A Formal Framework for Probabilistic Unclean Databases

Christopher De Sa
Cornell University, Ithacan, NY, USA
cdesa@cs.cornell.edu

Ihab F. Ilyas
University of Waterloo, Waterloo, ON, Canada
ilyas@uwaterloo.ca

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel
bennyk@cs.technion.ac.il

Christopher Ré
Stanford University, Stanford, CA, USA
chrismre@cs.stanford.edu

Theodoros Rekatsinas
University of Wisconsin - Madison, Madison, WI, USA
thodrek@cs.wisc.edu

Abstract

Most theoretical frameworks that focus on data errors and inconsistencies follow logic-based reasoning. Yet, practical data cleaning tools need to incorporate statistical reasoning to be effective in real-world data cleaning tasks. Motivated by empirical successes, we propose a formal framework for unclean databases, where two types of statistical knowledge are incorporated: The first represents a belief of how intended (clean) data is generated, and the second represents a belief of how noise is introduced in the actual observed database. To capture this noisy channel model, we introduce the concept of a Probabilistic Unclean Database (PUD), a triple that consists of a probabilistic database that we call the *intention*, a probabilistic data transformator that we call the *realization* and captures how noise is introduced, and an observed unclean database that we call the *observation*. We define three computational problems in the PUD framework: cleaning (infer the most probable intended database, given a PUD), probabilistic query answering (compute the probability of an answer tuple over the unclean observed database), and learning (estimate the most likely intention and realization models of a PUD, given examples as training data). We illustrate the PUD framework on concrete representations of the intention and realization, show that they generalize traditional concepts of repairs such as cardinality and value repairs, draw connections to consistent query answering, and prove tractability results. We further show that parameters can be learned in some practical instantiations, and in fact, prove that under certain conditions we can learn a PUD directly from a single dirty database without any need for clean examples.

2012 ACM Subject Classification Theory of computation → Data modeling; Theory of computation → Incomplete, inconsistent, and uncertain databases

Keywords and phrases Unclean databases, data cleaning, probabilistic databases, noisy channel

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.6

Related Version `https://arxiv.org/abs/1801.06750`

Funding *Ihab F. Ilyas*: This work was supported by NSERC under a Discovery Grant.
Benny Kimelfeld: This work was supported by the Israel Science Foundation (ISF) Grant 1295/15.
Theodoros Rekatsinas: This work was supported by the Wisconsin Alumni Association, Amazon under an ARA Award, and by NSF under grant IIS-1755676.

1 Introduction

Managing errors and inconsistency in databases is traditionally viewed as a challenge of a logical nature. It is typical that errors in a database are defined with respect to *integrity constraints* that capture normative aspects of downstream applications. The aim of integrity constraints is to guarantee the consistency of data used by these applications. Typically, an unclean database is defined as a database J that violates the underlying set of integrity constraints. In turn, a *repair* of a database J is a clean database I wherein all integrity constraints hold, and is obtained from J by a set of operations (e.g., deletions of tuples or updates of tuple values) that feature some form of non-redundancy [4, 2].

Various computational problems around unclean databases have been investigated in prior work [4, 28, 32, 29]. Past theoretical research has established fundamental results that concentrate on tractability boundaries for repair-checking and consistent query answering [2, 15, 26]. In their majority, these works adopt a deterministic interpretation of data repairs and cast all repairs equally likely. These theoretical developments have inspired practical tools that aim to automate data cleaning [7, 43, 12, 42, 20]. The majority of proposed methods assume as input a set of integrity constraints and use those to identify possible repairs via search-based procedures. To prioritize across possible repairs during search, the proposed methods rely on the notion of *minimality* [11, 23, 30]. Informally, minimality states that given two candidate sets of repairs, the one with fewer changes with respect to the original database is preferable. The use of minimality as an operational principle to find data repairs is a practical artifact that is used to limit the search space. These approaches to data cleaning suffer from two major drawbacks: First, they do not permit concrete statements about the "likelihood" of possible repairs. Consequently, they categorize query answers to a limited set of validity labels (e.g., certain, possible, and impossible); these labels might be unsuitable for downstream applications. Second, combinatorial principles such as minimality, while desired, do not entail the richness of the arguments and evidences (e.g., statistical features of data) that are needed to reason about and generate correct repairs.

