InfoClean: Protecting Sensitive Information in Data Cleaning

FEI CHIANG and DHRUV GAIROLA, McMaster University

Data quality has become a pervasive challenge for organizations as they wrangle with large, heterogeneous datasets to extract value. Given the proliferation of sensitive and confidential information, it is crucial to consider data privacy concerns during the data cleaning process. For example, in medical database applications, varying levels of privacy are enforced across the attribute values. Attributes such as a patient's country or city of residence may be less sensitive than the patient's prescribed medication. Traditional data cleaning techniques assume the data is openly accessible, without considering the differing levels of information sensitivity. In this work, we take the first steps toward a data cleaning model that integrates privacy as part of the data cleaning process. We present a privacy-aware data cleaning framework that differentiates the information content among the attribute values during the data cleaning process to resolve data inconsistencies while minimizing the amount of information disclosed. Our data repair algorithm includes a set of data disclosure operations that considers the information content of the underlying attribute values, while maximizing data utility. Our evaluation using real datasets shows that our algorithm scales well, and achieves improved performance and comparable repair accuracy against existing data cleaning solutions.

CCS Concepts: • **Information systems** → *Data model extensions*; *Inconsistent data*;

Additional Key Words and Phrases: Data quality, data cleaning, data privacy

ACM Reference format:
Fei Chiang and Dhruv Gairola. 2018. InfoClean: Protecting Sensitive Information in Data Cleaning. *J. Data and Information Quality* 9, 4, Article 22 (April 2018), 26 pages.
https://doi.org/10.1145/3190577

1 INTRODUCTION

The increasing size and complexity of modern datasets have made it increasingly difficult for organizations to extract value from their data. Rising costs to curate, pre-process, and clean the data have been estimated to cost businesses $3.1 trillion per year, in the US alone, in 2016 [40]. A wealth of data cleaning solutions have been proposed in recent years: constraint-based cleaning that uses dependencies as a benchmark to repair data values such that the data and dependencies are consistent [8, 10, 15, 17, 18], quantitative data cleaning that uses statistical methods to identify and repair data quality problems according to expected statistical distributions [6, 24], identifying causality of errors and error propagation [13, 47], and leveraging master data as a source of ground truth [26]. In declarative data cleaning, a set of data updates (repairs) are generated based on transforming the dirty data values to match clean and consistent portions of the data. These clean subsets

Authors' addresses: F. Chiang and D. Gairola, Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario, L8S 4K1, Canada; emails: {fchiang, gairold}@mcmaster.ca.

https://doi.org/10.1145/3190577

of data either: (1) reside in the same relation (as the dirty data) where it is assumed the number of errors is small, and the majority of the relation is clean, or (2) are external knowledge bases and master data sources that are assumed to be trusted and clean.

As increasing amounts of personal and confidential information reside in databases, data cleaning solutions have not kept pace to limit the disclosure of sensitive data during the cleaning process. Existing approaches assume that all data values are equally informative and fully accessible [8, 10, 15, 26, 28]. In reality, this is often not the case. Organizations invest significant efforts to protect their proprietary data, where a set of attribute values is confidential. For example, a patient's medical record describing her diagnosed illness, prescribed medications, and date of birth are quasi-identifiers or sensitive attributes compared to values such as her general symptoms, residential city, or gender.

Applications in the financial, insurance, and health care sectors rely on a curated master data source containing confidential information that is not disclosed publicly but is used as a trusted source for data cleaning. These values have restricted access, either providing limited viewing to a subset of the data, or obscuring the true value(s) via generalization (e.g., revealing a person's age group rather than her actual age). Unfortunately, existing data cleaning approaches do not consider these restrictions, leading to inadvertent disclosure of private data. Although recent work has proposed user-defined data cleaning operations over differentially private relations with provable data privacy bounds for querying, these techniques consider privacy requirements and data cleaning as independent steps by de-coupling these two tasks [36]. In addition, the cleaning operations are applied after privatization, making it difficult to detect and reason about errors over randomized data. User-defined cleaning operations are necessary and are assumed to be the ground truth. The manual burden of tuning parameters to achieve a differentially private relation also limits its adoption in practice.

In this work, we take the first steps toward a privacy-aware data cleaning algorithm that considers the information sensitivity of data values during the data cleaning process. We propose a data cleaning approach that allows data analysts to define the data quality rules that the data should satisfy, and errors are defined as violations to these rules. We quantify privacy as the information content in each attribute value and minimize the total amount of information disclosed. While, differentially, private solutions provide provable privacy guarantees over the data and estimate the level of distortion present in query results, implementing differential privacy in practice is difficult [36]. Meanwhile, existing constraintbased data cleaning solutions, unfortunately, do not consider information disclosure restrictions. Our goal is to provide a consistent data instance (satisfying the constraints) while revealing a minimum amount of information. We propose a pragmatic, information-theoretic data cleaning framework that aims to safeguard data values with high information content from being disclosed during data cleaning. We present a motivating, practical example.

Example 1.1. Table 1 and Table 2 show patient medical records from a pharmacy and hospital database, respectively. Given similar schemas, the tables describe patient identification number (PID), gender (GEN), date of birth (DOB), experienced symptoms (SYMP), prescribed drugs (DRUG), and diagnosed illness (ILLNESS). Given a functional dependency F: [SYMP, DRUG] $\rightarrow$ [ILLNESS] that states for a given set of symptoms and prescribed drugs, we can determine the diagnosed illness. We can see that in Table 1, pharmacy records t_4 - t_6 violate F. These inconsistencies may occur due to user-input error during data integration where F does not hold over all the data sources, or semantic changes due to data evolution. Resolving these inconsistencies is important for accurate record keeping of patients, reliable data analysis, and to avoid propagating these errors in later data processing tasks.

Table 1. Target: Pharmacy Patient Records

tid	GEN	DOB	SYMP	DRUG	ILLNESS
t_1	M	1985	nausea	Ampicillin	meningitis
t_2	F	1991	fatigue	Penicillin	scarlatina
t_3	M	1978	fatigue	Metformin	type2 diabetes
t_4	M	1979	fever	Havrix	**hepatitis A**
t_5	F	1960	fever	Havrix	**hepatitis A**
t_6	F	1960	fever	Havrix	**influenza**
t_7	M	1962	open sore	Aldara	skin cancer
t_8	M	1976	itchiness	Aldara	common wart
t_9	F	1969	itchiness	Clobex	dermatitis
t_{10}	M	1967	diarrhea	Enemas	colitis

Table 2. Master: Hospital Patient Records

tid	PID	GEN	DOB	SYMP	DRUG	ILLNESS
m_1	145	M	1979	fever	Havrix	hepatitis A
m_2	366	F	1972	fever	Penicillin	scarlatina
m_3	257	M	1978	fatigue	Metformin	type2 diabetes
m_4	496	M	1978	fever	Advil	influenza
m_5	365	F	1960	fever	Advil	influenza
m_6	861	M	1948	fatigue	Clofarabine	leukemia
m_7	475	M	1962	open sore	Aldara	skin cancer
m_8	963	M	1962	itchiness	Aldara	common wart
m_9	573	F	1967	itchiness	Clobex	dermatitis
m_{10}	688	M	1967	diarrhea	Enemas	colitis
m_{11}	787	M	1960	fever	Advil	influenza

To resolve these inconsistencies, existing cardinality minimal approaches minimize the number of required updates; that is, we prefer to update t_6[ILLNESS] = `'hepatitis A'` rather than update t_4, t_5 to `influenza` since the former involves only a single update [10, 42]. Set minimal approaches seek repairs I' of I such that there is no subset C of the updated cells in I' that can be reverted to their original values in I (while keeping the current values of cells in $\Delta(I, I') \setminus C$ in I') without violating the functional dependencies. In other words, all repairs in I' are necessary such that if these repairs were reverted, the data would become inconsistent with respect to (w.r.t.) the functional dependencies, e.g., updating the DRUG values in t_4, t_5 to, say, `Vaqta` [7]. If we consult with a trusted master data source (Table 2), we identify two possible repairs based on similar matching records: (1) update t_6[ILLNESS] = `'hepatitis A'` based on tuple m_1; or (2) update t_6[DRUG] = `Advil` based on tuples m_4, m_5, m_{11}. However, in reality, the hospital is reluctant to directly reveal such personal patient information. Intuitively, we would consider a patient's diagnosed illness to be more private than the general symptoms since symptoms can be shared across multiple illnesses. A privacy-aware data cleaning solution would differentiate the degree of sensitive information among the candidate repairs.

We apply information theoretic measures to quantify the amount of information in each attribute value. Values with higher information content are considered more sensitive (private). In our example, we compute the information content between candidate repair values `'hepatitis A'` versus `'Advil'` in Table 2. We also consider repair operations that *generalize* data values to

minimize the amount of revealed information. To find the best repair, we solve a multi-objective optimization problem that minimizes information disclosure, and the number of updates, and maximizes data consistency. We make the following contributions:

(1) We present *InfoClean*, a privacy-aware data cleaning framework that supports the cooperation between a dirty (target) and a clean (master) data source such that the master discloses a minimal amount of information, and the target utilizes this information to (maximally) clean its data. Although our focus in this article is toward data cleaning, our framework has similarities and applications to areas such as consistent data exchange from master data. This is in contrast to existing data cleaning techniques that focus on cleaning in a single relational instance [36].
(2) We propose a data repair operation, *generalization*, that generalizes a repair value to control the amount of revealed information. We define metrics to quantify how much to generalize based on hierarchical relationships from a given ontology.
(3) We present a data repair algorithm that minimizes the information disclosure during the data cleaning process. We model our problem as a multi-objective optimization problem that balances data utility (consistency) against data privacy.
(4) We evaluate our techniques using real datasets to demonstrate the effectiveness and efficiency of our solution. Our results show that we achieve precision values upward from 80% while limiting the disclosure of sensitive values. We also show that InfoClean achieves improved repair accuracy across varying privacy levels against an existing differentially private data cleaning solution.

We introduce preliminary definitions in Section 2, and give an overview of the framework in Section 2.5. We define our repair objectives in Section 3, and present the details of the InfoClean framework in Section 4. We report our evaluation results in Section 5, related work in Section 6, and conclude in Section 7.

2 PRELIMINARIES

We define the dependencies we use in our framework, and introduce ontology and information theoretic definitions. We then define our problem statement and provide a brief overview of our framework.

