A variant Σ' of constraint set Σ is said *maximal* w.r.t. the constraint-variance bound θ, if there does not exist any Σ'' such that $\Sigma' \preceq \Sigma''$ and $\Theta(\Sigma, \Sigma'') \leq \theta$. Consequently, any non-maximal Σ' can be directly removed from $\mathbb{D}$.

Pruning Non-maximal Constraint Variants

We say that a variant φ' of constraint φ, $\varphi \preceq \varphi'$, is *maximal*, if there does not exist another variant φ'' such that $\varphi' \preceq \varphi''$ and $edit(\varphi, \varphi'') = edit(\varphi, \varphi')$.

It is obvious to see that any φ' must be maximal in a maximal constraint variant Σ'. (Otherwise, we could build a $\Sigma'', \Sigma' \preceq \Sigma''$, by $\varphi'', \varphi' \preceq \varphi''$, with the same constraint variation cost.) In other words, we only need to consider those maximal φ'_i when obtaining Φ_i for a $\varphi_i \in \Sigma$.

Proposition 2. *For any inserted predicate $P : x\phi y \in pred(\varphi') \setminus pred(\varphi)$, if $\phi \in \{\leq, \geq, \neq\}$, then φ' is not maximal.*

Instead of inserting all the possible predicates (indicated in Section 2.2), we only need to consider the predicates with operators in $\{<, >, =\}$ when obtaining the possible variants Φ_i of a φ_i. The others with $\leq, \geq, \neq$ can be ignored.

Example 5. *Consider φ_1 again in Example 1. Besides φ_2, we may have other variants for φ_1, e.g., by inserting predicates on* Income*.*

$$\begin{aligned}\varphi_5 :&\forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name}\\ &\wedge t_\alpha.\mathsf{Income} = t_\beta.\mathsf{Income} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})\\ \varphi_6 :&\forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathsf{Name} = t_\beta.\mathsf{Name}\\ &\wedge t_\alpha.\mathsf{Income} \leq t_\beta.\mathsf{Income} \wedge t_\alpha.\mathsf{CP} \neq t_\beta.\mathsf{CP})\end{aligned}$$

According to $\leq \in Imp(=)$ in Table 1 for the predicates on Income*, we have $\varphi_6 \preceq \varphi_5$. That is, φ_5 is a refinement of φ_6. According to Proposition 2, the constraint variant φ_6 by inserting a predicate with operator $\leq$ is not maximal.*

The minimum data repair w.r.t. φ_6 is exactly the results in Figure 1(b), with data repair cost 7 (by count). On the other hand, the minimum data repair cost w.r.t. φ_5 is 3 (the same as the results in Figure 1(c)). It verifies the conclusion in Lemma 1 that for $\varphi_6 \preceq \varphi_5$, the minimum data repair cost w.r.t. φ_6 is always no less than that of φ_5 (7>3). Consequently, we can directly ignore the constraint variant φ_6 without computing its data repair (with cost 7).

3.2 Variants with Bounded Data Repair Cost

This section focuses on pruning/removing the unnecessarily considered candidate constraint variants Σ' that would

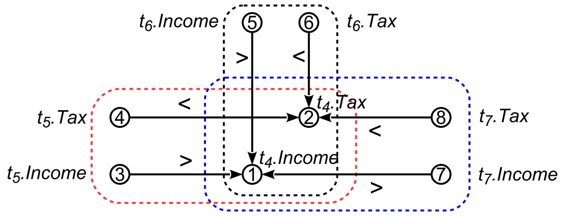

Figure 2: Conflict hypergraph

never generate the minimum data repair. We obtain $\delta_l(\Sigma', I)$ and $\delta_u(\Sigma', I)$, the lower and upper bounds of possible minimum data repairs of I w.r.t. Σ', respectively. Intuitively, for constraint variants Σ' and Σ'', if $\delta_u(\Sigma', I) < \delta_l(\Sigma'', I)$, then Σ'' can be directly discarded.

3.2.1 Conflict Graph for Representing Violations

Given a Σ over I, let us first capture all the violations, upon which the data repair cost bounds can be derived.

Definition 5. *The violation set $viol(I, \varphi) = \{\langle t_i, t_j, \dots\rangle \mid \langle t_i, t_j, \dots\rangle \nvDash \varphi, t_i, t_j, \dots \in I\}$ is a set of tuple lists that violate φ.*

The violation set of Σ is $viol(I, \Sigma) = \cup_{\varphi\in\Sigma} viol(I, \varphi)$.

We denote $cell(t_\alpha, t_\beta, \dots; \varphi)$ all the cells that are involved in the predicates $pred(\varphi)$ of φ.

$$\begin{aligned} cell(t_\alpha, t_\beta, \dots; \varphi) = &\{v_1 \mid P : v_1\phi c \in pred(\varphi)\}\cup \\ &\{v_1, v_2 \mid P : v_1\phi v_2 \in pred(\varphi)\}\end{aligned}$$

The degree $Deg(\varphi)$ of φ is defined as the number of distinct cells in φ, i.e., $Deg(\varphi) = |cell(t_\alpha, t_\beta, \dots; \varphi)|$.

Following [8], we represent violations in I w.r.t. Σ by a *conflict hypergraph* $\mathcal{G}$, where each vertex is a cell in I. Each violation tuple list $\langle t_i, t_j, \dots\rangle \in viol(I, \varphi)$ forms a hyperedge, consisting of $cell(t_i, t_j, \dots; \varphi)$, in the graph. The data repair problem is thus to find a I' such that all the conflict hyperedge are eliminated.

Example 6. *Consider again φ_4' in Example 4.*

$$\varphi_4' : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathit{Income} > t_\beta.\mathit{Income} \wedge t_\alpha.\mathit{Tax} < t_\beta.\mathit{Tax}).$$

For the relation I in Figure 1(a), the violation set is

$$viol(I, \varphi_4') = \{\langle t_5, t_4\rangle, \langle t_6, t_4\rangle, \langle t_7, t_4\rangle\}.$$

Figure 2 illustrates the conflict hypergraph built upon the violation set, where each node denotes a cell in the relation. Each hyperedge corresponds to a violation tuple list (pair). For instance, the hyperedge for $\langle t_5, t_4\rangle \in viol(I, \varphi_4')$ consists of $cell(t_5, t_4; \varphi_4') = \{t_5.\mathit{Income}, t_4.\mathit{Income}, t_5.\mathit{Tax}, t_4.\mathit{Tax}\}$.

3.2.2 Bounds on Minimum Data Repair Cost

To eliminate the violation of a conflict hyperedge, at least one vertex (cell) in the edge should be repaired. We denote

$$\min_{a\in dom(A)} dist(I(t.A), a)$$

the weight of vertex $t.A$, i.e., the minimum cost should be paid to repair $t.A$.

Let $\mathbb{V}^*(\mathcal{G})$ be the *minimum weighted vertex cover* of the hypergraph $\mathcal{G}$ corresponding to Σ, I, with weight

$$\|\mathbb{V}^*(\mathcal{G})\| = \sum_{t.A\in\mathbb{V}^*(\mathcal{G})} \min_{a\in dom(A)} dist(I(t.A), a).$$

Lemma 3. *For any valid repair I' of I, i.e., $I' \vDash \Sigma$, we have $\Delta(I, I') \geq \|\mathbb{V}^*(\mathcal{G})\|$.*

Since computing $\mathbb{V}^*$ is already known to be NP-hard, we consider an constant factor-f approximation of $\mathbb{V}^*$ [20], i.e., $\frac{\|\mathbb{V}(\mathcal{G})\|}{\|\mathbb{V}^*(\mathcal{G})\|} \leq f$, where f is the maximum degree of hyperedges and $\mathbb{V}(\mathcal{G})$ is an approximation of $\mathbb{V}^*(\mathcal{G})$.

Let $Deg(\Sigma) = \max_{\varphi\in\Sigma} Deg(\varphi)$ be the degree of Σ, i.e., the maximum degree of φ in Σ. It follows $\|\mathbb{V}(\mathcal{G})\| \leq f\|\mathbb{V}^*(\mathcal{G})\| \leq Deg(\Sigma)\Delta(I, I')$. Therefore, we define the lower bound of minimum data repair cost as

$$\delta_l(\Sigma, I) = \frac{\|\mathbb{V}(\mathcal{G})\|}{Deg(\Sigma)}.$$

Recall that the violation can be eliminated by simply assigning any cell in a hyperedge to fresh variable fv. Since $\mathbb{V}(\mathcal{G})$ covers all the hyperedges, assigning all the cells in $\mathbb{V}(\mathcal{G})$ forms a valid data repair. We thus define the upper bound of the minimum data repair cost as

$$\delta_u(\Sigma, I) = \sum_{t.A\in\mathbb{V}(\mathcal{G})} dist(I(t.A), fv).$$

Example 7 (Example 6 continued)**.** *Consider the conflict graph $\mathcal{G}$ in Figure 2. Suppose that the count cost is used, having $dist(a, b) = 1, a \neq b, a, b \in dom(A)$. That is, each vertex has weight 1. Let $AMWVG(\mathcal{G}) = \{t_4.\mathit{Tax}\}$ be an approximate minimum weighted vertex cover, with $\|AMWVG(\mathcal{G})\| = 1$. Referring to the number of 4 cells in φ_4', we have $Deg(\Sigma) = Deg(\varphi_4') = 4$. The lower bound of minimum data repair cost is $\delta_l(\Sigma, I) = 0.25$. Similarly, suppose that $dist(a, fv) = 1.1, a \in dom(A)$. The upper bound of minimum data repair cost can be computed as $\delta_u(\Sigma, I) = 1.1$.*

3.3 θ-tolerant Repair Algorithm

Consider a set $\mathbb{D}$ of constraint variation candidates Σ_i for Σ, whose variations are bounded by θ, $\Theta(\Sigma, \Sigma_i) \leq \theta$. Algorithm 1 returns a data repair $I_{\min} = \arg\min_{I_i} \Delta(I, I_i)$, where I_i is the minimum repair of I w.r.t. $\Sigma_i, i = 1, \dots, |\mathbb{D}|$.

First, it is notable that Σ itself, with $\Theta(\Sigma, \Sigma) = 0 \leq \theta$, should be a valid constraint "variation" candidate. Therefore, the upper bound $\delta_{\min}$ of the θ-tolerant minimum data repair can be initialized as $\delta_u(\Sigma, I)$ in Line 1.