Effective data cleaning needs to incorporate statistical reasoning. Our recent work on HoloClean [34] casts data repairing as a statistical learning and inference problem and reasons about a *most probable* repair instead of a *minimal* repair. Our study shows that HoloClean obtains more accurate data cleaning results than competing minimality-based data cleaning tools for a diverse array of real-world data cleaning scenarios [34]. HoloClean uses training data to learn a probabilistic model for how clean data is generated and how data errors are injected. HoloClean's model follows the *noisy channel model* [22], the de-facto probabilistic framework used in natural language tasks, such as spell checking and speech recognition, to reason about noisy data. To the best of our knowledge, existing theoretical frameworks for data cleaning do not capture this type of probabilistic reasoning.

Goals. We aim to establish a formal framework for *probabilistic unclean databases* (PUD) that adopts a statistical view of database cleaning. We do so by following the aforementioned noisy channel paradigm of HoloClean. Within the PUD framework, we formalize fundamental computational problems: *cleaning*, *query answering*, and *learning*. With that, we aim to draw connections between theoretical database research and important aspects of practical systems. In particular, our goal is to open the way for analyses and algorithms with theoretical guarantees for such systems. We argue that our framework is basic enough to allow for nontrivial theoretical advances, as illustrated by our preliminary results that *(a)* draw connections to traditional deterministic concepts, *and (b)* devise algorithms for special cases.

Probabilistic unclean databases. We view an unclean database as if a clean database I had been "distorted" via a noisy channel into a dirty database J; we aim to establish a model of this channel. Given the observed unclean database J, we seek the true database I from which J is produced. This model adopts Bayesian inference: out of all possible I, we seek the one for which the probability, given J, is highest. Following Bayes' rule, our objective is to find $\arg\max_I \Pr(I) \cdot \Pr(J|I)$. This objective decomposes in two parts: (1) the *prior* model for a clean database captured by $\Pr(I)$, and (2) the *channel* or *error* model characterized by $\Pr(J|I)$. To capture that, we define a *Probabilistic Unclean Database* (*PUD*) as a triple $(\mathcal{I}, \mathcal{R}, J^\star)$ where: (1) $\mathcal{I}$, referred to as the *intention model*, is a distribution that produces intended clean databases; (2) $\mathcal{R}$, referred to as the *realization model*, is a function that maps each clean database I to a distribution $\mathcal{R}_I$ that defines how noise is introduced into I; and (3) $J^\star$ is an observed unclean database. The distribution $\mathcal{I}$ defines the prior $\Pr(I)$ over clean databases, while the distribution $\mathcal{R}_I$ defines the aforementioned noisy channel $\Pr(J|I)$.

Computational problems. We define and study three computational problems in the PUD framework: (1) *data cleaning*, where given a PUD $(\mathcal{I}, \mathcal{R}, J)$, we seek to compute a database I that maximizes the probability $\mathcal{I}(I) \times \mathcal{R}_I(J)$; (2) *probabilistic query answering*, that is, the problem of evaluating a query Q over a PUD following the traditional possible tuple semantics [13, 40]; and (3) *learning* a PUD, where we consider *parametric* representations $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ of the intention and realization models, and seek to estimate the parameter vectors Ξ^* and Θ^* that maximize the likelihood of training data.

Preliminary analysis. PUDs allow for different instantiations of the intention and realization models. To establish preliminary complexity and convergence results, we focus on specific instantiations of the intention and realization models. We study intention models that can describe the distribution of tuple values as well as both soft and hard integrity constraints. We also focus on simple noise models. We study (1) realizations that introduce new tuples, hence, the clean database is a subset of the observed unclean database, and (2) realizations that update table cells, hence, the clean database is obtained via value repairs over the observed unclean database.

We present PUD instantiations for which solving the data cleaning problem has polynomial-time complexity. For instance, we show that in the presence of only one key constraint, soft or hard, data cleaning in PUDs can be solved in polynomial time. This result extends results for deterministic repairs that focus on hard integrity constraints to *weak* (soft) key constraints (e.g., two people are unlikely to, but might, have the same first and last name). Here, the most probable repair under the PUD framework may violate weak key constraints. We also draw connections between data cleaning in the PUD framework and *minimal repairs*. We identify conditions under which data cleaning in the PUD framework is equivalent to *cardinality repairs* [32] and *optimal V-repairs* [23]. For PUD learning, we consider both *supervised* and *unsupervised* learning. In the former case, we are given intension-realization pairs, and in the former, we are given only realizations (i.e., dirty databases). Our results discuss convexity and gradient computation for the optimization problem underlying the learning problem.