2.1 Functional Dependencies

For a relation R, a functional dependency (FD) F: $X \rightarrow Y$ defines a relationship between X and Y, where X and Y are attribute sets in R. A data instance I of R satisfies F, denoted $I \vDash F$, if for every pair of records t_1 and t_2 in I, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. Consequently, if $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$, then we say that t_1 and t_2 are inconsistent (dirty) records w.r.t. F. For a set of FDs Σ, if $I \models F$ for every $F \in \Sigma$, then we state $I \models \Sigma$. Without loss of generality, we assume all FDs $F : X \rightarrow Y$ have been decomposed so Y contains a single attribute. While we discuss FDs as the data quality rules, our framework is amenable to other types of dependencies, such as conditional FDs [9], matching dependencies [25], and denial constraints [18].

2.2 Ontologies

An ontology provides a formal description of a domain by modeling the structure of data (via classes, properties, and attributes), and the semantics of the data (via relationships, constraints, and inference axioms). We assume an ontology defines the taxonomy of classes and hierarchical relationships that exist for a particular domain. An example disease ontology is given in Figure 1 [1].

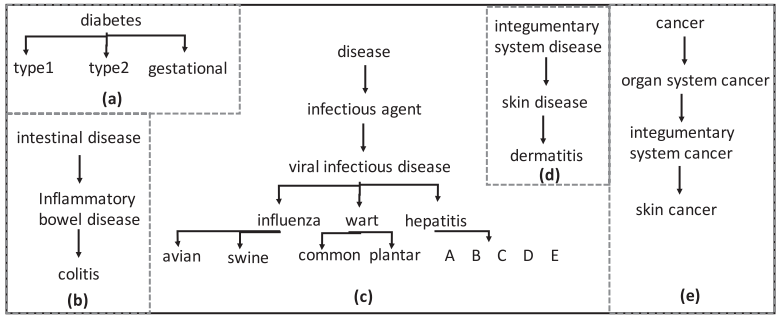

Fig. 1. Example ontologies [1].

2.3 Data Repairs and Record Matching

A data repair updates a data value in T to resolve an inconsistency w.r.t. F (with the goal of $T \models F$). We assume that all inconsistencies occur in a *target* relation T containing tuples t. To correct these errors, T consults with a trusted *master* data source, M, that discloses its data values to T via a trusted third party (TTP). Suppose we have tuples t_1 and $t_2 \in T$ that violate $F : X \rightarrow Y$, so t_1 and t_2 agree on X ($t_1[X] = t_2[X]$) and disagree on Y ($t_1[Y] \neq t_2[Y]$).

To repair a violation, we consider three types of repairs (without loss of generality, we assume $t_2 \in T$ is changed).

Type 1 *Repair the Y values to be the same.*
Here, we would change $t_2[Y]$ to equal $m[Y]$. We will do this when the similarity matching between t_2 and m satisfies the given similarity threshold.

Type 2 *Change the X values to be different.*
Here, we would change $t_2[X]$ to be different from $t_1[X]$. By referencing the matched records $m \in M$, we check with M that one or more of the attribute values in X is incorrect by checking if $m[Y] = t_2[Y]$ and $m[X]$ is close to $t_2[X]$ (where we will define a precise notion of closeness). If so, then we will consider changing $t_2[X]$ to $m[X]$.

Type 3 *Generalize the Y value.*
If we decide that $m[Y]$ is the correct value, we would generalize this value so that less information is disclosed to T. This generalized value, $\text{gen}(m[Y])$, is a node within a given ontology O, where $\text{gen}(m[Y]) \in \text{ancestor}(m[Y])$ (i.e., $\text{gen}(m[Y])$ is an ancestor of $m[Y]$). In our example, the value "`hepatitis A`" can be generalized to "`viral infectious disease`," or further to "`i nfectious agent`" according to Figure 1(c). In Section 3.1.1, we present a new definition of consistency between F and T, when T contains generalized values.

Record Matching. We perform a record matching between records in T and M using the attributes XY. While our framework is amenable to other record-matching techniques [34], we seek records $m \in M$ such that (1) $m[X] = t_1[X]$; (2) $m[Y] = t_1[Y]$; or (3) $m[Y] = t_2[Y]$. Our framework can be extended to consider matching along all record attributes, with preference given to XY, and using additional information from non-FD attribute matches to rank the records.

2.4 Information Theory Basics

We use information theoretic tools to quantify the information disclosure from M.

Self-Information. Self-information quantifies the amount of information associated with the occurrence of an event in a probability space [21].

$$I(x) = \log\left(\frac{1}{p(x)}\right) = -\log(p(x)), \tag{1}$$

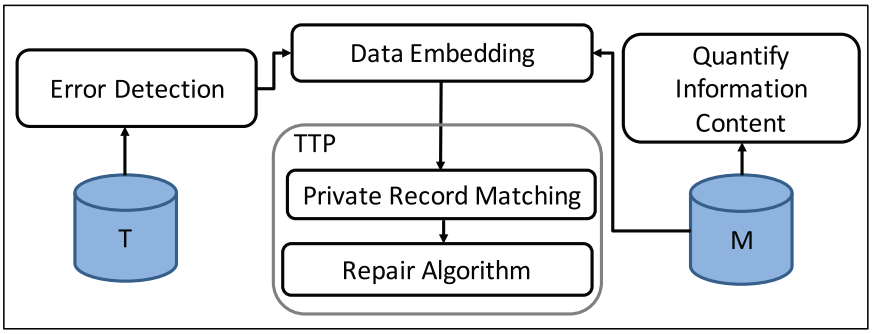

Fig. 2. InfoClean overview.

where $p(x)$ is the probability of event x. The smaller the probability, the larger the self-information. Let R be a relation, and $(X_1, X_2, \ldots X_n) \in R$ be an attribute set that generates a probability space. The co-occurrence of each attribute value $(x_{i1}x_{j2}\ldots x_{kn})$ for $x_{i1} \in X_1, \ldots x_{kn} \in X_n$ corresponds to an event. We assume that the values occur independently and apply self-information across multiple attributes to capture the information content in a set of co-occurring attribute values.

$$I(x_{i1}, x_{j2}, \ldots x_{kn}) = -\log(p(x_{i1})p(x_{j2})\ldots p(x_{kn})) \tag{2}$$

Conditional Entropy. For a given random variable Y, the conditional entropy of X given $Y = y$ is defined as

$$H(X|y) = -\sum_{x \in X} p(x|y) \log p(x|y), \tag{3}$$

where $p(x|y) = \frac{p(x,y)}{p(y)}$ is the conditional probability of x given y. Conditional entropy, $H(X|y)$, measures the uncertainty in X when y is known. For a given F, the conditional entropy measures the information in X when $Y = y$ is given as $H(X_1X_2\ldots X_n|Y = y) = -\sum_{x_j \in X_i} p(x_jx_k\ldots x_n|y) \log p(x_jx_k\ldots x_n|y)$.

2.5 Framework Overview

Problem Statement: Given a set of FDs Σ defined over T, where $T \not\models \Sigma$, and master data M, identify a set of data repairs to T such that $T \models \Sigma$, and the information disclosed from M is minimized.

Solution Overview. Figure 2 shows the architecture of our framework. Given a set of FDs Σ defined over T, the *error detection* module identifies all tuples $t \in T$ that violate an $F : X \rightarrow Y$, $F \in \Sigma$. We cluster these inconsistent tuples according to their X values, and sort in descending order by frequency. We adopt a greedy approach, and clean error tuples that occur most frequently first. To resolve these inconsistencies, T communicates with master data M via a TTP. The data from both T and M are obfuscated in the *data embedding* module. All data exchange between T, M, and the TTP use the embedded values to minimize information disclosure among all participants. We measure the amount of information contained in each value of M in the *quantify information content* module.

In *private record matching*, the TTP matches each inconsistent record $t \in T$ against those from M, and identifies a set of records $m \in M$ that are most similar (i.e., satisfying a similarity threshold). Based on these matches, we generate a space of repairs to update values in t based on values from m. In the *repair algorithm* module, we traverse the space of candidate repairs and solve a multi-objective optimization problem to identify repairs that maximize data consistency (i.e., data utility), and minimize information disclosure (privacy loss), and the number of updates to T.

3 REPAIR OBJECTIVES

InfoClean limits the disclosure of private information in M, while maximizing data consistency for T via the fewest number of updates. We discuss each of these objectives next.

3.1 Minimizing Information Disclosure

We apply an information theoretic approach and quantify privacy as the amount of information content contained in each value of M. Privacy loss is then defined as the amount of information loss incurred when M discloses a value $v \in M$ to T. We model the privacy of M by assigning an information content score to each *cell* in M that represents the value of information for the value in that cell. We compute the information content scores w.r.t. $F : X \rightarrow Y$, and construct a table M_{info}, with schema XY that contains the information content score of each value.

We define a score (similar to one used in Arenas and Libkin [4]) with extensions that not only consider the value itself, but also the frequency of occurrence within an attribute. The score is based on computing the information gain of losing a specific data value in M, and then some value is replaced back (this could be any value in the attribute domain). For example, suppose F: [SYMP, DRUG] $\rightarrow$ [ILLNESS] holds over Table 2. Suppose we lose the value in m_5[ILLNESS], we can infer that the lost value could be "`influenza`" based on the value from m_4. Hence, the information content of m_5[ILLNESS] is low. In contrast, if we lose the value in m_5[SYMP], then it is more difficult to infer the value. This is because either "`fever`" or perhaps some other value outside the attribute domain could be substituted and still satisfy F. Hence, the information content of m_5[SYMP] should be higher since it is more difficult to infer this value.

Let $adom(A)$ be the active domain of an attribute $A \in M$. Since it is possible that some value outside of the active domain can be substituted and F would still hold, we ensure that at least one value v exists outside the active domain, and use $adom(A) \cup v$ as the domain of A. Let $M_{c \leftarrow a}$ denote an instance constructed from the master dataset M by replacing the attribute value in cell c by a. Define a probability space for each cell c, $\mathcal{E}(M, c) = (adom(A) \cup v, P)$, where P is the probability density function given by:

$$P(a) = \begin{cases} 0 & M_{c \leftarrow a} \not\models F \\ \frac{1}{|b|} \cdot freq(a) & otherwise \end{cases} \tag{4}$$

and $b = \{a | M_{c \leftarrow a} \models F\}$. The information content of the cell c with respect to F is:

$$inf(c) = H(\mathcal{E}(M, c)) = \sum_{a \in adom(A) \cup v} P(a) \log \frac{1}{P(a)} \tag{5}$$

We define $freq(a) = \frac{num(a)}{|M|+1}$, where $num(a)$ is the number of cells that contain the value a in M. We add 1 to the denominator to include the value v that lies outside the active domain. By including frequency, we differentiate among the varying levels of information content, as values with higher frequency (more redundancy) are less informative.