For each variant Σ_i, if the lower bound $\delta_l(\Sigma_i, I)$ is no greater than the previously known upper bound $\delta_{\min}$ (Line 3), the data repair over Σ_i, I should be processed. (See the following section for the DATAREPAIR function.) Lines 5-7 update the bound $\delta_{\min}$, once a better result I_i is found.

Algorithm 1 θ-TOLERANTREPAIR($\mathbb{D}, \Sigma, I$)

```
Input: An instance $I$, a constraint set $\Sigma$, and a set $\mathbb{D}$ of con-
    straint variants with variation bounded by $\theta$
Output: A minimum data repair $I_{\min}$ w.r.t. $\mathbb{D}$
1: $\delta_{\min} := \delta_u(\Sigma, I)$
2: for each constraint variant $\Sigma_i \in \mathbb{D}$ do
3:   if $\delta_l(\Sigma_i, I) \leq \delta_{\min}$ then
4:     $I_i :=$ DATAREPAIR($\Sigma_i, I, \mathbb{V}(\mathcal{G}_i), \delta_{\min}$)
5:     if $\Delta(I, I_i) < \delta_{\min}$ then
6:       $\delta_{\min} := \Delta(I, I_i)$
7:       $I_{\min} := I_i$
8: return $I_{\min}$
```

Example 8. *Consider the relation in Figure 1(a) and $\Sigma = \{\varphi_4\}$ in Example 3. For a $\theta = \frac{1}{2}$, suppose that the (pruned)* *constraint variants are $\mathbb{D} = \{\Sigma_1, \Sigma_2\}$ where $\Sigma_1 = \{\varphi'_4\}$ in Example 4, and $\Sigma_2 = \{\varphi''_4\}$ contains*

$$\varphi''_4 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.\mathit{Income} > t_\beta.\mathit{Income} \wedge t_\alpha.\mathit{Tax} = t_\beta.\mathit{Tax}).$$

Note that both Σ_1 and Σ_2 are maximal w.r.t. θ.

As shown in Example 6, for Σ_1, its conflict hypergraph $\mathcal{G}_1$ (in Figure 2) can be obtained. We have $\delta_u(\Sigma_1, I) = 1.1$, referring to Example 7.

Similarly, for Σ_2, we obtain conflict hypergraph $\mathcal{G}_2$ with $\mathbb{V}(\mathcal{G}_2) = \{t_2.\mathit{Tax}, t_3.\mathit{Tax}, t_5.\mathit{Tax}, t_6.\mathit{Tax}, t_7.\mathit{Tax}\}$. Referring to the count cost as introduced in Example 7 and $Deg(\Sigma_2) = Deg(\varphi'_4) = 4$, it follows $\delta_l(\Sigma_2, I) = 5/4 = 1.25$.

With $\delta_u(\Sigma_1, I) < \delta_l(\Sigma_2, I)$, the constraint variant Σ_2 can be further pruned without calling DataRepair.

Let ℓ be the maximum number of tuples $|\{t_\alpha, t_\beta, \dots\}|$ involved in a φ of Σ. The construction of $\mathcal{G}_i$ for each $\Sigma_i \in \mathbb{D}$ costs $O(|I|^\ell)$. As illustrated in the following section, for a certain set of changing cells $\mathbb{V}(\mathcal{G}_i)$, the DataRepair algorithm runs in $O(|I|^\ell)$ time. Thereby, the θ-TolerantRepair algorithm runs in $O(|I|^\ell|\mathbb{D}|)$ time. Typical constraints are with $\ell = 1$ or 2, i.e., involving one or two tuples. For instance, linear denial constraints [16, 1] and constant conditional functional dependencies [3] have $\ell = 1$, declaring constraints on single tuples. Functional dependencies and general conditional functional dependencies, with $\ell = 2$, consider constraints involving two tuples. The proposed pruning practically reduces computing time costs but does not improve the complexity.

4. DATA REPAIR

Next, given a set Σ' of constraints (variants), we focus on generating the minimum data repair of I w.r.t. Σ'. In particular, it is expected to share the computation among different constraint variants Σ'.

Existing holistic approach [8] is incapable of the aforesaid sharing, owing to multiple ad-hoc repair rounds. The rationale is that putting only violations into repair context may introduce new violations to others after one round repairing.

In this section, to advance data repair with sharing, we introduce a novel violation-free repair (in Section 4.1) that ensures no new violations introduced after repairing. The idea is to consider in repair context not only the violations but also the suspects that may potentially introduce violations. This one round data repair thus enables sharing among various constraint variants (in Section 4.2).

4.1 Violation Free Data Repair

Since the problem of finding the minimum data repair has already been known to be NP-hard [16], we thereby focus on approximation approaches for repairing. Following the same line of [8], the repairing starts from heuristically selected cells, i.e., the aforesaid $\mathbb{V}(\mathcal{G})$, denoted by $\mathbb{C}$ for simplicity. To perform the one-round violation-free repair, it is essential to guarantee that the modification on cells in $\mathbb{C}$ will not introduce new violations (conflict hyperedges).

4.1.1 Suspects of Potential Violations

Let us first capture the "suspects" that may lead to new violations after modifying the cells in $\mathbb{C}$. Recall that a violation occurs in a tuple list $t_i, t_j, \dots$ w.r.t. φ, when all the predicates in φ are satisfied. Intuitively, if a predicate of φ is not satisfied, which does not contain cells from $\mathbb{C}$, the repairing on $\mathbb{C}$ will not affect $\langle t_i, t_j, \dots\rangle \vDash \varphi$. That is, $\langle t_i, t_j, \dots\rangle$ does not need to be suspected.

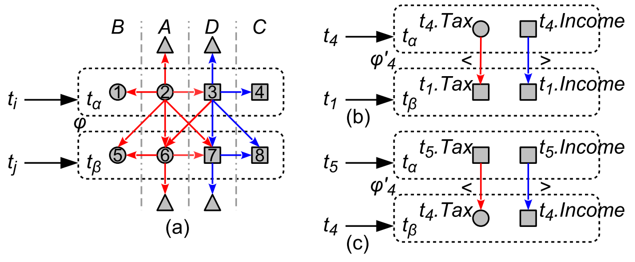

Figure 3: Suspect condition (blue arrow) and repair context (red arrow with operator inverse)

Definition 6. *The* suspect set *$susp(\mathbb{C}, \varphi)$ of a φ is a set of tuple lists $\langle t_i, t_j, \dots\rangle$ satisfying all the predicates in φ which do not involve cells in $\mathbb{C}$.*

All the tuple lists $\langle t_i, t_j, \dots\rangle$ satisfying the following suspect condition $sc(t_i, t_j, \dots; \varphi)$ w.r.t. $\mathbb{C}$ should be suspected,

$$sc(t_\alpha, t_\beta, \dots; \varphi) = \{I(v_1)\phi c \mid P : v_1\phi c \in \mathit{pred}(\varphi), v_1 \notin \mathbb{C}\} \cup$$
$$\{I(v_1)\phi I(v_2) \mid P : v_1\phi v_2 \in \mathit{pred}(\varphi), v_1, v_2 \notin \mathbb{C}\}$$

which considers all the predicates on cells in $\mathit{cell}(t_\alpha, t_\beta, \dots; \varphi)$ except those ones from $\mathbb{C}$.

To understand the suspect condition more clearly, let us illustrate an example constraint φ declared on two tuples t_i (as t_α of φ) and t_j (as t_β of φ) in Figure 3(a). Each circle denotes a changing cell from $\mathbb{C}$, while a square is a cell not in $\mathbb{C}$ (that will not be modified). The constant associated in a predicate is represented by a triangle. Each arrow $v_1 \to v_2$ denotes a predicate declared on two cells $v_1\phi v_2$ (or between a cell and a constant $v_1\phi c$). The suspect condition considers predicates with cells not from $\mathbb{C}$, i.e., blue arrows attached on two squares or between squares and triangles. For instance, a suspect condition $I(t_j.D)\phi I(t_j.C)$, i.e., 7→8, corresponds to the satisfaction of predicate $t_\beta.D\phi t_\beta.C \in \mathit{pred}(\varphi)$. Consequently, if all the blue edges specified by predicates in φ are satisfied, i.e., $sc(t_i, t_j; \varphi)$ is satisfied, $\langle t_i, t_j\rangle$ becomes a suspect tuple list (pair).

Lemma 4. *For any $\mathbb{C}$, it always has $viol(I, \varphi) \subseteq susp(\mathbb{C}, \varphi)$.*

Lemma 4 states that any tuple list already in violation is naturally suspected. In other words, identifying the suspect set is sufficient to capture all the violations.

Example 9. *Consider again φ'_4 in Example 4 over the relation in Figure 1(a). Let $\mathbb{C} = \{t_4.\mathsf{Tax}\}$ be the set of changing cells, i.e., an $\mathbb{V}(\mathcal{G})$ for the conflict hypergraph $\mathcal{G}$ in Figure 2. We have the suspect set $susp(\mathbb{C}, \varphi'_4) = \{\langle t_4, t_1\rangle, \langle t_4, t_2\rangle, \langle t_4, t_3\rangle, \langle t_5, t_4\rangle, \langle t_6, t_4\rangle, \langle t_7, t_4\rangle, \langle t_8, t_4\rangle, \langle t_9, t_4\rangle, \langle t_{10}, t_4\rangle\}$. For instance, consider $\langle t_4, t_1\rangle$ in Figure 3(b), where $t_4.\mathsf{Tax} \in \mathbb{C}$ is denoted by a circle, while other cells not in $\mathbb{C}$ are squares. The predicate $t_4.\mathsf{Tax} < t_1.\mathsf{Tax}$ on $t_4.\mathsf{Tax} \in \mathbb{C}$ is a red arrow, while the blue arrow denotes the other $t_4.\mathsf{Income} > t_1.\mathsf{Income}$ whose cells are not in $\mathbb{C}$.*

The suspect condition on $\langle t_4, t_1\rangle$ is thus $sc(t_4, t_1, \varphi'_4) = \{I(t_4.\mathsf{Income}) > I(t_1.\mathsf{Income})\}$. Referring to Figure 1(a), we have $I(t_4.\mathsf{Income}) = 3k > I(t_1.\mathsf{Income}) = 0$. That is, $\langle t_4, t_1\rangle$ satisfies all the predicates (blue arrow) in φ'_4 except *those (red arrow) with cells from $\mathbb{C}$. Referring to Definition 6, $\langle t_4, t_1\rangle$ is a suspect tuple pair. It is possible that repairs on $t_4.\mathsf{Tax}$ may introduce violations to $t_1.\mathsf{Tax}$, i.e., when $I'(t_4.\mathsf{Tax}) < I(t_1.\mathsf{Tax})$ with all the predicates in φ'_4 satisfied.*

For the violation set $viol(I, \varphi'_4) = \{\langle t_5, t_4\rangle, \langle t_6, t_4\rangle, \langle t_7, t_4\rangle\}$ in Example 6, we have $viol(I, \varphi'_4) \subseteq susp(\mathbb{C}, \varphi'_4)$ by Lemma 4.