Our PUD model can be viewed as a generalization of the approach of Gribkoff et al. [19], who view the dirty database as a tuple-independent probabilistic database [13], and seek the *most-probable database* that satisfies a set of underlying integrity constraints (e.g., functional dependencies). In contrast, our modeling allows for arbitrary distributions over the intention, including ones with *weak* constraints that we discuss later on. Interestingly, our PUD model

goes in the reverse direction of the *operational* approach of Calautti et al. [10], who view the dirty database as a deterministic object and its *cleaning* (rather than the *error*) as a probabilistic process (namely a Markov chain of repairing operations).

Vision. This paper falls within the bigger vision of bridging database theory with learning theory as outlined in a recent position article [1]. We aim to draw connections between the rich theory on inconsistency management by the database community, and fundamentals of statistical learning theory with emphasis on *structured prediction* [5]. Structured prediction typically focuses on problems where, given a collection of observations, one seeks to predict the most likely assignment of values to structured objects. In most practical structured prediction problems, structure is encoded via logic-based constraints [18] in a way similar to how consistency is enforced in data cleaning. It is our hope that this paper will commence a line of work towards theoretical developments that take the benefit of both worlds, and will lead to new techniques that are both practical and rooted in strong foundations.

Organization. We begin with preliminary definitions in Section 2. In Section 3 we present the concept of PUDs. We present the three fundamental computational problems in Section 4, and describe preliminary results in Sections 5 and 6. We conclude with a discussion in Section 7. For space limitations, all proofs are in the extended version of our paper [36].

2 Preliminaries

We first introduce concepts, definitions and notation that we need throughout the paper.

Schemas and databases. A *relation signature* is a sequence $\alpha = (A_1, \ldots, A_k)$ of distinct *attributes* A_i, where k is the *arity* of α. A (*relational*) *schema* $\mathbf{S}$ has a finite set of *relation symbols*, and it associates each relation symbol R with a signature that we denote by $sig_{\mathbf{S}}(R)$, or just $sig(R)$ if $\mathbf{S}$ is clear from the context. We assume an infinite domain Const of *constants*. Let $\mathbf{S}$ be a schema, and let R be a relation symbol of $\mathbf{S}$. A *tuple* t over R is a sequence $(c_1, \ldots, c_k)$ of constants, where k is the arity of $sig(R)$. If $t = (c_1, \ldots, c_k)$ is a tuple over R and $sig(R) = (A_1, \ldots, A_k)$, then we refer to the value c_j as $t.A_j$ (where $j = 1, \ldots, k$). We denote by $\mathrm{tuples}(R)$ the set of all tuples over R.

In our databases, tuples have unique record identifiers. Formally, a table r over R is associated with a finite set $ids(r)$ of identifiers, and it maps each identifier i to a tuple $r[i]$ over R. A *database* I over $\mathbf{S}$ consists of a table R^I over each relation symbol R of $\mathbf{S}$, such that no two occurrences of tuples have the same identifier; that is, if R_1 and R_2 are distinct relation symbols in $\mathbf{S}$, then $ids(R_1^I)$ and $ids(R_2^I)$ are disjoint sets. We denote by $ids(I)$ the union of the sets $ids(R^I)$ over all relation symbols R of $\mathbf{S}$. If $i \in ids(R^I)$, then we may refer to the tuple $R^I[i]$ simply as $I[i]$.

A *cell* of a database I is a pair (i, A), where $i \in ids(R^I)$ for a relation symbol R, and A is an attribute inside $sig(R)$. We denote the cell (i, A) also by $i.A$, and we denote by $cells(I)$ the set of all cells of I.

Let I and J be databases over the same schema $\mathbf{S}$. We say that I is a *subset* of J if I can be obtained from J by deleting tuples, that is, $ids(R^I) \subseteq ids(R^J)$ for all relation symbols R of $\mathbf{S}$ (hence, $ids(I) \subseteq ids(J)$) and $I[i] = J[i]$ for all $i \in ids(I)$. We say that I is an *update* of J if I can be obtained from J by changing attribute values, that is, $ids(R^J) = ids(R^I)$ for all relation symbols R of $\mathbf{S}$.