Example 3.1. Referring to Table 2, define relations, I_1 and I_2, consisting of (m_4, m_5), and ($I_1 \cup m_{11}$), respectively, with a schema of (SYMP, DRUG). Let c_1 and c_2 represent the cell m_4[DRUG] in I_1 and I_2, respectively. We compute $inf(c_1) = \frac{1}{3} \log(\frac{3}{1}) = 0.159$, while $inf(c_2) = \frac{2}{4} \log(\frac{4}{2}) = 0.151$, reflecting that c_2 contains less information than c_1.

In context of the entire Table 2, we explain how to compute the information content score $inf(c)$ for cell c equal to m_4[`SYMP`]. The value m_4[`SYMP`] = "`fever`" can be replaced with other values a from the attribute domain that would still allow us to satisfy $F : [SYMP, DRUG] \rightarrow [ILLNESS]$. There are five such values from the attribute domain, namely {"`fever,`" "`fatigue,`" "`open sore,`" "`itchiness,`" "`diarrhea`"}. We compute the probability $P(a)$ for each of these a values. For example, for $P(a)$ where a is "`fever,`" we obtain $P(a) = \frac{1}{5} \cdot \frac{5}{66+1} = \frac{1}{67}$, where the value "`fever`" occurs

five times across all 66 cells of M (we exclude column `tid`), and we +1 to consider a value outside the attribute domain. The information content score is given by Equation (5), resulting in a value of 0.027. For brevity, we omit similar details to calculate $P(a)$ and inf for the remaining four values. We obtain a final information content score, $inf(c) = 0.068$.

A candidate repair r updates values $t \in T$ based on values $m[A_i] \in M$, for attributes A_i. We compute the privacy loss incurred by M as $pvt(r) = \sum_{(m[A_i])} inf(m[A_i])$, where we sum the inf scores for each disclosed $m[A_i]$ value.

3.1.1 Generalization. Our current levers to control the amount of information disclosure is deciding whether a value should be disclosed or not. By disclosing a value, we provide T with clean data, thereby increasing its utility. While value suppression incurs zero privacy loss, there is no utility gain for T. We would like to provide a more flexible repair mechanism that allows M to disclose some useful information to T but constrains the amount of privacy loss.

We propose a new repair operation called *generalization* that discloses a semantically-related, but broader value than the actual value $v \in M$. For example, in Table 2, instead of disclosing m_1[ILLNESS] = "`hepatitis A`," we refer to Figure 1, and we disclose the category in which "`hepatitis A`" belongs to, namely, "`viral infectious disease.`" This reveals semantically related, but less specific information to T.

Our framework supports generalization for both numeric and categorical values. For numeric values, we seek a range $[a, b]$ in which a value v belongs, e.g., `age = 24` can be generalized to `age ∈ [20-30]`. For categorical values, an ontology O provides a hierarchical taxonomy of related terms, organized by levels $l \in \{1 \ldots k\}$.[1] Values in M appear at the leaf (ground) level $l = 1$.

We revise the pvt metric to consider the level where the generalized value occurs. Let $g = \text{gen}(m[A_i])$ denote the generalized value of $m[A_i]$. We multiply the inf score by a weight $w_g = \frac{1}{l_g \cdot d_g}$, where d_g represents the number of descendants of g, and l_g represents the level where g occurs in O. The idea is that by disclosing a generalized version of $m[A_i]$, the chances of inferring the actual value $m[A_i]$ becomes more difficult as g occurs in higher levels of O, and as g contains more descendants. The weight w_g discounts the information disclosed according to these two properties. For values at the ground level, we set $(d_g \cdot l_g) = 1$ as a base case.

FD Consistency. Given the presence of generalized values in T, we need to redefine the notion of consistency w.r.t. F. Let T' be T updated with generalized values.

Definition 3.2. Relation T' satisfies F, denoted $T' \models F$, if for every pair of tuples $t'_1, t'_2 \in T'$, if $t'_1[X] = t'_2[X] = a$, where a is either: (i) a ground (non-generalized) value or (ii) a is generalized but has only one child, then $t'_1[Y] = t'_2[Y] = b$ (where b is a ground value), or $t'_1[Y]$ overlaps $t'_2[Y]$ (i.e., $t'_1[Y]$ is an ancestor of $t'_2[Y]$).

Example 3.3. Continuing our example, instead of disclosing "`hepatitis A`," M discloses the generalized value "`viral infectious disease.`" We update T t_4[ILLNESS] = t_5[ILLNESS] = "`viral infectious disease.`" To verify consistency between T and F: [SYMP, DRUG] → [ILLNESS], we check t_4 (similarly for t_5) and t_6, and determine that they are consistent (Definition 3.2) since the value in t_4[ILLNESS] overlaps the value in t_6[ILLNESS] = "`influenza.`"

3.2 Data Utility and Number of Updates

Data Utility. We define the utility of T as the degree of consistency between T and $F : X \rightarrow Y$. We apply an information theoretic approach by quantifying the amount of information we have about

[1]For numeric values, we can consider a hierarchy similar to categorical values that partition ranges into equal widths for each child node.

Y when X is known. If this knowledge of X consistently provides us with the same knowledge of Y, then there is no uncertainty. This can be modeled via entropy (with a value of zero). Otherwise, the uncertainty of which values we observe in Y is modeled via a non-zero entropy value.

This intuition is captured via the *InD* metric, defined as $InD = H(XY) - H(X)$, where $H(A)$ is the entropy function $H(A) = \sum_{j=1}^{n} p_j \log \frac{1}{p_j}$, for attribute A with values $A = \{a_1, \ldots, a_n\}$, and probabilities $\{p_1, \ldots, p_n\}$ [22]. If $InD = 0$, then $T \vDash F$; otherwise, $T \nvDash F$. Higher *InD* values indicate greater inconsistency between T and F.

Example 3.4. We compute the *InD* score for T (Table 1), where $X = \{\text{SYMP, DRUG}\}$, and $Y = \{\text{ILLNESS}\}$. We get $H(XY) - H(X) = (8 * (\frac{1}{10} \log 10) + \frac{2}{10} \log 5) - (7 * (\frac{1}{10} \log 10) + \frac{3}{10} \log \frac{10}{3}) = 0.082$. If we update $t_6[\text{ILLNESS}] =$ "`hepatitis A`," we get $H(XY) = H(X)$ and $InD = 0$, indicating $T \vDash F$.

Number of Updates. We seek repairs that make the fewest number of changes to T to satisfy F. Our goal is to identify a set of repairs that minimizes the privacy loss in M and maximizes the data utility in T via the fewest number of updates.

We note that all objectives, *pvt*, *InD*, and *#upd* are normalized to a value within [0,1] for a repair candidate s'. For *pvt*, we divide $pvt(s')$ by the total information content across all cells in M. We measure the utility of s' on T by dividing $InD(s', T)$ (the utility of T after s' is applied) by $InD(\{\}, T)$, which represents the utility of the current version of T with no repairs applied (represented by {}). Lastly, we normalize *#upd* by dividing it by the number of changes across all repairs.

4 INFOCLEAN

We present details of InfoClean, a privacy-aware data cleaning framework. We first discuss the metric embeddings we use, followed by the record-matching process, the multi-objective optimization function, and finally, the data repair algorithm.

4.1 Trusted Third Party

To perform private record matching, pairs of attribute values must be compared using comparison functions to evaluate the distance between these values, all without knowing the actual values. Encrypting the values is not a feasible option since encryption generally does not preserve the distance between the original values.

We use a TTP that performs the record-matching task and generates a set of repair recommendations to resolve the inconsistencies in T. We assume the TTP is semi-honest and follows protocol but can infer information about T and M. To avoid inadvertent information disclosure, we apply a metric embedding that obfuscates the data by mapping the values into a vector space, such that the actual data values are hidden from the TTP. By transforming the actual values into a metric space, we preserve the distances between the record values. The TTP is then able to compare the values (using the embedded versions), while data privacy is preserved. Several methods exist to embed a set of objects in a metric space, including FastMap, multi-dimensional scaling, and SparseMap [31]. We apply the SparseMap metric embedding for its privacy guarantees and improved performance over FastMap. SparseMap is an instance of Lipschitz embeddings and contains optimizations for efficient implementation in practice. We briefly discuss SparseMap, and we refer the reader to Hjaltason and Samet [31], and Scannapieco et al. [44] for further details.

4.2 Data Embeddings

SparseMap. Lipschitz embeddings define a coordinate space where each axis is defined on the notion of a *reference set*. A reference set consists of a subset of the objects to be embedded, which, in our case, are the data values from T and M. Given B objects to be embedded, and a distance

d in the original space, the embeddings are defined via a set S of subsets from B. Specifically, $S = \{S_1, S_2 \ldots S_k\}$, where each S_i is a reference set. For an object $b \in B$, the mapping M for an object b is defined as $M(b) = (d(b, S_1), \ldots d(b, S_k))$, where $d(b, S_i)$ is the minimum distance $d(b, s)$ for each $s \in S_i$. Hence, the embedding models the coordinates of b as the distance of b to the closest element in each reference set S_i. The embedding exploits the triangle inequality to obtain a lower bound for $d(b_1, b_2)$. If $|d(b_1, x) - d(b_2, x)| \leq d(b_1, b_2)$, then when we are considering sets that are subsets of the type $|d(b_1, x) - d(b_2, x)|$, we can obtain a lower bound for $d(b_1, b_2)$. This likely occurs when the reference sets are randomly chosen according to two properties: (1) the number of subsets of B is approximated by $\lfloor \log_2 N \rfloor^2$, where N is the number of strings; and (2) $|S_i| = 2^q$, where $q = \lfloor (i-1)/(\log_2 N) + 1 \rfloor$. In our case, the embedded objects consist of a set of strings, modeled as vectors. The most commonly used distance function is Euclidean distance.

Lipschitz embeddings generally suffer from two drawbacks. First, to embed an object b, $O(N^2)$ distance computations are required since we must compare every pair of objects. This quickly becomes problematic for large N. Second, the number of subsets that must be considered is large, equal to $\lfloor \log_2 N \rfloor$.[2] SparseMap addresses these two issues by defining heuristics to: (1) approximate the distance between b and a subset S_i, and (2) perform greedy sampling to select the best coordinates by considering previously selected coordinates [31]. These heuristics help to reduce the cost of building the space at the expense of a lower quality embedding. To handle this, we have used a SparseMap embedding that guarantees *contractiveness*, similar to past work [44]. The contractiveness property ensures that all distances in the embedded space are a lower bound of distances in the original space. This helps to improve the quality of the embedding by improving the recall accuracy, i.e., the number of correct results found in the embedded space compared to the total number of results from the original space.