4.1.2 *Repair Context over Suspects*

For each $\langle t_i, t_j, \dots\rangle$ in the suspect set of φ, the following repair context $rc(t_i, t_j, \dots; \varphi)$ ensures that the repaired cells in $\mathbb{C}$ will not satisfy the predicates $P : x\phi y$ declared on $\mathbb{C}$, by specifying the inverse $\overline{\phi}$ of ϕ.

$$rc(t_\alpha, t_\beta, \dots; \varphi) = \{I'(v_1)\overline{\phi}c \mid P : v_1\phi c \in pred(\varphi), v_1 \in \mathbb{C}\} \cup$$
$$\{I'(v_1)\overline{\phi}I'(v_2) \mid P : v_1\phi v_2 \in pred(\varphi), v_1, v_2 \in \mathbb{C}\} \cup$$
$$\{I'(v_1)\overline{\phi}I(v_2) \mid P : v_1\phi v_2 \in pred(\varphi), v_1 \in \mathbb{C}, v_2 \notin \mathbb{C}\} \cup$$
$$\{I(v_1)\overline{\phi}I'(v_2) \mid P : v_1\phi v_2 \in pred(\varphi), v_1 \notin \mathbb{C}, v_2 \in \mathbb{C}\}$$

Each repair context predicate in $rc(t_\alpha, t_\beta, \dots; \varphi)$ corresponds to the inverse of a predicate in φ with cells from $\mathbb{C}$ (red arrow in Figure 3). For instance, $I(t_\alpha.D)\overline{\phi}I'(t_\beta.A)$, corresponding to 3→6, ensures that the predicate $t_\alpha.D\phi t_\beta.A \in pred(\varphi)$ will not be satisfied after modifying $t_\beta.A \in \mathbb{C}$.

We assemble the repair contexts for each suspect tuple list $\langle t_i, t_j, \dots\rangle$ of all $\varphi \in \Sigma$ to generate a data repair I'.

Proposition 5. *Any assignment that satisfies all the repair contexts forms a valid repair I' without introducing any new violations, i.e., $I' \models \Sigma$.*

Again, the repair cost is expected to be small.

$$\min \sum_{t_i.A\in\mathbb{C}} dist(I(t_i.A), I'(t_i.A)) \qquad (3)$$
$$\text{s.t.} \quad rc(t_i, t_j, \dots; \varphi) \qquad \langle t_i, t_j, \dots\rangle \in susp(\mathbb{C}, \varphi), \varphi \in \Sigma$$

Existing solver can be employed to efficiently compute solutions, e.g., liner programming (LP) for numerical values and value frequency map (VFM) for string types [8].

Example 10 (Example 9 continued)**.** *For the suspect tuple list $\langle t_4, t_1\rangle \in susp(\mathbb{C}, \varphi'_4)$, its repair context $rc(t_4, t_1; \varphi'_4) = \{I'(t_4.\mathsf{Tax}) \geq I(t_1.\mathsf{Tax})\}$ is obtained by considering the inverse of each predicate in φ'_4 with cells from $\mathbb{C}$, i.e., red arrow in Figure 3(b). Similarly, $rc(t_5, t_4; \varphi'_4) = \{I'(t_5.\mathsf{Tax}) \geq I(t_4.\mathsf{Tax})\}$ is obtain from the red arrow in Figure 3(c).*

By considering all the tuple pairs in $susp(\mathbb{C}, \varphi'_4)$, we obtain the entire repair contexts in formula (3), as shown in Figure 4(a). Given the repair contexts $I'(t_4.\mathsf{Tax}) \geq I(t_1.\mathsf{Tax}) = 0$ and $I'(t_4.\mathsf{Tax}) \leq I(t_5.\mathsf{Tax}) = 0$, a solution to the problem in formula (3) is $I'(t_4.\mathsf{Tax}) = 0$ (as shown in Example 4).

4.1.3 *Problem Solving with Fresh Variable*

As mentioned, there may not exist any assignment from the currently known values that can satisfy all the repair contexts. Following the same line of DC-based data repairing [8], we attempt to assign some cells by fresh variable *fv*. Recall that *fv* (denoting values out of the current domain such as unknown or not applicable) does not satisfy any predicate including the repair context predicates. Once a cell is assigned by *fv*, all the repair context predicates involving the cell can be removed. Heuristically, we would select a cell $t.A \in \mathbb{C}$ with the largest number of appearance in the repair contexts, to assign *fv*.

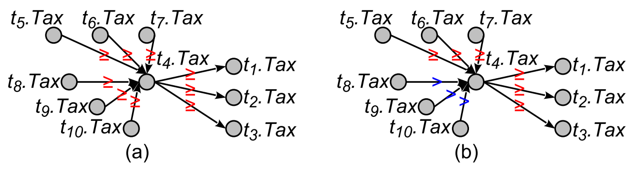

Figure 4: Repair contexts (a) rc_1 **and (b)** rc_2

The repairing keeps on assigning *fv*, i.e., removing repair context predicates, till the problem with the remaining repair contexts can be solved. The more the repair context predicates are removed, the more likely the problem can be solved. The worst case is the assignment of *fv* for all cells in $\mathbb{C}$ (see more discussions below on performance analysis).

Rather than gradually (slowly) assigning *fv* in each iteration, we are indeed able to identify ahead some cells $t.A$ that must be assigned to *fv*. Let

$$rc(t.A, \Sigma) = \{$$
$$I(v_1)\overline{\phi_1}I'(t.A), I'(t.A)\overline{\phi_2}I(v_2), I'(t.A)\overline{\phi_3}c \in rc(t_i, t_j, \dots; \varphi)$$
$$\mid t \in \{t_i, t_j, \dots\}, v_1, v_2 \notin \mathbb{C}, \langle t_i, t_j, \dots\rangle \in susp(\mathbb{C}, \varphi), \varphi \in \Sigma\}$$

denote all the repair context predicates declared between $t.A$ and a constant c, or between $t.A$ and cell $v_1, v_2 \notin \mathbb{C}$ that cannot be modified. If $rc(t.A, \Sigma)$ is unsatisfiable, we directly assign $I'(t.A) = fv$.

Example 11. *Consider again φ_4 in Example 3, with $\mathbb{C} = \mathbb{V}(\mathcal{G}) = \{t_2.\mathsf{Tax}, t_3.\mathsf{Tax}, t_5.\mathsf{Tax}, t_6.\mathsf{Tax}, t_7.\mathsf{Tax}\}$ as illustrated in Example 8. Following the same line of Example 10, we compute all the repair contexts for each suspect tuple pair in $susp(\mathbb{C}, \varphi_4)$. Now consider the subset of repair contexts on the cell $t_2.\mathsf{Tax}$, i.e., $rc(t_2.\mathsf{Tax}, \{\varphi_4\}) = \{I'(t_2.\mathsf{Tax}) > I(t_1.\mathsf{Tax}), I(t_4.\mathsf{Tax}) > I'(t_2.\mathsf{Tax}), I(t_8.\mathsf{Tax}) > I'(t_2.\mathsf{Tax}), I(t_9.\mathsf{Tax}) > I'(t_2.\mathsf{Tax}), I(t_{10}.\mathsf{Tax}) > I'(t_2.\mathsf{Tax})\}$. Given that $I(t_1.\mathsf{Tax}) = 0$ and $I(t_4.\mathsf{Tax}) = 3k$, $rc(t_2.\mathsf{Tax}, \{\varphi_4\})$ indeed requires $0 < I'(t_2.\mathsf{Tax}) < 3k$. As presented in Example 3, since there is not value in the current $dom(\mathsf{Tax})$ that meets the requirement, we can directly assign $I'(t_2.\mathsf{Tax}) = fv_1$ by fresh variable, without solving the problem in formula (3) over the entire repair contexts.*

4.2 Enabling Sharing among Problem Solving

To enable sharing, we decompose $\mathbb{C}$ into a set of components $\mathbb{C}_1, \dots, \mathbb{C}_m$, such that for any $v_1 \in \mathbb{C}_i, v_2 \in \mathbb{C}_j$, there does not exist $I'(v_1)\overline{\phi}I'(v_2) \in rc(t_i, t_j, \dots; \varphi)$, for some tuple list $\langle t_i, t_j, \dots\rangle \in susp(\mathbb{C}, \varphi), \varphi \in \Sigma$. That is, the assignment of $v_1 \in \mathbb{C}_i$ will not affect $v_2 \in \mathbb{C}_j$. Let

$$rc(\mathbb{C}, \Sigma) = \bigcup_{\langle t_i, t_j, \dots\rangle \in susp(\mathbb{C}, \varphi), \varphi\in\Sigma} rc(t_i, t_j, \dots; \varphi)$$

denote all the repair contexts in formula (3) for short. Since components are disjoint w.r.t. repair contexts, $rc(\mathbb{C}_k, \Sigma)$ is indeed a projection of $rc(\mathbb{C}, \Sigma)$ on $\mathbb{C}_k$, having $rc(\mathbb{C}, \Sigma) = rc(\mathbb{C}_1, \Sigma) \cup \dots \cup rc(\mathbb{C}_m, \Sigma)$. Consequently, we can solve the problem by resolving the subproblems w.r.t. $rc(\mathbb{C}_k, \Sigma)$ for each component $\mathbb{C}_k$ individually.

Decomposing into components not only accelerates the solution computing but also enhances the possibility of result sharing among (sub)problems. Obviously, the solution can be directly shared, if $rc(\mathbb{C}_k, \Sigma_1) = rc(\mathbb{C}_k, \Sigma_2)$.

To further explore the sharing opportunity, we consider the following refinement relationship among repair contexts.