A *query* Q over a schema $\mathbf{S}$ is associated with fixed arity, and it maps every database D over $\mathbf{S}$ into a finite set $Q(D)$ of tuples of constants over the fixed arity.

Integrity constraints. Various types of logical conditions are used for declaring integrity constraints, including *Functional Dependencies* (FDs), *conditional FDs* [7], *Denial Constraints* (DCs) [17], referential constraints [14], and so on. In this paper, by *integrity constraint* over a schema $\mathbf{S}$ we refer to a general expression φ of the form $\forall x_1, \ldots, x_m[\gamma(x_1, \ldots, x_m)]$, where $\gamma(x_1, \ldots, x_m)$ is a safe expression in Tuple Relational Calculus (TRC) over $\mathbf{S}$. For example, an FD $R : A \to B$ is expressed here as the integrity constraint

$$\forall x, y\, [(x \in R y \in R) \to (x.A = y.A \to x.B = y.B)]\ .$$

A *violation* of $\varphi = \forall x_1, \ldots, x_m[\gamma(x_1, \ldots, x_m)]$ in the database I is a sequence $i_1, \ldots, i_m$ of tuple identifiers in $ids(I)$ such that I violates $\gamma(I[i_1], \ldots, I[i_m])$, and we denote by $V(\varphi, I)$ the set of violations of φ in I. We say that I *satisfies* φ if I has no violations of φ, that is, $V(\varphi, I)$ is empty. Finally, I *satisfies* a set Φ of integrity constraints if I satisfies every integrity constraint φ in Φ.

Minimum repairs. Traditionally, database *repairs* are defined over inconsistent databases, where *inconsistencies* are manifested as violations of integrity constraints. A repair is a consistent database that is obtained from the inconsistent one by applying a *minimal* change, and we recall two types of repairs: *subset* (obtained by deleting tuples) and *update* (obtained by changing values). Moreover, the repairing operations may be weighted by tuple weights (in the first case) and cell weights (in the second case).

Formally, let $\mathbf{S}$ be a schema, Φ a set of integrity constraints over $\mathbf{S}$, and J a database that does not necessarily satisfy Φ. A *consistent subset* (resp., *consistent update*) of J is a subset (resp., update) I of J such that I satisfies Φ. A *minimum subset repair* of J w.r.t. a weight function $w : ids(J) \to [0, \infty)$ is a consistent subset I of J that minimizes the sum $\sum_{i \in ids(J) \setminus ids(I)} w(i)$. As a special case, a *cardinality repair* of J is a minimum subset repair w.r.t. a constant weight (e.g., $w(i) = 1$), that is, a consistent subset with a maximal number of tuples. A *minimum update repair* of J w.r.t. a weight function $w : cells(J) \times \mathsf{Const} \to [0, \infty)$ is a consistent update I of J that minimizes the sum $\sum_{i.A \in cells(I)} w(i.A, I[i].A)$.

Probabilistic databases. A *probabilisitic database* is a probability distribution over ordinary databases. As a representation system, our model is a generalization of the *Tuple-Independent probabilistic Database* (TID) wherein each tuple might either exist (with an associated probability) or not [13, 40]. In our model, each tuple comes from a general probability distribution over tuples (where inexistence is one of the options). This allows us to incorporate beliefs about the likelihood of tuples and cell values.

We now give the formal definition. Let $\mathbf{S}$ be a schema. A *generalized TID* is a database $\mathcal{K}$ that is defined similarly to an ordinary database over $\mathbf{S}$, except that instead of a tuple, the entry $R^{\mathcal{K}}[i]$ is a discrete probability distribution over the set $tuples(R) \cup \{\bot\}$, where the special value $\bot$ denotes that no tuple is generated. Hence, for every tuple t over R, the probability that $R^{\mathcal{K}}[i]$ produces t is given by $R^{\mathcal{K}}[i](t)$, or just $\mathcal{K}[i](t)$; moreover, the number $\mathcal{K}[i](\bot)$ is the probability that no tuple is generated for the identifier i. Therefore, $\mathcal{K}$ defines a probability distribution over databases I over $\mathbf{S}$ such that $ids(I) \subseteq ids(\mathcal{K})$ and the probability $\mathcal{K}(I)$ of a database I is defined as follows:

$$\mathcal{K}(I) \stackrel{\text{def}}{=} \prod_{i \in ids(I)} \mathcal{K}[i](I[i]) \times \prod_{i \in ids(\mathcal{K}) \setminus ids(I)} \mathcal{K}[i](\bot)$$

Table 1 Main symbols used in the framework.