Embed T and M. We describe how SparseMap is applied to our framework. The embedding consists of four steps:

(1) Build a generator set. Both T and M agree on a set of random strings that will be used as the generator set, i.e., a set of strings that generate the reference set.
(2) Build a reference set. The reference set is used to embed the actual data records.
(3) Embed the reference set. To reduce the number of distance computations, we can first embed the reference set to obtain an *embedded reference set*, which is then used in conjunction with the reference set, to embed the data records.
(4) Embed the records using both the reference set and the embedded reference set. The embedded records are sent to the TTP who performs record matching.

The generator set consists of a set of strings that will be used to build the reference set. Both M and T decide on the number of strings (N) to be used for the embedding and the length of each string in the set L_s. M and T each generate their own random strings that are included in the generator set. Each element of the reference set is a randomly chosen subset of the generator set.

Example 4.1. Suppose M and T would like to exchange information in attribute SYMP. If M and T agree that $N = 10$ and $L_s = 7$; then, a generator set would consist of 10 randomly generated strings, each of length 7. For example, one such generator set would be (GEtuzDz, DUOcGCk, DZHOgUD, XDtiVZm, JyeruGf, bKLQQID, uOxjCpT, rJmnZaU, OfmUnlC, MJZCIOn). To build the reference set, which is a (randomly chosen) subset of the generator set, we approximate the number of subsets by $\lfloor \log_2 N \rfloor^2$, where N is the size of the generator set. If the subsets are given by $\{S_1, S_2, \ldots, S_9\}$, each subset S_i has size 2^q where $q = \lfloor (i-1)/(\log_2 N) + 1 \rfloor$. One possible reference set is $\{S_1, S_2, \ldots, S_9\} \leftarrow$ {(DZHOgUD, uOxjCpT), (DZHOgUD, DUOcGCk), …, (JyeruGf, JyeruGf,

bKLQQID, OfmUnlC, DZHOgUD, uOxjCpT, DZHOgUD, DUOcGCk)}. We have 9 elements in the reference set since $\lfloor \log_2 10 \rfloor^2 = 9$.

Once a reference set is defined, we can embed T and M's records. However, embedding the records using the reference set has been shown to be expensive. SparseMap provides a faster alternative that involves embedding the reference set itself to reduce the number of computations. To embed the reference set, the distance function d accepts a string as a first argument, and a set of strings as the second argument, and returns the closest distance between the first argument and a member of the second argument. This process repeats for each subset S_i, producing embedded (numeric) vectors $z \in S'_i$ for each $\{S'_1, \ldots, S'_k\}$. In the final step, M and T embed their records using $\{S_1, \ldots, S_k\}$ and $\{S'_1, \ldots, S'_k\}$. Algorithm 1 and Algorithm 2 present a skeleton of the process. In Algorithm 1, the generator set $\{g_0, \ldots, g_N\}$ is used to create reference set $\{S_0, \ldots, S_w\}$, which is then embedded to reduce the number of future computations. We compute the distance between each reference set in computeDistance() (Algorithm 2) and embed each relation T and M.

ALGORITHM 1: embedData(T, M)

$\{g_0, \ldots, g_N\} \leftarrow N$ random strings each of length L_s
$\{S_0, \ldots, S_w\} \leftarrow$ select w random subsets of $g_0, \ldots, g_N$
$\{S'_0, \ldots, S'_w\} \leftarrow$ embed $\{S_0, \ldots, S_w\}$
$T' \leftarrow$ **computeDistance**(T), and reduce dimension from w to (user-defined) k.
$M' \leftarrow$ **computeDistance**(M)
return (T', M')

ALGORITHM 2: computeDistance(v)
Given value v, embed v w.r.t. $\{S_0, \ldots, S_k\}$ and $\{S'_0, \ldots, S'_k\}$.

```
D[0..k] ← 0
mx ← max integer
p ← 0
D[0] ← d(v, S_1)
for S'_i ∈ {S'_2, ..., S'_k} do
    for z_j ∈ S'_i do
        if mx > |D[i − 1] − z_j[i − 1]| then
            p ← j
            mx ← |D[i − 1] − z_j[i − 1]|
        end if
    end for
    D[i] ← edit_distance(v, S_i[p])
    p ← 0
    mx ← max integer
end for
return D
```

4.3 Private Record Matching

The TTP performs record matching between records in embedded relations T' and M'. For each error tuple $t' \in T'$, the TTP compares t' with $m' \in M'$ along the XY attributes for a given $F : X \rightarrow Y$. Specifically, the TTP checks whether (i) $t'[X] \approx m'[X]$ and $t'[Y] \neq m'[Y]$ or (ii) $t'[X] \neq m'[X]$

and $t'[Y] \approx m'[Y]$, where $\approx$ indicates that t' and m' are sufficiently close according to a distance function. We use the Euclidean distance function `eucl()` and a threshold τ, and the TTP checks whether $\texttt{eucl}(t', m') > \tau$. If so, then the TTP uses $m'[X]$ (or $m'[Y]$) to repair $t'[X]$ (or $t'[Y]$), respectively.

Since the SparseMap embedding preserves the original distances between the actual values, computing Euclidean distances between T' and M' allows us to determine record similarity without compromising privacy. Our goal is to resolve errors in XY w.r.t. F, and we perform record matching using the XY attributes. Our framework is amenable to consider matching using a greater number of attributes (i.e., attributes not in F, $(R \setminus XY)$). However, while this may improve the record-matching accuracy (by having additional data points for comparison), the increased space of repair candidates needs to be carefully controlled to limit the number of false positives.

From the record matching, we generate a space of repairs (values from M that will be used to repair T), and seek repairs r containing values that minimize the privacy loss (pvt) for M, maximize the data utility (InD) for T, and minimize the number of updates to T. This problem involves competing goals, e.g., sacrificing privacy improves utility, we model this as a multi-objective problem, where more than one Pareto optimal solution may exist [12]. We apply a weighted optimization function that allows users to define weights for each objective indicating their relative importance. We present the weighted optimization function next, followed by the repair algorithm.

4.4 Weighted Optimization Function

To optimize the three objectives: privacy, utility, and the number of updates, we apply a weighted optimization function that models each objective along with a weight as follows,

$$\begin{aligned} &\min \quad \sum_{i=1}^{k} w_i f_i(x) \\ &x \in S \end{aligned} \tag{6}$$

For a set of objectives $\{f_1, \ldots, f_k\}$, and corresponding weights $w_i > 0$, $\sum_{i=1}^{k} w_i = 1$ [11], where S is the space of possible repairs. We model each of our three objectives: privacy (pvt), utility (InD), and number of updates (upd) via one of the f_i functions, and define weights α, β, and γ, respectively. We seek a repair r that minimizes the sum of the weighted function, $wsum(r)$, given below,

$$wsum(r) = (\alpha \cdot pvt(r)) + (\beta \cdot InD(T)) + (\gamma \cdot \#upd(r)) \tag{7}$$

We use $wsum(r)$ to evaluate the quality of a repair candidate r when evaluating the space of possible repairs (described in the next section). The weighted function provides the flexibility for users to specify the relative importance of the objectives according to their application requirements. We note two limitations of the weighted function:

(1) Our problem is a non-convex optimization problem, where the complete Pareto-optimal set of solutions may not be found [11, 12]. This indicates that some Pareto-optimal solutions can never be reached, regardless of the weight settings. In our search algorithm, we do not seek the complete solution set, only a single solution. As discussed in the next section, if no optimal solutions are found, we can re-start the search with a new initialization point.

(2) Finding the optimal weight settings for different applications requires tuning. The standard practice is to tune the parameters by evaluating different weight settings in order to gauge the relative importance of different objectives, and to explore the tradeoff between the objectives [11]. We expect that higher α levels will lead to reduced performance due

to overhead costs for computing the information content values. Setting $\alpha = 0$ reduces our solution to a non-privacy aware data cleaning solution. In Section 5.2, we evaluate different weight settings to gauge the relative influence of *pvt*, *ind*, and *changes* on repair accuracy.

4.5 Repair Algorithm

We use simulated annealing to navigate the space of possible repairs. For each candidate repair, r, we use the previously defined cost function $wsum(r)$ to identify an r that minimizes the objectives. The simulated annealing search algorithm adopts a strategy where it initially accepts a less than optimal solution with a high probability (at high temperatures), and then gradually lowers this acceptance probability at lower temperatures. The rate at which the temperature is lowered is determined by the cooling schedule. If the cooling schedule is lowered slowly enough, the algorithm will find an optimum solution with probability approaching one [29, 41]. Simulated annealing is known to be sensitive to the search starting point. To avoid cases where the search may be trapped into a local optimum, without making progress toward the global optimum, we aim to define good initializations that start the search at candidate repairs, which capture desired characteristics in the optimization function. If we have a good candidate at initialization, then the search algorithm should identify a repair that minimizes our objective function by the time the final (user-defined) temperature is reached, at which point the search terminates. The search starts at an initial repair candidate, defined as the current solution, r_{cur}. Next, we randomly select a neighbor, r_{nbr}, and check if r_{nbr} further minimizes our objective function. If so, then it is marked as the current repair; otherwise, we mark r_{nbr} as the current solution with some probability p_j.

Each iteration of the algorithm is associated with a temperature K that is determined by a cooling schedule. A commonly used cooling schedule is $K_z = \eta * K_{z-1}$, where K_z represents the temperature at some time z, and η is the cooling rate between (0,1). Given the current temperature K, the probability function P is defined as $P(K, r_{nbr}, r_{cur}) = exp(\frac{-(r_{nbr} - r_{cur})}{K})$. The search continues until the final (user-defined) temperature is achieved. To begin the search, we define: (i) how an initial repair candidate is defined; and (ii) how to define the neighborhood for r_{cur}.

4.5.1 Search Initialization. The choice of the initial repair candidate influences the quality of the final solution. We propose two initialization strategies: greedy and random initialization. In greedy initialization, we generate a repair candidate r_g consisting of values from master tuple $m \in M$ where $\texttt{sim}(m, t)$ is maximal (`sim()` is the similarity function used during record matching). The intuition is we seek a repair r_g with values similar to the clean values in tuple m. This will help to repair the dirty values in t, increase the utility, and minimize InD. In random initialization, we randomly select values from the set of matching tuples $m \in M$ to include in the starting repair r_{rand}, without preference for the top-matched tuples.