Definition 7. *We say that $rc(\mathbb{C}_k, \Sigma_2)$ refines $rc(\mathbb{C}_k, \Sigma_1)$, denoted by $rc(\mathbb{C}_k, \Sigma_2) \sqsubseteq rc(\mathbb{C}_k, \Sigma_1)$, if for each $x\overline{\phi_1}y \in rc(\mathbb{C}_k, \Sigma_1)$, there exists $x\overline{\phi_2}y \in rc(\mathbb{C}_k, \Sigma_2)$ s.t. $\overline{\phi_1} \in Imp(\overline{\phi_2})$.*

That is, the repair contexts in $rc(\mathbb{C}_k, \Sigma_2)$ are more strict (refined). For instance, rc_2 (with $t_8.\mathsf{Tax} > t_4.\mathsf{Tax}$) in Figure 4(b) refines rc_1 (with $t_8.\mathsf{Tax} \geq t_4.\mathsf{Tax}$) in Figure 4(a).

Proposition 6. *For an optimal solution $I'(\mathbb{C}_k)$ of $rc(\mathbb{C}_k, \Sigma_1)$, if $I'(\mathbb{C}_k)$ also satisfies $rc(\mathbb{C}_k, \Sigma_2)$ and $rc(\mathbb{C}_k, \Sigma_2) \sqsubseteq rc(\mathbb{C}_k, \Sigma_1)$, then $I'(\mathbb{C}_k)$ is an optimal solution to $rc(\mathbb{C}_k, \Sigma_2)$ w.r.t. the problem in formula (3).*

A natural idea is to materialize the optimal solutions of the previously solved $rc(\mathbb{C}_k, \Sigma_1)$, and reuse them directly when solving $rc(\mathbb{C}_k, \Sigma_2)$. To search the materialized $rc(\mathbb{C}_k, \Sigma_1)$ for $rc(\mathbb{C}_k, \Sigma_2)$, efficient filtering can be applied to fast exclude the repair contexts on distinct $\mathbb{C}_k$.

Example 12. *Let $rc_1 = rc(\{t_4.\mathsf{Tax}\}, \{\varphi'_4\})$ denote all the repair contexts in Figure 4(a) in Example 10. Suppose that there is another rc_2 of repair contexts on the same $\mathbb{C} = \{t_4.\mathsf{Tax}\}$ as shown in Figure 4(b). Note that we have $I(t_8.\mathsf{Tax}) > I'(t_4.\mathsf{Tax}) \in rc_2$, whereas $I(t_8.\mathsf{Tax}) \geq I'(t_4.\mathsf{Tax}) \in rc_1$. It is similar for $t_9.\mathsf{Tax}$ and $t_{10}.\mathsf{Tax}$. Referring to $\geq \in Imp(>)$ in Table 1, we have $rc_2 \sqsubseteq rc_1$. Since the optimal solution $I'(t_4.\mathsf{Tax}) = 0$ for rc_1 also satisfies all the repair contexts in rc_2, according to Proposition 6, we can directly conclude that $I'(t_4.\mathsf{Tax}) = 0$ is an optimal solution for rc_2 as well.*

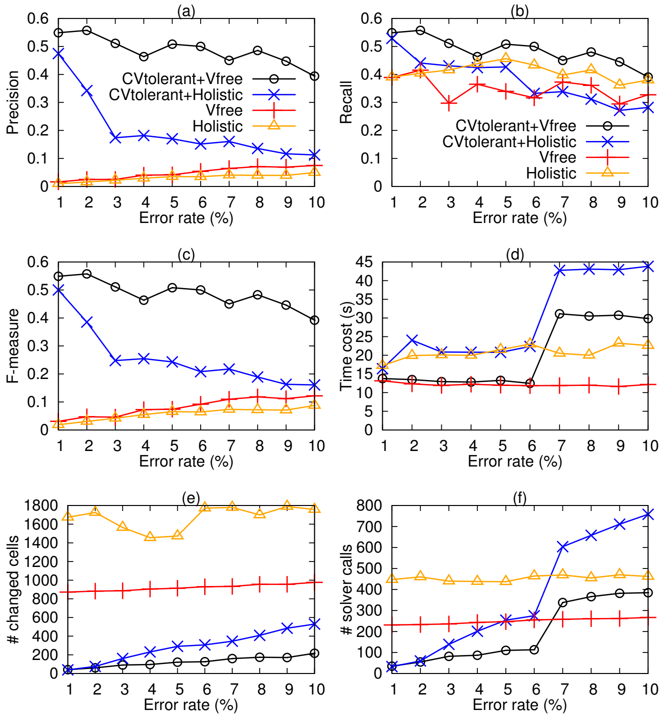

Figure 5: Vfree vs. Holistic data repairing with and without constraint variance tolerance (HOSP)

5. EXPERIMENTAL STUDY

In this section, we compared our proposed θ-tolerant repair with the state-of-the-art approaches (with/without constraint repairs). By default, we consider the unit predicate cost with $c(P) = 1$ and $\lambda = -0.5$ in formula (1). Details on experiment preparation can be found in Appendix D.1.

5.1 Evaluating Proposed Techniques

First, we compare our violation-free (Vfree) data repair algorithm, in Section 4, with the existing Holistic data repair [8]. As DC-based data repair algorithms, both Vfree and Holistic can work together with our proposed constraint-variance tolerant repair CVtolerant, in Section 3.

Figure 5 reports the performance under various error rates (with $\theta = 1$ for CVtolerant, see discussion below on choosing θ). Generally, our proposed Vfree shows higher accuracy in Figure 5(c) than Holistic, especially when working with CVtolerant. The results verify the effectiveness of considering suspects rather than only violations in the repair context.

Time cost of Vfree is significantly lower than that of Holistic, either with or without CVtolerant, as shown in Figure 5(d). The rationale is that our Vfree needs only one round violation-free repair, while Holistic may have multiple rounds for handling newly introduced violations in each round.

The CVtolerant+Vfree approach can always achieve the highest accuracy in Figures 5(a)-(c). The time costs of CVtolerant increase heavily from error rate 0.6 to 0.7. It is because the number of calling DataRepair (in Line 6 of Algorithm 1) increases from 1 to 2. The corresponding number of calling solvers (in Line 9 of Algorithm 2) increases largely as well, as illustrated in Figure 5(f). Nevertheless, owing to our efficient pruning and sharing techniques, most tests can obtain the results by calling DataRepair within 2 times, and the times of calling solver for CVtolerant are comparable to that of without constraint variance.

An interesting result is the slight increase of accuracy with the increase of error rate, by approaches without CVtolerant. The reason is the large number of changed cells by Vfree and Holistic, as shown in Figure 5(e), most of which are indeed not noises and negatively repaired by the inaccurate constraints. With the increase of error rate, some of them become true noises and thus the repair accuracy increases a bit. When more accurate constraints are captured via variance toleration, CVtolerant (either with Vfree or Holistic) shows accuracy decrease with the increase of error rate.

Figure 6 studies the impact of varying the constraint variance tolerance level θ (under an error rate of 7%). With the increase of θ, i.e., with higher tolerance of constraint variances, many negative repairs can be avoided, and the repairing accuracy (precision as well as f-measure) increases. However, when θ increases to 3, no data need to be changed, i.e., the constraints are over-refined and overfit the data.

Observing the number of repaired cells may help in determining θ. A large number of changed cells (under a small θ) indicate that data is overrepaired (with imprecise constraints). In contrast, an extremely small size of repaired cells (under a large θ) means that the constraints are over-refined (overfitting the data without identifying violations).

Similar results are observed over CENSUS, in Figures 7 and 8 analogous to Figures 5 and 6 over HOSP. Specifically, as shown in Figure 7(b), our CVtolerant methods show significantly higher accuracy than those without constraint-variance tolerance. Again, the time cost of Vfree is significantly lower than that of Holistic, either with or without

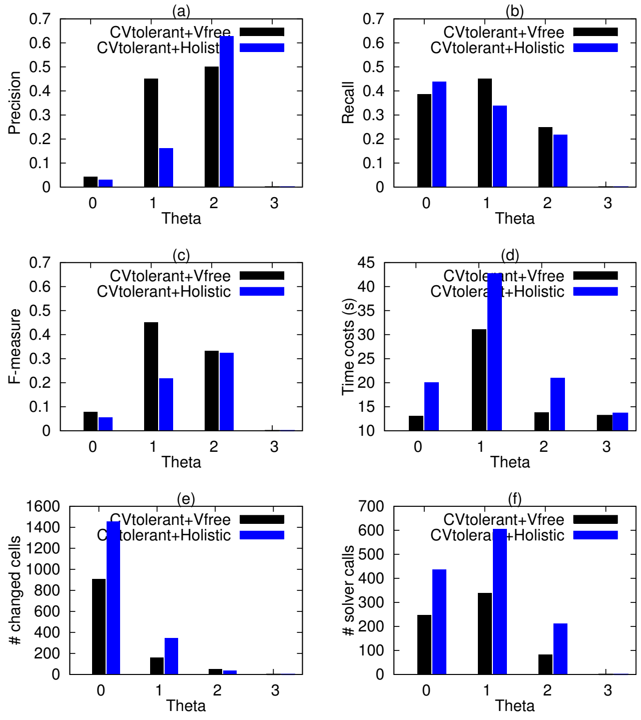

Figure 6: Varying tolerance level θ (HOSP)

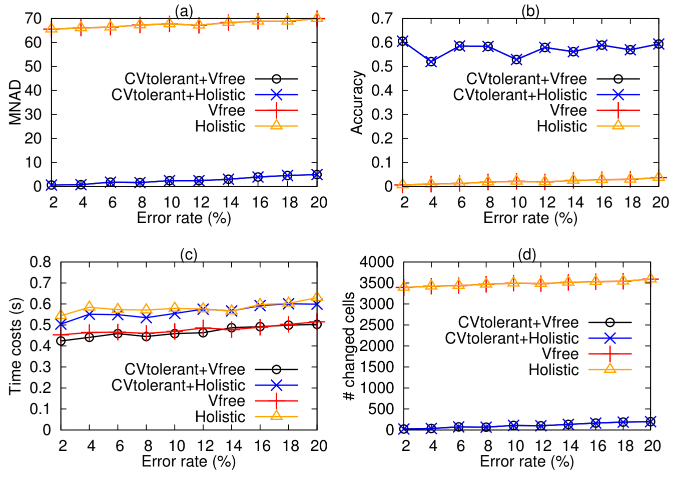

Figure 7: Comparison with and without constraint variance tolerance (CENSUS)