$\mathbf{S}$	A schema.
$\mathcal{U}$	A PUD $(\mathcal{I}, \mathcal{R}, J^\star)$.
$\mathcal{I}$	An intention model (probabilistic database).
$\mathcal{R}$	A realization model, maps every I into a probabilistic database $\mathcal{R}_I$.
$J^\star$	An observed unclean database.
$\mathcal{R}\circ\mathcal{I}$	Distribution over pairs (I, J) given by $\mathcal{R}\circ\mathcal{I}\,(I, J) = \mathcal{I}(I) \cdot \mathcal{R}_I(J)$.
$\mathcal{U}^\star$	A probabilistic database given by $\mathcal{U}^\star(I') = \Pr_{(I,J)\sim\mathcal{R}\circ\mathcal{I}}(I = I' \mid J = J^\star)$.
$(\mathcal{D}, \tau, J^\star)$	A parfactor/subset PUD.
$(\mathcal{D}, \kappa, J^\star)$	A parfactor/update PUD.
$\mathcal{D}$	A parfactor database $(\mathcal{K}, \Phi, w)$ with $w : \Phi \to (0, \infty)$.
$\mathcal{K}$	A generalized tuple-independent database (generalized TID).
Φ	A set of integrity constraints φ.
τ	Maps $i \in \mathit{ids}(R^{\mathcal{D}})$ to a discrete distribution $\tau[i]$ over tuples$(R) \cup \{\bot\}$.
κ	Maps $(i, t) \in \mathit{ids}(R^{\mathcal{D}}) \times$ tuples(R) to a discrete distribution $\kappa[i, t]$ over tuples(R).

We incorporate weak integrity constraints by adopting the standard concept of *parametric factors* (or *parfactors* for short), which has been used in the *soft keys* of Jha et al. [21] and the *PrDB* model of Sen et al. [38], and which can be viewed as a special case of the *Markov Logic Network* (MLN) [35]. Under this concept, each constraint φ is associated with a weight $w(\varphi) > 0$ and each violation of φ contributes a factor of $\exp(-w(\varphi))$ to the probability of a random database I. Formally, a *parfactor database* over a schema $\mathbf{S}$ is a triple $\mathcal{D} = (\mathcal{K}, \Phi, w)$, where $\mathcal{K}$ is a generalized TID, Φ is a finite set of integrity constraints, both over $\mathbf{S}$, and $w : \Phi \to (0, \infty)$ is a weight function over Φ. The probability $\mathcal{D}(I)$ of a database I is defined as follows.

$$\mathcal{D}(I) \stackrel{\text{def}}{=} \frac{1}{Z} \times \mathcal{K}(I) \times \exp\left(-\sum_{\varphi\in\Phi} w(\varphi) \times |V(\varphi, I)|\right)$$

Recall that $V(\varphi, I)$ the set of violations of φ in I. The number Z is a *normalization factor* (also called the *partition function*) that normalizes the sum of probabilities to one:

$$Z \stackrel{\text{def}}{=} \sum_I \mathcal{K}(I) \times \exp\left(-\sum_{\varphi\in\Phi} w(\varphi) \times |V(\varphi, I)|\right)$$

Observe that the above sum is over a countable domain, since we assume that every $R^{\mathcal{K}}[i]$ is discrete (hence, there are countably many random databases I). Since we normalize the probability, it is not really necessarily for $\mathcal{K}$ to be normalized, as $\mathcal{D}$ would be a probability distribution even if $\mathcal{K}$ is *not* normalized. In fact, in our analysis, we will *not* make the assumption that $\mathcal{K}$ is normalized.

3 Probabilistic Unclean Databases

We introduce the Probabilistic Unclean Database (PUD) framework and describe examples of PUD instantiations that correspond to data cleaning applications in the HoloClean system [34]. In our framework, a PUD consists of three components following a noisy-channel model: *(1)* an *intention* model for generating clean databases, *(2)* a noisy *realization* model that can distort the intended clean database, *and (3)* an observed unclean database. The formal definition follows.