4.5.2 Defining the Neighborhood. From the current repair candidate r_{cur}, our simulated annealing search will visit a neighboring repair r_{nbr}, defined as: (i) r_{nbr} differs from r_{cur} in one attribute, to consider (minimal) incremental changes between successive repairs; or (ii) $r_{nbr}[XY] = m_i[XY]$, i.e., the values in r_{nbr} are from a master tuple m_i, $m_i \neq m_j$, where $r_{cur}[XY] = m_j[XY]$. In the latter case, we want to ensure that we do not make repair recommendations from different master tuples, i.e., we do not recommend a repair on attribute X from m_i and a repair on attribute Y from m_j.

4.5.3 Search Algorithm. Algorithm 3 gives details of our search algorithm based on simulated annealing. Given embedded relations T', M', the information content table, and the FD F, we initialize the repair set s to the empty set, set the starting repair, and assign the temperature schedule to K (lines 1-7). We visit a neighbor repair r_{nbr}, and compare the current repair r_{cur} against r_{nbr}. If

ALGORITHM 3: genRepairs(T', M', M_{info}, F)

```
1: s ← ∅
2: r ← initialize()
3: r_cur ← r
4: for i ← 1 to ∞ do
5:     K ← schedule[i]
6:     if K = 0 then return s
7:     end if
8:     r_nbr ← getNeighbor(r)
9:     r_cur ← nextCandidate(r_cur, r_nbr, K, M_info)
10:    if r_nbr is best solution so far then
11:        r_cur ← r_nbr
12:    end if
13:    if r_nbr minimizes the objective function then
14:        s ← s ∪ r_nbr
15:    end if
16: end for
17: return s
```

r_{nbr} minimizes our objective function, and is as good as r_{cur}, then r_{nbr} is added to s, and updated to r_{cur} if it is the best solution thus far (lines 8-14).

Algorithm 4 shows details of the main driver routine, InfoClean(). InfoClean takes input relations T, M, and FD F, and identifies errors over T with respect to (w.r.t.) F (lines 3, 4). These error violations V are processed in decreasing cardinality, i.e., tuples with the largest number of errors are processed first (line 5). For each error tuple t' that participates in an (error) violation V_i, we seek repairs to correct t' by calling genRepairs(). After evaluating the space of repairs, we select the repair s' containing the fewest number of updates to T, and assign this to r. The generalize() function computes the information loss of disclosing the values $a \in r$, and determines whether we should disclose a or generalized values of a (lines 6-14). InfoClean completes after processing all error tuples in V and disclosing values to T for cleaning.

4.6 Information-Theoretic Risk

In this section, we provide a brief discussion of extensions to our information-theoretic privacy model to bound the overall privacy loss and identity disclosure for a record in the master relation M.

4.6.1 Privacy Properties. Past work has studied the use of information theory to quantify privacy loss and used rate distortion theory to quantify the utility to privacy tradeoff limits [43]. In particular, Sankar et al. show that a tight relationship exists between utility U, privacy Γ, and the minimal information rate, which is the number of information bits disclosed per record in a relation R [43]. Fixing the value of any one determines the other two, e.g., fixing the utility U quantifies the maximal achievable privacy Γ, and the minimal information disclosure rate $R(U, E)$, which is lower bounded by E relative to Γ.

In our model, we measure the information bits per attribute value (i.e., per cell in relation M). Given an upper bound Φ on the privacy loss, InfoClean guarantees information disclosure across all attributes will not exceed Φ. This indicates that T's knowledge about M is not increased beyond Φ by interacting with M via the TTP. By extending InfoClean's privacy loss model from attribute

ALGORITHM 4: InfoClean(T, M, F)

```
(T', M', M_F) ← embedData(T, M, F)
M_info ← computeInfoContent(M, F)
V ← calcViolations(T, F)
{V_0, ..., V_k} ← orderViolations(V, F)
for V_i ∈ {V_0, ..., V_k} do
    for t' ∈ V_i do
        s ← genRepairs(t', M', M_info, F)
        Select the repair s' of minimal size (fewest updates)
        r ← {s' | s' ∈ s, min |s'|}
        generalize(r, M_info)
        for a ∈ r do
            Disclose (actual or generalized) a to T.
        end for
    end for
end for
```

to record level, the privacy model investigated by Sankar et al. can be applied to provide tight bounds on the maximal achievable privacy for a given utility level, and vice versa. We intend to study this further in future work, along with extensions to other privacy characterizations such as k-anonymity [45] and t-closeness [37].

4.6.2 Identity Disclosure. Our current disclosure model is based on heuristics with the goal of safeguarding values with the highest amount of information content w.r.t. an FD $F : X \rightarrow Y$. For an individual record $m \in M$, we can measure the risk of identity disclosure using mutual information. The *mutual information* $I(X;Y)$ between two random variables X and Y measures the mutual dependence of the two variables, and is defined in terms of entropy as:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{8}$$

Let m be a record in M with original attributes X and Y. Define m' as the disclosed version of m containing attributes X', Y, where X' is a perturbed version of X that may be an equivalent, generalized, or suppressed value of X. The risk of identity disclosure for a record m can be viewed as the difference in mutual information between the disclosed values in m' and the original values in m:

$$I(X;Y) - I(X';Y) \geq 0 \tag{9}$$

Equation (9) states that when $m = m'$, then the difference in mutual information is zero. We can upper bound this identity risk by a constant δ for each individual record m. Alternatively, we can consider an alternative approach similar to t-closeness [37] that places an upper bound $I(X';Y)$ such that the distance between the distribution of values in Y for each unique X' set of tuples to the values in the original data M be no more than a constant t.

5 EXPERIMENTS

We now turn to the evaluation of our techniques using real datasets. Our evaluation focuses on four objectives:

(1) We measure the accuracy of our proposed repairs as we vary the number of tuples, error rate, similarity threshold, optimization parameters, FD complexity, cooling rate, and error skew.

(2) A performance study evaluating the scalability of our techniques w.r.t. the number of records, and the error rate.
(3) We compare our solution against PrivateClean [36], an existing differentially private data cleaning solution that applies cleaning operations over privatized relations. We show that our techniques achieve an average of 8x an improvement in accuracy and a 50% gain in performance due to the inherent difficulty of detecting and cleaning errors over randomized data.
(4) A comparison study of our data repair approach against an existing data repair algorithm, in open (no privacy) settings [15]. We validate that our data cleaning approach is comparable to existing solutions.

5.1 Experimental Setup

We implement our solution using Java 1.7, and we ran tests on a server with four virtual CPUs (2.1GHz each) with 32GB of memory. For all experiments, (except Experiment-7), we injected errors uniformly across all records in either the X or Y attributes for an FD $X \rightarrow Y$ (the total number of errors is determined by an error rate e). The errors include updating the values in X or Y to other values in the attribute domain, or replacing a small number of characters to obtain a value outside the attribute domain. We used the greedy initialization described in Section 4.5.1. We use the following default values: $\tau = 0.7$, $\alpha = 0.2$, $\beta = 0.795$, and an 8% error rate.

Datasets. We use two real datasets: (1) The IMDB dataset containing 14 attributes and 1.2 million tuples [2]; and (2) the Books dataset, containing 12 attributes and 3 million tuples [49]. We define two FDs per dataset:

F_{I1}: [actFirstName, actLastName, MovieYr] → MovieGenre
F_{I2}: [actFirstName, MovieName] → ActRole.
F_{B1}: [Author, Year] → Title
F_{B2}: [Title, Publisher] → Author.

5.2 Qualitative Evaluation

We evaluate the accuracy of our repairs using the IMDB dataset and compute the precision and recall as we vary the number of tuples, the error rate, similarity threshold τ, α and β optimization parameters, FD complexity, cooling rate, and error skew. We verify the correctness of the proposed repairs by validating against online sources.

Experiment-1: Accuracy vs. #Tuples. Figure 3 shows precision and recall values as we vary the number of tuples using greedy and random initializations. We observe that greedy initialization provides an average 16.6% and 11.4% improvement in precision and recall, respectively, over random initialization. This re-affirms that carefully choosing a good initial repair influences the final repair quality. During the simulated annealing search, a good initialization repair candidate helps to narrow the search toward candidates that are more likely to satisfy our optimization objectives, and minimize the time checking non-viable candidates. As expected, Figure 3 shows a gradual linear decline in accuracy as the number of tuples increases. However, we are still able to maintain precision and recall values of $91\%^{+}$ and $77\%^{+}$, respectively, across all data sizes. We observe that the linear decline is not as steep compared to Figure 4 where the error rate varies. This can be explained by our observations that for an increasing number of tuples in Figure 3, we see an increase in the number of tuples participating in an error; however, a moderate increase in the number of *distinct* errors leads to reduced precision and recall values.

Experiment-2: Accuracy vs. Error Rate. Figure 4 shows the precision and recall values for random and greedy initializations as we vary the error rate, using the IMDB dataset with 500k tuples. We observe a slow decline in accuracy as the error rate increases. The greedy initialization

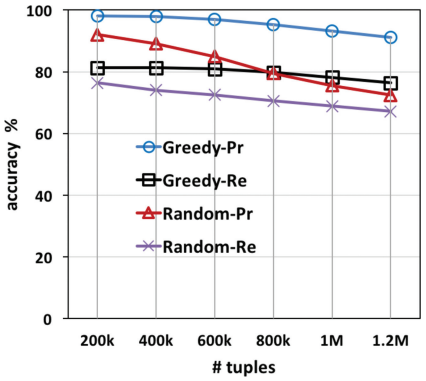

Fig. 3. Accuracy vs. #tuples.

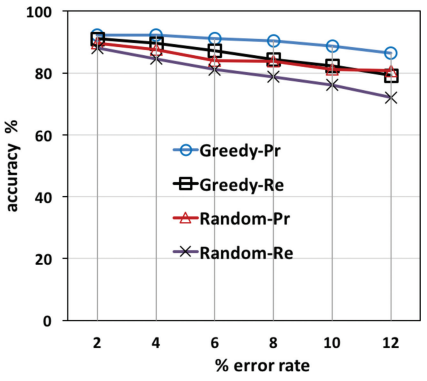

Fig. 4. Accuracy vs. error rate.

precision and recall values are upward from 86% and 76%, respectively, across all error levels. We achieve an average 9% precision and 5% recall improvement over random initialization, with larger gains occurring at the top error rates where a greater number of corrections are needed. As mentioned previously, the rate of decline in accuracy is influenced by the increased number of distinct errors as the error rate increases. Not only is more time spent on addressing these new errors, but the recommended repairs do not correct the increased number of errors, leading to the drop in precision and recall. The improved performance of greedy initialization allows the search to start with better candidates compared to random initialization where it is more difficult to detect all good solutions under a fixed number of iterations.