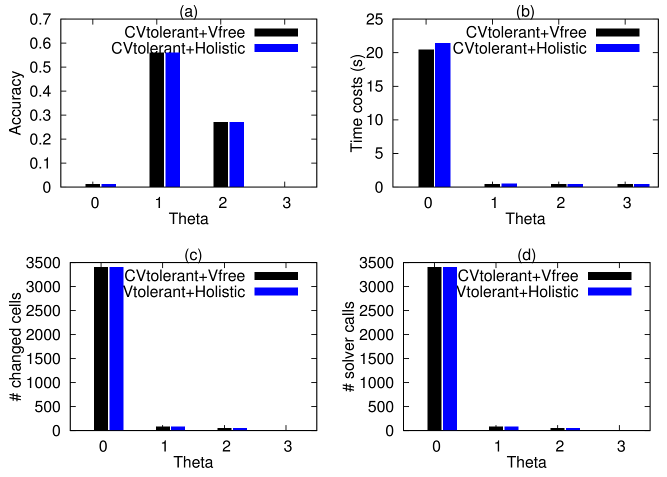

Figure 8: Varying tolerance level θ (CENSUS)

CVtolerant, as shown in Figure 7(d). The accuracy results of Vfree and Holistic are similar, because the constraints employed in this test are liner DCs where the repairing w.r.t. suspect set is similar to that on violation set. Similar to Figure 6(e) over HOSP, Figure 8(c) illustrates that the larger the constraint variance tolerance level θ is, the fewer the number of cells need to be changed. With a moderate $\theta = 1$, i.e., neither oversimplified constraints leading to many changed cells nor overrefined constraints without repairing any data, the accuracy is highest in Figure 8(a), which is also observed in Figure 6(c) over HOSP.

5.2 Comparison to FD-based Repairs

Next, we compared the state-of-the-art approaches using FDs as constraints, over the categorized data HOSP, including (1) Vrepair [14] and Holistic [8] without considering constraint variances, (2) Unified [5] with a unified cost model for both data and constraint repairs, (3) Relative [2] considering the relative trust between data and constraints via a data repair cost threshold, and (4) our proposed CVtolerant.

Figures 9 and 10 report the performance on various error rates and data sizes. In general, our CVtolerant approach shows significantly higher repair accuracy than all the other approaches. The time cost increases almost linearly in the number of tuples, in Figure 10(b). The results verify the complexity analysis (in Sections 3 and 4) of the proposed θ-tolerant repair approach.

The number of changed cells by Unified decreases sharply from 6 to 7 in Figure 11(b). It is because the cost of data repair exceeds that of constraint repair in the unified cost model. Constraint repair is performed rather than data repair, i.e., the number of changed cells drops. Its corresponding accuracy varies significantly as well, from 7 to 8, in Figure 10(a). The results verify again that constraints and data are not equally trustworthy for repairing, as indicated in [2].

The Relative method takes extremely high time costs, as shown in Figure 10(b), which is also reported in [2]. The method uses a fixed threshold τ of data repair cost to filter constraint repairs, rather than the dynamically determined minimum data repair cost bound $\delta_{\min}$ in our proposed algorithms. A large number of constraint and data repairs need to be considered by Relative with very high overhead. Owing to the extremely high time costs, we stop attempting the method with data sizes greater than 1000 in Figure 10 (given the clearly lower accuracy under various error rates, as well as on various # constraints and # attributes below).

In Figures 9(a) and 10(a), CVtolerant with weighted cost shows a bit higher accuracy than CVtolerant with unit cost, and significantly better results compared to the existing Unified and Relative methods. The results verify again that CVtolerant with either weighted or unit cost can improve the state of the art in further selecting the right attribute (predicate) for constraint variation. The improvement of weighted cost compared to unit cost in CVtolerant is not very significant. The reason is that inserting inappropriate predicates,

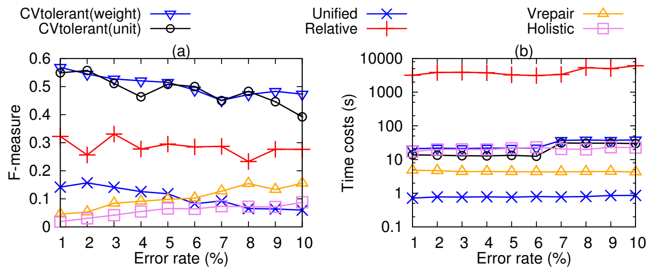

Figure 9: Comparison under FD constraints with various data error rates (HOSP)

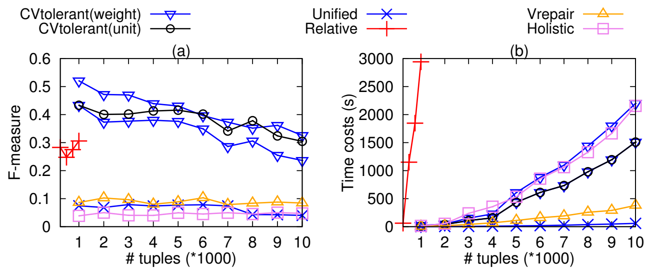

Figure 10: Scalability on number of tuples (HOSP)

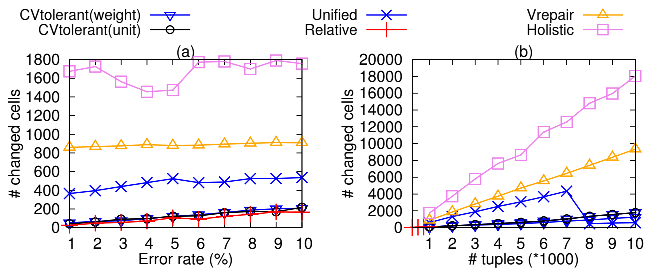

Figure 11: Changed cells (HOSP)

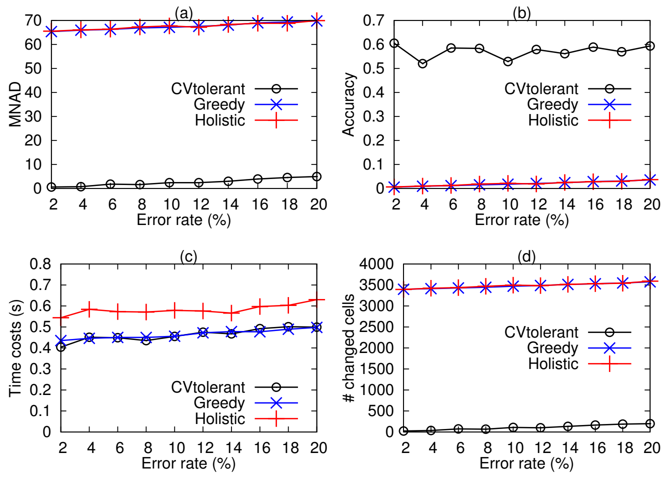

Figure 12: Comparison under DC constraints with various data error rates (CENSUS)

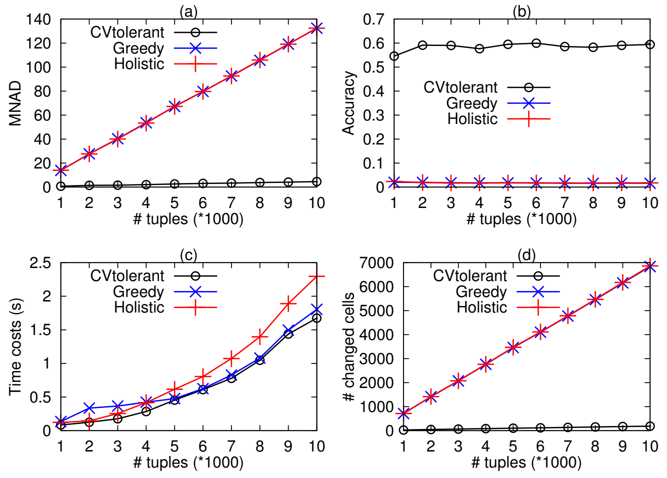

Figure 13: Scalability on num. of tuples (CENSUS)

e.g., on attribute Year with worse coincided distribution, does not help much in refining oversimplified constraints, and thus the data repair cost is high. Instead, inserting the right predicates, e.g., on attribute Birthday with better coincided distribution, does refine the oversimplified constraints, and the corresponding data repair with lower cost will be returned following the minimum change principle. That is, CVtolerant with unit cost on predicates achieves considerably good accuracy as well. Nevertheless, by introducing fine-grain cost of predicates, the right attributes and predicates can be more precisely selected and the repair accuracy improves as reported in Figures 9 and 10.

5.3 Comparison to DC-based Repairs

This experiment over CENSUS considers numerical data repairing by DC-based approaches: (1) Greedy [16] using a modified greedy strategy, (2) Holistic [8] putting violations into repair context, and (3) our proposed CVtolerant.

As shown in Figures 12(a) and 13(a), the MNAD distance (between truth and repair) of our CVtolerant is significantly lower than those of Greedy and Holistic, i.e., more close to the ground truth. Similarly, for the accuracy relative to the introduced noises, in Figures 12(b) and 13(b), CVtolerant shows clear improvement as well. Again, the rationale is that without repairing inaccurate constraints, existing approaches change a large number of cells (that are indeed correct) as illustrated in Figure 13(d). Time costs of CVtolerant are only a slightly higher than existing methods. Indeed, as illustrated in Figure 13(c), all the approaches scale well.

5.4 Evaluation over Various Errors

Real errors could be correlated, that is, errors often appear together in the same tuples on a certain set of attributes, rather than independently appear in random tuple-attribute cells. Such error patterns could be prevalent in real-world. For instance, tuples collected from unreliable sources could involve multiple errors, e.g., not only the Name value contains error but also Birthday.

Figure 14 presents an experiment on various correlation levels of errors, by observing different numbers of errors that appear together in the same tuples with correlations. As shown in Figure 14(a), the accuracy drops a bit with the increase of errors in a tuple (higher error appearance correlation). Nevertheless, the proposed method CVtolerant still clearly outperforms other methods, which is similar to the previous results in Figure 9(a).