Experiment-3: Accuracy vs. τ. We investigate the influence of the similarity parameter τ on repair accuracy where higher τ values indicate more stringent matching between records from T and M. Figure 5 shows that as we increase τ from 60-90%, precision remains relatively steady, while recall decreases from 80% to 40%. The drop in recall can be explained by a strict matching condition (as τ increases) that limits the number of matched tuples from M that can correct tuples in T. For example, consider two errors e_1 and e_2 in T that can be fixed by tuples in M when $\tau = 0.6$. Suppose at $\tau = 0.6$, the best matched tuple for e_1 returns a similarity matching of 0.95, while for e_2, there are several matched tuples with similarity scores up to 0.8. If we increase $\tau = 0.9$, e_1 continues to be fixed with its correctly matched tuple (i.e., maintaining the precision value), but e_2 will have no matched tuples, and thus, will not be corrected (i.e., lowering the overall recall scores). Finally, similar to previous results, we observe that the greedy initialization again outperforms random initialization, in this case, by an average of 10%.

Experiment-4: Accuracy vs. α, β. We study the influence of the privacy loss (α) and utility (β) parameters on repair accuracy. Figure 6 and Figure 7 show the precision and recall values as we vary α and β, respectively, using 500K records of the IMDB dataset. In Figure 6, we observe that for $\alpha \in [0.1, 0.5]$, the precision and recall values for `InfoClean` are relatively stable, at approximately 90% and 85%, respectively. However, for $\alpha > 0.5$, we observed that the repair quality declines as the *pvt* value exerts greater influence. At high α levels, the disclosure restrictions limit the number of values that can be corrected. For example, at $\alpha = 0.9$, precision drops to 85% and recall to 34%. We observed that fewer values from M are disclosed, leading to fewer values being fixed in T, thereby causing the significant drop in recall.

Another consequential effect of an increased α value is that smaller solutions are found by the repair algorithm. Smaller solutions refer to candidates that are composed of fewer attribute values

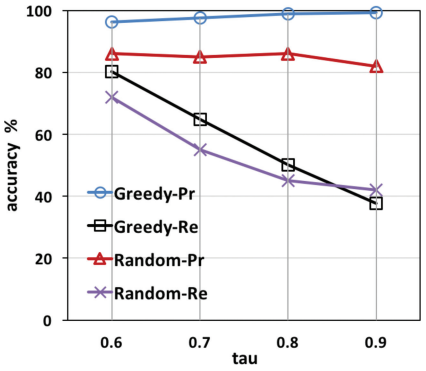

Fig. 5. Accuracy vs. τ.

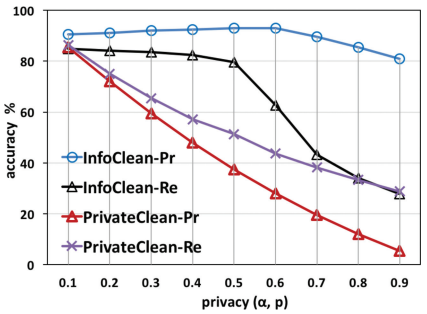

Fig. 6. Quality comparison: InfoClean vs. PrivateClean.

in the repair. These repairs aim to minimize the *pvt* objective by disclosing fewer values from M. Since fewer data values are revealed, there are less clean values available for repair, resulting in an increased number of errors that are not captured, and significantly lowering the recall. Although precision also decreases, it occurs at a slower rate, as the repairs (while involving fewer attributes) are still correct.

We studied the influence of varying β on repair accuracy. We varied β between [0.1, 1], while conversely varying γ between 0.9 to 0, and fixed $\alpha = 0.05$. When $\beta \in [0, 0.98]$, we observed very little change in the precision and recall values. This can be explained by the relative difference in magnitude between the InD and $\#upd$ objectives in our evaluation (over 500K records) that causes the influence of the $\beta \cdot InD$ objective to be very small relative to $\gamma \cdot \#upd$. However, when we increased β beyond 0.98, Figure 7 shows that precision increases to 88% and recall increases to 85%. This leads to more repair values being disclosed by M, leading to more fixes in T, thereby increasing the number of corrected tuples.

Guidelines for α, β tuning: In our evaluation, we achieved the best results for settings of $alpha \leq 0.5$ where the model has the ability to restrict disclosure of some values but has sufficient flexibility to still reveal enough values to achieve good precision and recall results. From our observations, we recommend starting with an $\alpha \leq 0.5$, and to refine further (via tuning) based on the current workload. Once α is determined, the β value can be initialized to a value that is loosely $(1 - \alpha)$, with a small portion allocated toward γ. As our evaluation shows, larger β values (above 0.9) are desirable if the relative magnitude of β is small compared to the other weights.

Experiment-5: Complexity of FDs. We evaluate InfoClean's effectiveness to correct errors for increasingly constrained (complex) FDs. We use the Books dataset with 100K records, and a single FD $X \rightarrow Y$, F_B: `[Author]` $\rightarrow$ `[Title]` to measure the impact of an increasing number of attributes in X, where $|X|$ ranges from [1, 5]. That is, by increasing the number of attributes in the left-hand-side (LHS) X of F_B, we are constraining the dependency (i.e., increasing its complexity). We randomly replace values in X and Y with another value from the domain, where at least 50% of the total errors are in X. We use a 5% error rate.

Figure 8 shows the precision and recall curves both decrease for increasing $|X|$. As $|X|$ increases, the injected errors in X resulted in tuples from T that were matched with the wrong tuples in M due to our approximate similarity matching, and also resulted in more matched tuples from M. This increased space of possible repairs led to a drop in the precision, especially for $|X| \geq 3$. We

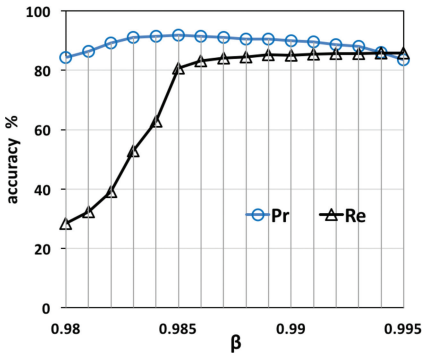

Fig. 7. Accuracy vs. β.

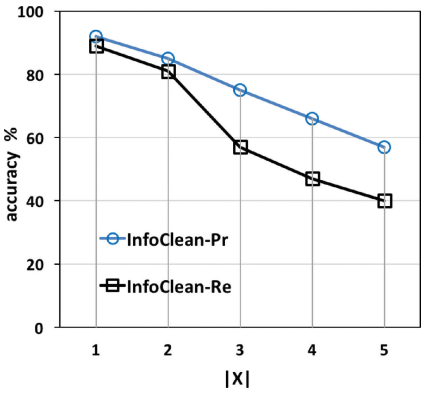

Fig. 8. Accuracy vs. FD complexity.

anticipate that more stringent matching criteria will help to improve precision scores by targeting the right master tuples during the record matching.

The recall scores decrease more dramatically, again for $|X| \geq 3$. Since our errors involve replacing an attribute value with another value in the domain, this created increased duplication in subsets of tuples and less duplication in other tuples. For errors that occurred in lower frequency values (i.e., less duplication), these values presumably are also more rare in M with higher information content, and less likely to be disclosed. In these cases, InfoClean does not disclose the desired value, but seeks another repair value, which is not the ground truth. This leads to unresolved errors, particularly when $|X| \geq 3$, and results in lower recall scores.

Experiment-6: Accuracy vs. η. We study the repair accuracy as we vary the cooling rate parameter $\eta \in [0.75, 0.95]$. Our cooling schedule is a multiplicative monotonic cooling that starts with a temperature K_z at time z, and decreases the temperature by a constant factor η at each iteration until the final, user-defined temperature is achieved (Section 4.5). Figure 9 shows that InfoClean is sensitive to the cooling schedule. Precision and recall scores of 80%+ are achieved for $\eta \geq 0.9$, which represent schedules that decline at slower rates. Repair accuracy drops as the value of η decreases as these schedules with faster rate of temperature decline, lead to less opportunity for the search algorithm to explore new repairs. We intend to investigate other cooling schedules, such as additive linear or quadratic cooling, that lower temperatures at slower rates than our current schedule. In particular, the additive quadratic cooling is non-monotonic and provides opportunities to refine repair solutions at the tail end of the cooling schedule.

Experiment-7: Accuracy vs. error skew. We evaluate InfoClean's sensitivity to data skew when the errors are distributed non-uniformly. For a fixed number of errors, we injected errors according to a Zipfian distribution. That is, the total number of errors is distributed across T_{err}% tuples where $T_{err} = \{20, 40, 60, 80, 100\}$. Figure 10 shows that InfoClean is sensitive to the error distribution. When the errors are concentrated to 20% of the tuples, InfoClean performs poorly, achieving 55% precision, and less than 20% recall. As the number of errors increase per tuple, this significantly distorts the original tuple. This leads to poor quality (and incorrect) matches against the master records and, consequently, the wrong values being selected as repairs. Applying more stringent similarity thresholds (τ) may help improve accuracy in cases where subsets of attribute values remain error-free. Figure 10 shows that as T_{err} increases, and errors are uniformly distributed across all tuples, InfoClean achieves precision and recall rates of $84\%^{+}$.

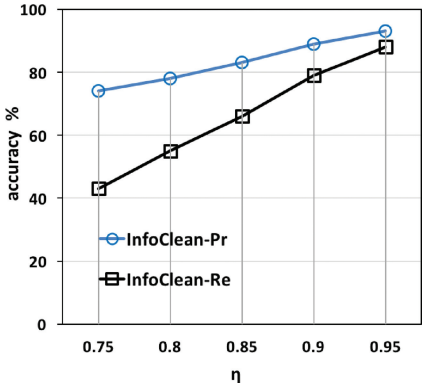

Fig. 9. Accuracy vs. η.