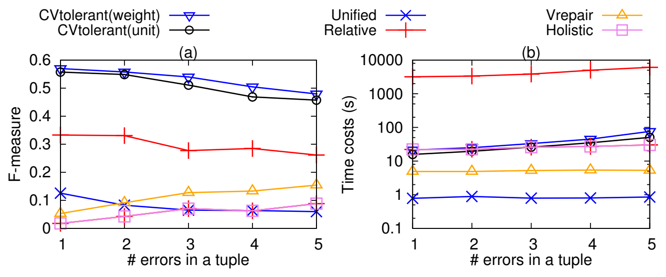

Figure 14: Comparison over correlated errors appearing together (HOSP)

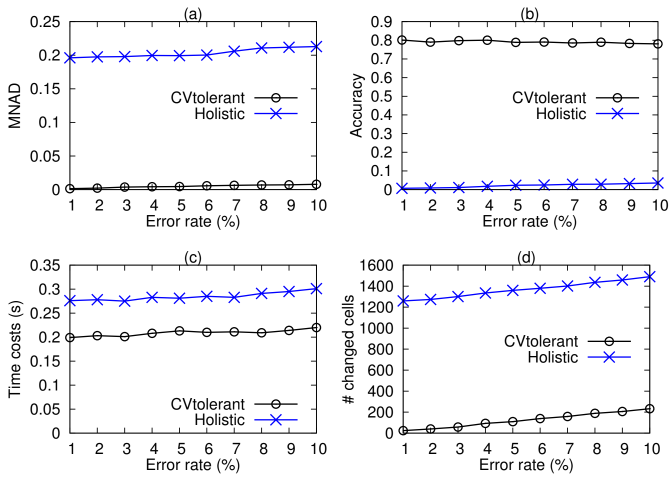

Figure 15: Comparison over real GPS data with errors naturally embedded

The GPS dataset has real errors naturally embedded and the ground truth values (locations) of errors manually labelled. Specifically, we collect GPS readings carrying a smartphone and walking along main roads at campus. The GPS readings are not accurate and may occasionally "jump" away from the trajectory. Since we know exactly the path of walking, a number of 243 dirty points are manually identified (among total 2409 points in the trajectory). The true locations of these dirty points are also manually labeled, as the ground truth. Figure 15 reports the results over the GPS data. As shown, e.g., in Figure 15(b), our proposed method CVtolerant achieves significantly higher accuracy than that of Holistic, which is similar to Figure 12 over CENSUS.

6. RELATED WORK

Integrity Constraints. In this paper, we use a general notation of integrity constraints, denial constraints (DCs) [7]. The reason is that DCs are able to express the semantics of several classes of integrity constraints, e.g., traditional key constraints and functional dependencies (FDs). Since constants can be declared in DCs, the conditional functional dependencies (CFDs) [3] can also be expressed with constants as conditions. The advantage is obvious to employ multiple classes of constraints, which can identify more violations.

Besides manual specification, DCs are often discovered from data [7]. The discovery focuses on finding a minimal cover of DCs that hold in the given training data. Instead, our constraint variance considers a set of maximal DCs with variation to the given DCs bounded by θ. The major difference is that the discovery concerns a concise representation of precise constraints from (usually clean) data, while the repair explores the variants of inaccurate constraints for the minimum repair of dirty data.

Data Repair. To eliminate violations to integrity constraints, the deletion model [6] manages to delete a minimum set of tuples such that the remaining data can satisfy the constraints. Information loss occurs in such deletion.

Rather than deletion, the data modification repair model [21] proposes to modify data values. Most cleaning techniques adopt this modification model, such as Vrepair [14] considering FDs as constraints, Greedy repair [16] under linear DCs, and Holistic repair [8] with general DCs. All these approaches assume that the constraints are accurate, while our study considers the inaccuracies of both constraints and data. We experimentally compare these approaches in Section 5 to demonstrate the superiority of our proposal.

Constraint Repair. When both constraints and data are dirty, existing methods simultaneously repair the constraints and data. Chiang and Miller [5] propose a unified cost model for both data and constraint repairs. With a model of tuple patterns, the description length based cost is the number of tuples not represented by the model (as well as the number of patterns) times the size of FD. A repair of FD increases its size but may reduce the number of tuples not represented by the model (violations to FD). Beskales et al. [2] consider the relative trust between data and constraints via a threshold of data repair cost. The idea is to consider potential constraint repairs whose bounds of data repair cost are within a threshold τ. Compared to the existing approaches, our proposal shows higher repairing accuracy in Section 5.

7. CONCLUSIONS

In this paper, we study the problem of repairing data under imprecise constraints. Besides inserting predicates for oversimplified constraints (as in existing approaches), we argue that the constraints could be overrefined as well, and need predicate deletion. To be tolerant of constraint variances (with both predicate insertion and deletion), a novel θ-tolerant repair model is introduced, which finds a minimum data repair by allowing a small variation on the constraints (with variation no greater than θ). In particular, our method addresses repairing over numerical attributes which are not supported by existing approaches [5, 2].

We devise efficient pruning among different constraint variations. The one-round, violation-free, repair-cost-bounded data repair algorithm further enables sharing the repair computation. Experiments on real datasets demonstrate that our proposal has significantly higher repair accuracy than the state-of-the-art Unified [5] and Relative [2] approaches (with consideration of constraint repair), while the corresponding time cost is comparable to the pure Holistic data repair [8] (which does not consider constraint variant).

Acknowledgement

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; Tsinghua National Laboratory Special Fund for Big Data Science and Technology; China NSFC under Grants 61572272, 61325008, 61370055 and 61202008.

8. REFERENCES

[1] L. E. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.

[2] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.

[3] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[4] 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.

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

[6] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[7] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

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

[9] W. Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[10] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[11] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[12] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC*, pages 302–311, 1984.

[13] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

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

[15] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *SIGMOD Conference*, pages 1187–1198, 2014.

[16] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *ICDE*, pages 216–225, 2007.

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

[18] J. E. Olson. *Data Quality: The Accuracy Dimension.* Morgan Kaufmann, 2003.

[19] S. Song, A. Zhang, J. Wang, and P. S. Yu. SCREEN: stream data cleaning under speed constraints. In *SIGMOD*, pages 827–841, 2015.

[20] V. V. Vazirani. *Approximation algorithms.* Springer, 2001.

[21] J. Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

APPENDIX

A. PROOFS

Proof of Lemma 1

Proof. Consider the minimum data repair I_1 w.r.t Σ_1. Given I_2 as the minimum data repair w.r.t Σ_2, the conclusion $\Delta(I, I_1) \geq \Delta(I, I_2)$ is naturally proved by showing that I_1 is also a valid data repair w.r.t Σ_2, i.e., to show $I_1 \vDash \Sigma_2$.

First, according to the definition of inverse and implication in Table 1, for any $\phi_1 \in \mathit{Imp}(\phi_2)$, it always has $\overline{\phi_2} \in \mathit{Imp}(\overline{\phi_1})$. For instance, $\leq \in \mathit{Imp}(<)$ implies $\geq \in \mathit{Imp}(>)$.

According to Definition 4 of refinement $\Sigma_1 \preceq \Sigma_2$, for each $\varphi_2 \in \Sigma_2$, there exists a $\varphi_1 \in \Sigma_1$ such that $\varphi_1 \preceq \varphi_2$.

Since $I_1 \vDash \Sigma_1$ (i.e., $I_1 \vDash \varphi_1$), for any $\langle t_i, t_j \rangle \in I_1$, there must exist some $P : x\phi_1 y \in \mathit{pred}(\varphi_1)$ such that $\langle t_i, t_j \rangle \nvDash P : x\phi_1 y$, which can also be written as $\langle t_i, t_j \rangle \vDash x\overline{\phi_1}y$.

Referring to Definition 3 of refinement $\varphi_1 \preceq \varphi_2$, for each $P : x\phi_1 y \in \mathit{pred}(\varphi_1)$, there exists a $Q : x\phi_2 y \in \mathit{pred}(\varphi_2)$ such that $\phi_1 \in \mathit{Imp}(\phi_2)$. It follows $\overline{\phi_2} \in \mathit{Imp}(\overline{\phi_1})$. In other words, $\langle t_i, t_j \rangle \vDash x\overline{\phi_1}y$ implies $\langle t_i, t_j \rangle \vDash x\overline{\phi_2}y$. We have $\langle t_i, t_j \rangle \nvDash Q : x\phi_2 y \in \mathit{pred}(\varphi_2)$ as well, i.e., $\langle t_i, t_j \rangle \vDash \varphi_2$.

To sum up, for each $\varphi_2 \in \Sigma_2$ and $\langle t_i, t_j \rangle \in I_1$, we have $\langle t_i, t_j \rangle \vDash \varphi_2$, i.e., $I_1 \vDash \Sigma_2$. As aforesaid, the conclusion $\Delta(I, I_1) \geq \Delta(I, I_2)$ is proved. □

Proof of Proposition 2

Proof. According to the definition of implication in Table 1, for any $\phi \in \{\leq, \geq, \neq\}$, we can always find a ϕ_1 such that $\phi \in \mathit{Imp}(\phi_1), \phi \neq \phi_1$. For instance, for the operator $\geq$, it always has $\geq \in \mathit{Imp}(>)$, and similarly $\leq \in \mathit{Imp}(<), \neq \in \mathit{Imp}(<)$.

For a $P : x\phi y \in \mathit{pred}(\varphi') \setminus \mathit{pred}(\varphi)$, we can always obtain another φ'' via replacing P of φ' by $Q : x\phi_1 y$, such that $\varphi' \preceq \varphi''$ according to Definition 3 of refinement. Their costs are the same $\mathit{inst}(\varphi, \varphi'') = \mathit{inst}(\varphi, \varphi')$. It concludes that φ' is not maximal. □

Proof of Lemma 3

Proof. To eliminate the violation of a tuple list $\langle t_i, t_j, \dots \rangle \nvDash \varphi$, at least one cell in $\mathit{cell}(t_i, t_j, \dots; \varphi)$ should be modified. It corresponds to at least one vertex in the hyperedge of $\mathit{cell}(t_i, t_j; \varphi)$.

Considering all the violation tuple lists, any valid data repair should at least modify one vertex for each hyperedge, i.e., a vertex cover of the hypergraph.

The minimum cost of modifying a cell is

$$\min_{a \in \mathit{dom}(A)} \mathit{dist}(I(t.A), a),$$

which is exactly the weight of a vertex. In other words, any valid data repair should have repair cost no less than the weight of the corresponding vertex cover.

Therefore, the weight of the minimum weighted vertex cover $\|\mathbb{V}^*(\mathcal{G})\|$ gives a lower bound of any valid data repair. □

Proof of Lemma 4

Proof. For each $\langle t_i, t_j, \dots \rangle$ in the violation set of φ, we have $\langle t_i, t_j, \dots \rangle$ satisfying all the predicates in φ. Referring to the suspect condition $\mathit{sc}(t_\alpha, t_\beta, \dots; \varphi)$, $\langle t_i, t_j, \dots \rangle$ belongs to suspect set as well. □

Proof of Proposition 5

Proof. The correctness of results $I' \models \Sigma$ includes two aspects.

First, we show that I' eliminates all the violations existing in I. For any $\langle t_i, t_j, \dots \rangle \in I$ that satisfies all the predicates of a $\varphi \in \Sigma$, i.e., a violation, there must have a cell in $cell(t_i, t_j, \dots; \varphi)$ that is selected to $\mathbb{C}$ (a vertex cover). That is, there must have a circle (belonging to $\mathbb{C}$) as illustrated in Figure 3(a).

The repair context $rc(t_i, t_j, \dots; \varphi)$ ensures that all the predicates involving cells from $\mathbb{C}$ (red arrows in Figure 3(a)) are not satisfied. In other words, no violation remains in $\langle t_i, t_j, \dots \rangle$ in the repaired I'.

Next, for the other tuple lists in the suspect sets, which satisfy all the predicates that do not contain cells from $\mathbb{C}$ (blue arrows in Figure 3(a)), the aforesaid assignment of cells in $\mathbb{C}$ (circles) guarantees violation-free for each suspect tuple list as well. □

B. DATA REPAIR ALGORITHM

Algorithm 2 presents the violation-free data repair with the aforesaid sharing enabled. As discussed in Section 4.1.3, by gradually assigning cells to *fv* (in Line 16), the algorithm guarantees to return a solution for each component $\mathbb{C}_k$. Before calling an existing solver (in Line 9), the program first searches the materialized subproblems and solutions. If the conditions in Proposition 6 are satisfied, the materialized solution can be directly reused (Lines 5-7).

```
Algorithm 2 DataRepair($\Sigma, I, \mathbb{C}, \delta_{\min}$)
Input: An instance $I$, a constraint set $\Sigma$, a set $\mathbb{C}$ of changing
    cells and a bound of data repair cost $\delta_{\min}$
Output: An assignment of cells in $\mathbb{C}$ for the repair $I'$
 1: initialize $rc(\mathbb{C}, \Sigma)$ and project into components
 2: for each component $\mathbb{C}_k$ of $\mathbb{C}$ do
 3:   while $\mathbb{C}_k \neq \emptyset$ do
 4:     search materialized subproblems
 5:     if exists some $rc(\mathbb{C}_k, \Sigma')$ whose solution satisfies
          $rc(\mathbb{C}_k, \Sigma)$ and $rc(\mathbb{C}_k, \Sigma) \sqsubseteq rc(\mathbb{C}_k, \Sigma')$ then
 6:       $I'(\mathbb{C}_k)$ := the solution of $rc(\mathbb{C}_k, \Sigma')$
 7:       $\mathbb{C}_k := \emptyset$
 8:     else
 9:       solve $rc(\mathbb{C}_k, \Sigma)$ by existing solver
10:       if $rc(\mathbb{C}_k, \Sigma)$ is solved then
11:         $I'(\mathbb{C}_k)$ := the solution returned by the solver
12:         materialize $rc(\mathbb{C}_k, \Sigma)$ with $I'(\mathbb{C}_k)$ for reuse
13:         $\mathbb{C}_k := \emptyset$
14:       else
15:         $t.A$ := the cell with the largest number of appearance in $rc(\mathbb{C}_k, \Sigma)$
16:         $I'(t.A) := fv$
17:         $\mathbb{C}_k := \mathbb{C}_k \setminus \{t.A\}$
18:     if $\Delta(I, I') \geq \delta_{\min}$ then
19:       return null
20: return $I'$
```

Example 13. *Suppose that we are given a set of (repaired) constraints* $\Sigma = \{\varphi_3, \varphi_4'\}$ *from Examples 1 and 4, and* $\mathbb{C} = \{t_2.\mathsf{CP}, t_5.\mathsf{CP}, t_8.\mathsf{CP}, t_4.\mathsf{Tax}\}$ *in Figure 1(a). The repair contexts are* $rc(\mathbb{C}, \Sigma) = \{I'(t_2.\mathsf{CP}) = I(t_3.\mathsf{CP}), I'(t_5.\mathsf{CP}) = I(t_6.\mathsf{CP}), I'(t_8.\mathsf{CP}) = I(t_9.\mathsf{CP})\} \cup rc_1$*, where* rc_1 *is the repair contexts in Figure 4(a) computed in Example 10.* $\mathbb{C}$ *is decomposed into four components* $\mathbb{C}_1 = \{t_2.\mathsf{CP}\}, \mathbb{C}_2 = \{t_5.\mathsf{CP}\}, \mathbb{C}_3 = \{t_8.\mathsf{CP}\}, \mathbb{C}_4 = \{t_4.\mathsf{Tax}\}$ *w.r.t. the repair context independence. By separately solving* $rc(\mathbb{C}_k, \Sigma)$ *for each component* $\mathbb{C}_k$*, the repair results are* $I'(t_2.\mathsf{CP}) = 564\text{-}389$, $I'(t_5.\mathsf{CP}) = 930\text{-}198, I'(t_8.\mathsf{CP}) = 824\text{-}870, I'(t_4.\mathsf{Tax}) = 0$.

Table 2: Approximation performance

Constraints	Approximation factor
linear DC/constant CFD	$d\lvert R\rvert$
binary DC/variable CFD/FD	$2d\lvert R\rvert$

To initialize $rc(\mathbb{C}, \Sigma)$, we need to search suspects in I for each tuple with cells from $\mathbb{C}$. The cost is $O(|I|^\ell)$, where ℓ is the maximum number of tuples $|\{t_\alpha, t_\beta, \dots\}|$ involved in a φ of Σ, as discussed at the end of Section 3. To solve the problem in formula (3), existing LP solvers are employed, e.g., [12] with cost $O(n^{3.5}L)$, where n is the number of variables, i.e., $|\mathbb{C}|$, and L is the number of bits of input. Therefore, for a fixed $\mathbb{C}$, the DataRepair algorithm takes $O(|I|^\ell)$ time.

Constant Factor Approximation

Theorem 7. *Algorithms 1 and 2 output a repair* I' *having* $\frac{\Delta(I,I')}{\Delta(I,I^*)} \leq d \cdot Deg(\Sigma)$*, where* $d = \frac{\max_{a \in dom(A), A \in attr(R)} dist(a, fv)}{\min_{a,b \in dom(A), A \in attr(R)} dist(a,b)}$ *and* I^* *is the optimal θ-tolerant repair with the minimum data repair cost.*

Table 2 reports the performance of our approach under several typical integrity constraints. For a set Σ of linear DCs [16, 1], i.e., with all the predicates in the form of $v\theta c$, we have $Deg(\Sigma) \leq |R|$. The approximation bound becomes $d|R|$. Similar situation occurs in constant CFD [3], which also specifies constraints on single tuples. Binary DCs contain only predicates declared on the same attributes of two tuples, in the form of $t_\alpha.A\theta t_\beta.A$. As special cases of binary DCs, the constraints, including FDs and variable CFDs, share the same bounds $2d|R|$. Since all these types of constraints are declared on tuple pairs, the number of involved cells is at most two times the number of attributes, i.e., $Deg(\Sigma) \leq 2|R|$. (See Section 6 for a more detailed introduction of the aforesaid integrity constraint notations.)

In short, for a R with fixed domain, our proposed approach is a constant factor approximation.

C. DISCUSSION ON USAGE

C.1 Relevance to Real World Usage

In practice, it is already difficult for data curators to understand and accept data repairs, which are suggested by tools following the minimum change property. Automatic repairing techniques (towards constraint satisfaction) play largely a role of suggesting repairs rather than determining repairs. In this sense, suggesting multiple possible repairs, instead of only one repair following the minimum change property and constraint satisfaction, could be helpful to data curators in repair selection. Therefore, in order to enhance the real-world usage, our approach (1) obtains multiple repair results w.r.t. various constraint variants (instead of one single minimum repair satisfying one fixed set of constraints), and (2) provides guidelines on evaluating and selecting the returned multiple repair results. The rationale is explained as follows.

(1) To reduce the complexity of considering both data repairs and constraint repairs at the same time, existing studies [5] employ an unified cost function on both data and

constraint repairs, and the repairing still follows the (unified) cost minimality. As also observed in our experimental evaluation, e.g., in Figure 10, the repair result suggested by unified cost minimality could be doubtful and thus with lower repair accuracy. In order to overcome the problem of returning only one minimum repair result, our proposal considers multiple minimum repair results, suggested by various constraint variants (constraint repairs) w.r.t. different constraint variation levels θ.

(2) To practically understand and select data repairs from the multiple results, one may observe the number of repaired cells in a data repair result. A large number of changed cells (under a small θ) indicate that data is over-repaired (with oversimplified constraints). In contrast, an extremely small size of repaired cells (under a large θ) means that the constraints are overrefined (overfitting the data without identifying violations).

C.2 Prevalence of Imprecise Constraints

The book [18] gives an explanation, together with some examples, of why both the data and the metadata (such as integrity constraints) could be inaccurate at the same time. In particular, the metadata, e.g., constraint on $\mathsf{Age} \geq 18$, are often collected from various internal/external sources, and thus could be inaccurate and incomplete. For instance, "the metadata (constraints) could be extracted from the relational database directory or catalog in a relational system, or a COBOL copybook or a PL/1 INCLUDE file that lays out the data in an IMS or VSAM data source". Moreover, "primary key, foreign key, and other referential constraint information, TRIGGER or STORED PROCEDURE logic embedded within the relational system, application source code for programs that insert, delete, and update information in the database" are also sources of metadata collection. Therefore, it is necessary "to verify some of the information to determine the completeness of (meta)data-gathering and to identify areas where one may suspect wrong or missing information".