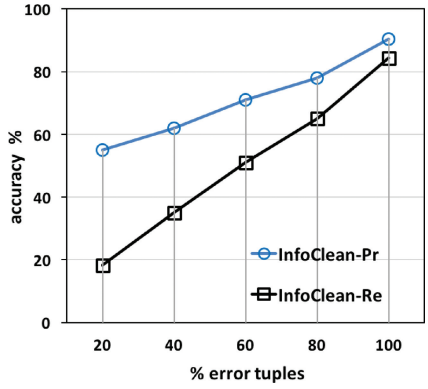

Fig. 10. Varying error skew.

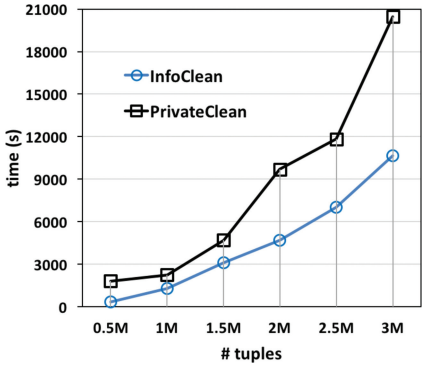

Fig. 11. Runtime: InfoClean vs. PrivateClean.

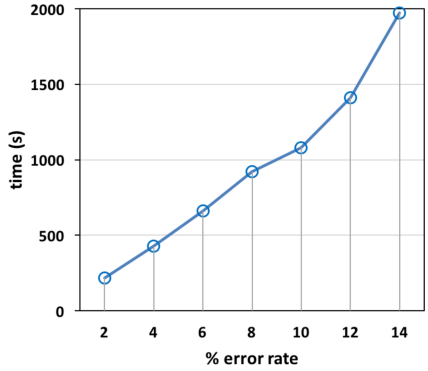

Fig. 12. Time vs. error rate.

5.3 Performance Evaluation

We evaluate the performance of InfoClean as we scale the number of tuples and the error rate.

Experiment-8: Scalability in #Tuples. Figure 11 shows the running times of `InfoClean` as we vary the number of tuples from the Books dataset, using greedy initialization. As expected, we observe a linear scale-up, where the time increases due to: (1) performing record matching for an increased number of errors, and an increased number of tuples participating in an error; (2) invoking the simulated annealing search to find a repair for each (new) error; (3) an increased space of repair candidates that must be navigated in the search space. In addition, as the number of tuples increases, we expect the number of Pareto-optimal solutions to also increase.

Experiment-9: Time vs. Error Rate. We study the performance of our techniques for increasing error rates from 2-14% using 500K tuples of the Books dataset. We distribute errors uniformly across the tuples. Figure 12 shows the running times scale linearly. The increased runtime is due to an increased number of errors that must be corrected, which involves invoking a simulated annealing search for each error, therefore, having a direct impact on the overall running time. In addition, an increased error rate also causes more tuples to possibly participate in an existing error, which can increase the space of candidate repairs that must be evaluated and considered in the final solution set. We note that Figure 12 shows a faster rate of time increase for error rates

greater than 10% that may be due to interactions between errors involving common tuples, since our defined FDs contain two overlapping attributes. We intend to explore a more sophisticated ranking of errors that considers interactions between tuples in future work.

5.4 Comparative Evaluation

We compare our solution against three existing techniques by measuring the repair accuracy, and performance, along two lines: (1) in cases when controlling data privacy is needed; and (2) against data cleaning techniques that do not consider privacy requirements.

5.4.1 Data Cleaning with Privacy. We evaluate `InfoClean` against an existing privacy-aware data cleaning solution, `PrivateClean` [36]. PrivateClean creates differentially private datasets, and relies on user-defined cleaning operators (extract, transform, merge) to correct errors on the privatized relation. We study the comparative quality and performance between the two techniques.

Experiment-10: Comparative Accuracy with PrivateClean. `PrivateClean` assumes error detection and reasoning are performed over privatized relations. The user-defined cleaning operations applied to the privatized relation are then applied to the original (non-privatized) relations to assess quality against the ground truth. Using the IMDB dataset, we generate a set of privatized relations at varying privacy levels by varying the privacy (randomization) parameter, p, which controls the level of perturbation in discrete attributes. We replace values in an attribute with probability p with another value in the domain uniformly at random. Increasing the randomization parameter p increases the privacy level by making rare values more common. This shares a similar definition to α, where increasing values indicate stronger privacy requirements. We set these two parameters to have equal values during our evaluation. We use the defined FDs as a benchmark to identify errors over the privatized relation.

We seek repairs that minimize our objective function and apply these fixes using the extract, transform, and merge operators over the privatized relation. To compute precision and recall, we apply these same fixes to the original (non-privatized) relation to measure against the ground truth. Figure 6 shows the precision and recall values for varying p values. We observe that PrivateClean's repair accuracy is significantly lower than our methods. This can be explained by the fact that detecting, reasoning, and cleaning errors over privatized relations is very difficult, with unclear mappings between the perturbed and original values. This leads to incorrect repairs that do not re-align the FDs and the original data values. While PrivateClean provides bounds on the amount of bias introduced over the privatized relations and the amount of distortion in query results over these relations, the utility of the data is significantly reduced, especially for data cleaning.

In our work, we work directly with the original data values, thereby maintaining the utility of the data and its original mappings to the errors and FDs, but limit the disclosure of highly informative values. At privacy levels above 0.5, we achieve an average 8.25x improvement in precision. We achieve an average 36% improvement in recall, focused on privacy levels ranging from [0.2, 0.6]. InfoClean recall values remain steady in this [0.2, 0.6] privacy range, but experience a faster decline than PrivateClean likely due to the more stringent disclosure restrictions that are exerted by the α parameter compared to the other two (i.e., β and $\#upd$) weights. At higher privacy levels, both techniques converge toward lower recall values (as expected) due to a limited number of values being disclosed to correct errors.

From this study, we believe that proposing privacy-aware repairs via controlling information disclosure is an intuitive approach that can be applied in practice. This allows users to understand

how repairs are generated (by working directly with the original data values) and achieves good (repair) accuracy levels.

Experiment-11: Comparative Performance with PrivateClean. Figure 11 shows the running times between InfoClean and PrivateClean for an increasing number of tuples. In comparing these two different approaches, in both techniques, an increase in the number of tuples leads to an increased number of distinct errors and an increase in the number of tuples participating in an error. This results in longer running times, for both solutions, to identify errors and to determine fixes. In PrivateClean, an overhead is incurred to randomize the data according to the randomization parameter p. However, the main culprit lies in the difficulty to identify, reason, and correct errors over this randomized data. By applying a cardinality-minimal approach for data cleaning, we tried to make as few changes as possible to the data to resolve the errors. However, validating that the errors identified over the perturbed relation corresponded to actual errors in the original relation was extremely time consuming. In fact, a large number of errors were identified due to data skew and unique values that occurred as a result of the randomization. This leads to a large number of identified errors, which increases the search space and search time to determine fixes.

In InfoClean, error detection is performed w.r.t. the set of defined FDs over the original data in linear time. Similarly, quantifying information content in M, and embedding the data values for T and M, both increase linearly w.r.t. the number of tuples. The bulk of the running time is due to the record-matching phase and the repair algorithm, which both have increased running times for an increased number of tuples and errors. For each error in T, the TTP must determine suitable matching tuples from M and carry out a search for a minimal repair. While our results show that InfoClean achieves an average 50% improvement in performance over PrivateClean, the two methods involve different priorities. PrivateClean was designed to provide bounds on the amount of allowed distortion in query results over privatized relations, where small cleaning fixes help to minimize this distortion. In contrast, InfoClean aims to provide a pragmatic data cleaning framework that considers limiting disclosure of sensitive data without dramatically decreasing data utility.

5.4.2 Data Cleaning with No Privacy. We study the comparative quality and performance of InfoClean against two existing (no privacy) data cleaning algorithms: the `Unified Model (UM)` [15] and the `Relative Trust Model (RelTrust)` [8]. The `UM` considers repairs that can modify the data or the FDs. The RelTrust repair algorithm extends the UM algorithm and removes the assumption that data and FD repairs are equally likely [8]. While both UM and RelTrust consider repairs to the data and to the FDs, we disable FD repairing for our evaluation. We implement UM-data that recommends only data repairs via a cardinality minimal data repair algorithm that aims to maximize consistency and minimize the number of updates to the data. The RelTrust algorithm uses a τ_r parameter that specifies the maximum number of allowed data changes, where smaller τ_r values indicate greater trust in the data. To measure the influence of τ_r on repair quality and performance, we implement a variation of RelTrust, called `RelTrustBasic`, that recommends data repairs according to τ_r. We set $\tau_r = 0.8$. In `InfoClean`, we apply greedy initialization and set the similarity threshold $\tau = 0.7$, $\alpha = 0.2$, and $\beta = 0.795$. We use the IMDB dataset with 500K records.

Experiment-12: Comparative Accuracy with the Unified Model and Relative Trust. Figures 13 and 14 show the comparative precision and recall, respectively, against the UM-data and RelTrustBasic techniques. As expected, Figure 13 shows that all techniques exhibit a decline in precision for increasing error rates as an increased number of errors must be fixed. We observe that Infoclean achieves comparable precision to UM-data, with an average −2.8% and −6% lower than UM-data and RelTrustBasic, respectively. This is expected given the disclosure restrictions that InfoClean must satisfy. The results demonstrate that the record matching in InfoClean from

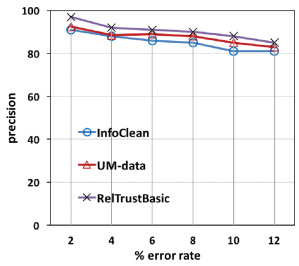

Fig. 13. Comparative precision.

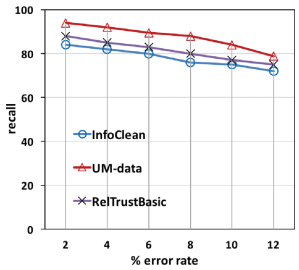

Fig. 14. Comparative recall.

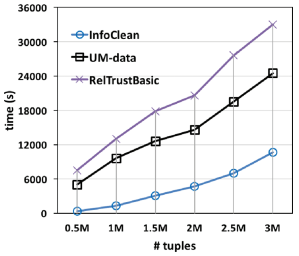

Fig. 15. Comparative runtimes.

a curated master data source helps to achieve comparable precision rates against UM-data and RelTrustBasic, which both rely on identifying repairs from the same source relation by using frequency as an indicator of cleanliness.

Figure 14 shows that UM-data and RelTrustBasic outperform InfoClean by an average 12.2% and 4% in recall, respectively. In InfoClean, repair candidates are generated from tuples in M. Depending on the similarity threshold τ, this may include tuples that are loosely similar, leading to repairs that do not capture all possible errors. Disclosure restrictions limit the values that can be used in a repair and influence InfoClean's recall scores, as some errors may not be corrected. In contrast, UM-data and RelTrustBasic repairs are generated from tuples in the same relational instance. In UM-data, matching is based on strict equality using attributes in the FD, leading to more precise matches and repair values.

Experiment-13: Comparative Performance with the Unified Model and Relative Trust. Figure 15 shows the comparative runtimes between InfoClean, UM-data, and RelTrustBasic with only data repairs, as we scale the number of tuples. InfoClean outperforms UM-data and RelTrustBasic by an average 54% and 62%, respectively. This can be explained by InfoClean's reduced search space, where (single) repairs satisfying the objective function are iteratively added to the solution set. In contrast, UM-data and RelTrustBasic adopt cardinality minimal repair approaches involving an expanded search space, and seek solutions with the fewest number (or τ_r bounded number) of data repairs.

6 RELATED WORK

Data Cleaning. There has been a rich source of work done in data cleaning, including approaches that use data dependencies to identify errors and recommend repairs by minimizing a cost function or the number of updates [7, 10, 14–16], methods that interact with the user to learn repair preferences [46, 48], and statistical-based approaches for outlier detection [6, 24, 38]. Extensions have included considering external sources such as master data and knowledge bases to improve repair accuracy [19, 26], holistic approaches that support multiple types of data quality rules [18, 23, 28], and crowd-based data cleaning solutions [19]. All these methods assume that the data is of equal information value, rely on open access to the data, and do not consider varying data sensitivities among the values. Recently, Krishnan et al. [36] proposed `PrivateClean`, a framework that generates differentially private relations, and a set of user-defined cleaning operators that work on top of privatized data. PrivateClean assumes that error detection, reasoning, and cleaning are all performed over the privatized relation. This is inherently difficult to do since the original values are randomized and discovering errors over perturbed data is hard. Errors identified over randomized data do not correspond to errors over the original data, and identifying the mappings is time consuming in practice. Furthermore, PrivateClean proposes generic (user-defined) data cleaning

operators (extract, transform, merge) and does not tackle the problem of finding the specific data repairs to correct errors. The data cleaning and data privacy requirements are decoupled into two separate tasks. While differential privacy provides provable privacy guarantees, these solutions are often hard to implement in practice. In InfoClean, we present a solution that can be used in practice that embeds data sensitivity into the data cleaning process such that the search for data repairs considers the information content of a proposed repair value. As future work, we intend to explore extensions to other privacy models such as k-anonymity and l-diversity.

Data Privacy. Incorporating data privacy alongside data management tasks has been well-studied, including areas such as privacy preserving data publishing [27, 35, 39], data mining [3], schema and information integration [20, 44], record linkage [30], and data quality assessment [5].

Barone et al. [5] propose a model that considers privacy requirements for improving data quality. Their model focuses on the assessment of data accuracy and completeness for a relation, and verifying these properties during the data quality management process. Their solution does not consider how errors are identified nor propose the specific data updates required to correct the errors. In similar spirit to our work, Scannapieco et al. [44] consider privacy preserving methods for schema and data matching by utilizing metric embeddings to ensure secure information exchange. In recent work, we explored interactive privacy-aware data cleaning applications [32] and extended repair operations to protect data privacy [33]. In future work, we intend to explore models that remove the TTP and the metric embeddings by incorporating k-anonymity and l-diversity models.

7 CONCLUSION AND FUTURE WORK

In this article, we propose *InfoClean*, a data cleaning framework to clean a target dataset by using information from a master dataset. Our framework facilitates the cooperation between these two datasets, and we present metrics that measure privacy loss and data utility. We develop a multi-objective optimization repair algorithm that generates the best repair solution(s) so that the amount of information disclosed by the master dataset is minimized, the data utility of the target dataset is maximized, and the number of updates to the target is minimized. We envision that our framework can be applied in practice where data disclosure is restricted during the data cleaning process. We are exploring several avenues of future work. We will explore extending the framework to include other privacy models such as k-anonymity, a broader set of dependencies, more extensive repair operations, and exploring more efficient search strategies.

REFERENCES

[1] Lynn Schriml and Elvira Mitraka. 2017. Disease ontology. Retrieved from http://disease-ontology.org.
[2] 2017. IMDB Dataset. Retrieved from http://www.imdb.com/interfaces.
[3] Charu C. Aggarwal and Philip S. Yu. 2008. *Privacy-Preserving Data Mining: Models and Algorithms* (1st ed.). Springer.
[4] Marcelo Arenas and Leonid Libkin. 2005. An information-theoretic approach to normal forms for relational and XML data. *J. ACM* 52, 2 (2005), 246–283.
[5] D. Barone, A. Maurino, F. Stella, and C. Batini. 2009. A privacy preserving framework for accuracy and completeness quality assessment. In *Emerging Paradigms in Informatics, Systems and Communication* (2009), 83–87.
[6] L. Berti-Equille, T. Dasu, and D. Srivastava. 2011. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *ICDE*. 733–744.
[7] George Beskales, Ihab F. Ilyas, and Lukasz Golab. 2010. Sampling the repairs of functional dependency violations under hard constraints. In *PVLDB*. 197–207.
[8] G. Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*. 541–552.
[9] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *ICDE*. 746–755.
[10] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*. 143–154.

[11] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.
[12] J. Branke, K. Deb, K. Miettinen, and R. Slowinski. 2008. *Multiobjective Optimization*. Springer, Lecture Notes in Computer Science.
[13] Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. 2014. Descriptive and prescriptive data cleaning. In *SIGMOD*. 445–456.
[14] F. Chiang and R. J. Miller. 2011. Active repair of data quality rules. In *Proceedings of the International Conference on Information Quality*. 174–188.
[15] F. Chiang and R. J. Miller. 2011. A unified model for data and constraint repair. In *ICDE*. 446–457.
[16] Fei Chiang and Siddharth Sitaramachandran. 2016. Unifying data and constraint repairs. *Journal of Data and Information Quality* 7, 3 (2016), 9:1–9:26.
[17] Fei Chiang and Yu Wang. 2014. Repairing integrity rules for improved data quality. *International Journal of Information Quality* 3, 4 (2014), 273–297.
[18] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. 458–469.
[19] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*. 1247–1261.
[20] Chris Clifton, Murat Kantarcioğlu, AnHai Doan, Gunther Schadow, Jaideep Vaidya, Ahmed Elmagarmid, and Dan Suciu. 2004. Privacy-preserving data integration and sharing. In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'04)*. 19–26.
[21] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory*. Wiley-Interscience.
[22] M. M. Dalkilic and E. L. Robertson. 2000. Information dependencies. In *Principles of Database Systems (PODS)*. 245–253.
[23] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: A commodity data cleaning system. In *SIGMOD*. 541–552.
[24] T. Dasu and J. M. Loh. 2012. Statistical distortion: Consequences of data cleaning. *PVLDB* 5, 11 (2012), 1674–1683.
[25] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *Proc. VLDB Endow.* 2, 1, 407–418.
[26] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards certain fixes with editing rules and master data. *PVLDB* 3, 1 (2010), 173–184.
[27] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. 2010. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.* 42, 4 (June 2010), 14:1–14:53.
[28] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. 2013. The LLUNATIC data-cleaning framework. *PVLDB* 6, 9 (2013), 625–636.
[29] Stuart Geman and Donald Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, 6 (Nov. 1984), 721–741.
[30] Rob Hall and Stephen E. Fienberg. 2010. Privacy-preserving record linkage. In *Privacy in Statistical Databases (PSD)*. 269–283.
[31] Gísli R. Hjaltason and Hanan Samet. 2003. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 5 (2003), 530–549.
[32] Dejun Huang, Dhruv Gairola, Yu Huang, Zheng Zheng, and Fei Chiang. 2016. PARC: Privacy-aware data cleaning. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM)*. 2433–2436.
[33] Yu Huang and Fei Chiang. 2015. Towards a unified framework for data cleaning and data privacy. In *Web Information Systems Engineering (WISE)*. 359–365.
[34] Bilal Hussain, Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. 2013. *An Evaluation of Clustering Algorithms in Duplicate Detection*. Technical Report CSRG-620. Dept. of Computer Science, University of Toronto.
[35] Daniel Kifer and Johannes Gehrke. 2006. Injecting utility into anonymized datasets. In *SIGMOD*. 217–228.
[36] Sanjay Krishnan, Jiannan Wang, Michael J. Franklin, Ken Goldberg, and Tim Kraska. 2016. PrivateClean: Data cleaning and differential privacy. In *SIGMOD*. 937–951.
[37] Ninghui Li and Tiancheng Li. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*. 106–115.
[38] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J. Miller, and Divesh Srivastava. 2015. Combining quantitative and logical data cleaning. *Proceedings of the VLDB Endowment* 9, 4 (2015), 300–311.
[39] Vibhor Rastogi, Dan Suciu, and Sungho Hong. 2007. The boundary between privacy and utility in data publishing. In *VLDB*. 531–542.
[40] Thomas Redman. 2016. Bad data costs the U.S. $3 trillion per year. *Harvard Business Review* (2016).
[41] Stuart J. Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach* (2nd ed.).
[42] L. V. S. Lakshmanan and S. Kolahi. 2009. On approximating optimum repairs for functional dependency violations. In *ICDT*. 53–62.
[43] L. Sankar, S. R. Rajagopalan, and H. V. Poor. 2010. An information-theoretic approach to privacy. In *Allerton Conference on Communication, Control, and Computing*. 1220–1227.

[44] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K. Elmagarmid. 2007. Privacy preserving schema and data matching. In *SIGMOD*. 653–664.
[45] L. Sweeney. 2002. k-anonymity: A model for protecting privacy. *Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (2002), 557–570.
[46] M. Volkovs, F. Chiang, J. Szchilta, and R. J. Miller. 2014. Continuous data cleaning. In *ICDE*. 244–255.
[47] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. 2015. Data X-ray: A diagnostic tool for data errors. In *SIGMOD*. 1231–1245.
[48] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *PVLDB* 4, 5 (2011), 279–289.
[49] C. Ziegler. 2017. Book Crossing Dataset. Retrieved from http://www2.informatik.uni-freiburg.de/cziegler/BX/.

Received June 2017; revised February 2018; accepted February 2018