C.3 Prevalence of Overrefined Constraints

Overrefined constraints could appear in both manually specified constraints and automatically discovered constraints, two common ways of obtaining integrity constraints. (1) For manually specified constraints, for example, a non-minimal super key {Name, Department, Birthday} could be declared, which includes unnecessary attributes to uniquely identify a tuple. This super key is overrefined, given that a candidate key {Name, Department} is sufficient to uniquely identify a tuple. (2) When discovering constraints from data with noises, it is very like that the discovered constraints may overfit the data. For instance, an overrefined constraint φ_3, i.e., Name,Year,Birthday$\to$CP, could be discovered (with high confidence) from the given data set in Figure 1(a). Such discovery of constraints from data with noises, together with some confidence guarantees, is prevalent when there is no clean data available for discovery [13]. Similar scenarios of overrefined constraints that overfit the data occur in sequential dependency discovery [10] and conditional functional dependency discovery [11].

Without addressing these overrefined constraints, as illustrated in Example 1, dirty data might not be identified and repaired. Therefore, special attention on such overrefined constraints is needed.

C.4 When Errors Fit the Constraints

Real-world errors may fall in the specified constraints, that is, tuples with errors indeed satisfy the specified constraints. There are two cases to take care of: (1) For the case of overrefined constraints, which fail to identify the errors, as φ_3 shown in Example 1, our proposed methods can handle this case by removing excessive predicates; (2) For the case of constraints that are indeed accurate, the proposed method of predicate removing or inserting does not help. Indeed, without further knowledge, all the constraint-based repairing methods fail to identify and repair such errors, which satisfy all the (accurate) constraints.

D. ADDITIONAL EXPERIMENTS

D.1 Experiment Preparation

Count cost is employed in evaluating data repair cost, and an equal weight is associated to each cell (in Definition 1). All the experiments are performed on a machine with 2.5GHz CPU and 8GB RAM.

Datasets. The comparison mainly performs over two real datasets, HOSP and CENSUS. The HOSP dataset includes 14 attributes and 20,442 tuples. The CENSUS dataset is with 40 attributes and 299,285 tuples. The HOSP dataset, from US Department of Health & Human Services, mainly contains categorized data, where up to 6 FDs are specified. The CENSUS dataset, part of the UC Irvine Machine Learning Repository, comes up with 3 DCs over numerical values such as Income.

Following the same line of evaluating data repairing [8], errors are introduced in the datasets by producing noises with a certain error rate. An error rate e denotes that $e\%$ of cells in the data are changed.

Criteria. For categorized data, we use *f-measure* to evaluate the accuracy of repairs. Let truth be the set of original cell values that are changed when introducing noises, and repair be the set of repair results. The repair accuracy is given by *f-measure*$= 2 \cdot \frac{precision \cdot recall}{precision + recall}$, where $precision = \frac{|\mathsf{truth} \cap \mathsf{repair}|}{|\mathsf{repair}|}$ and $recall = \frac{|\mathsf{truth} \cap \mathsf{repair}|}{|\mathsf{truth}|}$. Following the same criteria in [8], for the cells repaired by *fv*, it is counted as a partial 0.5 rather than 1, in $|\mathsf{truth} \cap \mathsf{repair}|$.

For numerical values, we observe mean normalized absolute distance (MNAD) [15], i.e., the distance between I_{truth} (before introducing noises) and I_{repair} (results after repairing). Since the absolute distance measures do not consider the variance of introduced noises I_{noise}, we study the relative accuracy [19] of repairing, $1 - \frac{\Delta(I_{repair}, I_{truth})}{\Delta(I_{repair}, I_{noise}) + \Delta(I_{truth}, I_{noise})}$. According to triangle inequality on distances, in the worst case, $\Delta(I_{repair}, I_{truth}) = \Delta(I_{repair}, I_{noise}) + \Delta(I_{truth}, I_{noise})$, we have the lowest accuracy=0. For the best repair results, $\Delta(I_{repair}, I_{truth}) = 0$, we have accuracy=1.

D.2 Experiments on Predicate Deletion

This experiment evaluates the accuracy and effectiveness of predicate removal. The input constraints are overrefined with excessive predicates and need removal, denoting by negative θ of constraint variance tolerance levels.

First, as illustrated in Figure 16(b), without constraint variance, $\theta = 0$, few data can be correctly repaired (low re-

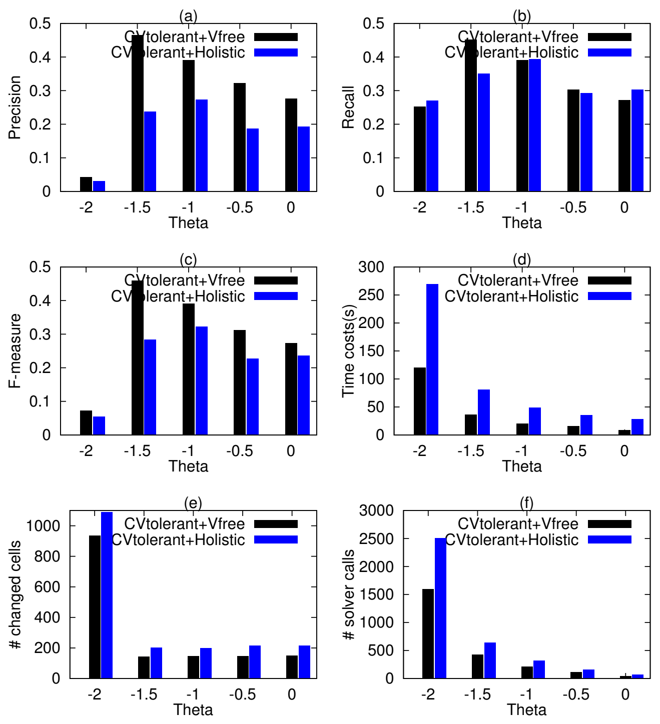

Figure 16: Varying tolerance level θ with predicate removal (HOSP)

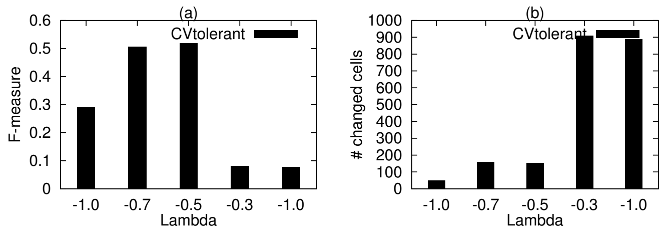

Figure 17: Varying weight λ of predicate deletion (HOSP)

call) under the input overrefined constraints. On the other hand, if too many predicates are removed, e.g., $\theta = -2$, the constraints become oversimplified and a large number of cells will be unnecessarily changed as show in Figure 16(e). Consequently, with a moderate $\theta = -1.5$, i.e., neither oversimplified constraints leading to many changed cells nor overrefined constraints without repairing much data, the accuracy is highest in Figure 16(c). Such phenomenon is also observed in Figure 6(c) of predicate insertion.

D.3 Evaluating the Predicate Deletion Weight

To further distinguish the contributions of predicate insertion and deletion, we conduct another experiment in Figure 17. Instead of a fixed -0.5, we vary the weight λ of predicate deletion, from 0 to -1, relative to predicate insertion. Generally, a smaller λ (closer to -1.0) denotes that more predicate insertions could be afforded, given the same predicate deletion and cost threshold θ. Oversimplified constraints are more likely to be refined and thus fewer cells will be changed in repairing, e.g., weight $\lambda = -0.7$ as shown in Figure 17(b).

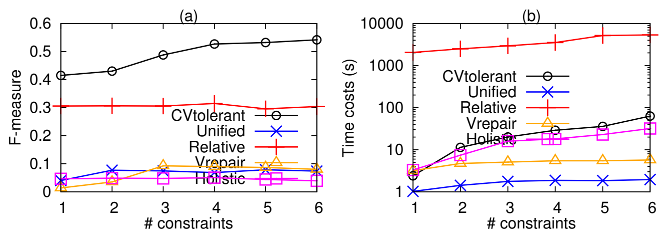

Figure 18: Varying number of FDs (HOSP)

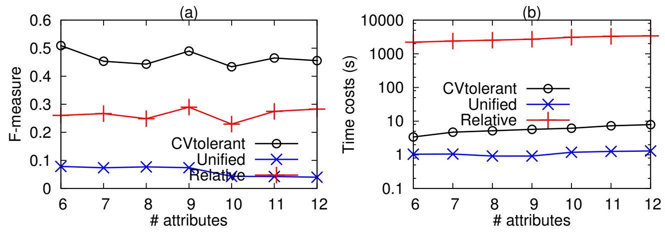

Figure 19: Varying number of attributes (HOSP)

However, for an extremely small $\lambda = -1.0$, the constraints become overrefined and few cells will be repaired. The corresponding accuracy is low as illustrated in Figure 17(a). The results verify our suggestion of avoid setting $\lambda = -1.0$. As shown, a weight $\lambda = -0.5$ (or -0.7), leading to neither oversimplified constraints with many changed cells nor overrefined constraints without repairing much data, achieves the highest repair accuracy.

D.4 Experiments on Various FDs

Figures 18 and 19 report the performance on various numbers of FDs and numbers of attributes. Relative method always tries to repair the data as many as possible (towards the maximum limit τ) in order to minimizing the constraint changes. In this sense, increasing the number of FDs does not help much in reducing unnecessary data repairs and thus has no significant improvement on the repairing accuracy, as shown in Figure 18(a). In contrast, our proposed CVtolerant approach benefits when more constraints are provided.

Figure 19(a) illustrates that with more attributes in a relation, the repairing accuracy is not affected largely. The rationale is that all the constraint repair approaches have mechanism on avoiding inserting many irrelevant predicates into the constraints.

Another interesting result is the slow increase of Relative's time cost (on # constraints and # attributes) in Figures 18(b) and 19(b), in contrast to the result (on # tuples) in Figure 10(b). The reason is that the given constraints are both oversimplified and overrefined, e.g., an FD(State→City) that needs an insertion of Zipcode and a deletion of State. Simply considering more attributes (more FDs) does not help much in reducing the violations to this imprecise FD, and such high (data-repair) cost constraint variants can be efficiently pruned without introducing significantly higher time costs in Figures 18(b) and 19(b). On the other hand, since the high cost data repair is inevitable, the time cost of Relative increases heavily w.r.t. # tuples in Figure 10(b). Nevertheless, our proposed CVtolerant can successfully handle the aforesaid case, with significantly lower time costs in various number of tuples, constraints and attributes.