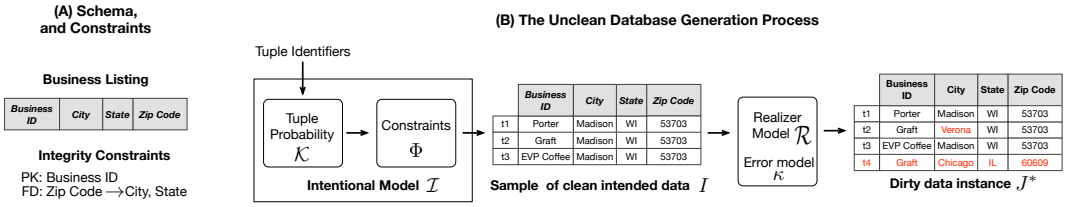

Figure 1 Overview of the PUD framework.

▶ **Definition 1.** *Let* $\mathbf{S}$ *be a schema. A PUD (over* $\mathbf{S}$*) is a triple* $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ *where:*

1. $\mathcal{I}$ *is a probabilistic database, referred to as the* intention *model;*
2. $\mathcal{R}$*, referred to as the* realization *model, is a function that maps each database* I *to a probabilistic database* $\mathcal{R}_I$*;*
3. $J^\star$ *is a database referred to as the* observed *or* unclean *database.*

▶ **Example 2.** Figure 1 illustrates a high-level example of the PUD framework. We use a running example from business listings. Figure 1(A) depicts the schema $\mathbf{S}$ of the example. The constraints include a primary key and a functional dependency. Figure 1(B) depicts the unclean database generation process. Intention $\mathcal{I}$ outputs a valid database I with three tuples. The realizer $\mathcal{R}$ takes as input this database I, injects the new tuple $t4$ and updates the `City` value of tuple $t2$ from "Madison" to "Verona."

A PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ defines a probability distribution, denoted $\mathcal{R} \circ \mathcal{I}$, over pairs (I, J). Conditioning on $J = J^\star$, the PUD $\mathcal{U}$ also defines a probability distribution, denoted $\mathcal{U}^\star$, over intentions I (i.e., a probabilistic database). In the generative process of $\mathcal{R} \circ \mathcal{I}$, we sample the intention I from $\mathcal{I}$, and then we sample J from the realization $\mathcal{R}_I$. Hence, the probability of (I, J) is given by

$$\mathcal{R} \circ \mathcal{I}\,(I, J) \stackrel{\text{def}}{=} \mathcal{I}(I) \cdot \mathcal{R}_I(J)\,.$$

In the probabilistic database $\mathcal{U}^\star$, the probability of each candidate intention I' is given by

$$\mathcal{U}^\star(I') \stackrel{\text{def}}{=} \Pr_{(I,J)\sim\mathcal{R}\circ\mathcal{I}}(I = I' \mid J = J^\star) = \frac{\mathcal{R} \circ \mathcal{I}\,(I', J^\star)}{\sum_I \mathcal{R} \circ \mathcal{I}\,(I, J^\star)}$$

that is, the probability conditioned on the random J being $J^\star$. For this distribution to be well defined, we require $J^\star$ to have a nonzero probability; that is, there exists I such that $\mathcal{R} \circ \mathcal{I}\,(I, J^\star) > 0$. Table 1 lists the main symbols in the framework, along with their meaning.

3.1 Example Instantiations of PUDs

Our definition of a PUD is abstract, and not associated with any specific representation model. We now present concrete instantiations of PUD representations. These instantiations are probabilistic generalizations of the (deterministic) concepts of *subset repairs* [32, 2] and *update repair* [23, 30], respectively. More precisely, in both instantiations, the PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ is such that $\mathcal{I}$ is represented as a parfactor database $\mathcal{D}$ (as defined in Section 2) and $J^\star$ is an ordinary database (as expected); the two differ in the representation of the realization model $\mathcal{R}$. In the first instantiation, $\mathcal{R}$ is allowed to introduce new random tuples (hence, the intended database is a *subset* of the unclean one) and in the second, $\mathcal{R}$ is allowed to randomly change tuples (hence, the intended database is an *update* of the unclean one). Formally, let $\mathbf{S}$ be a schema.

Intended Database

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53703
EVP Coffee	Madison	WI	53703

Dirty Database under Subset Realizer

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53703
EVP Coffee	Madison	WI	53703
EVP Coffee	Madison	WI	53703

Dirty Database under Update Realizer

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53704
EVP Coffee	adison	WI	53703

Figure 2 Examples of a subset realizer and an update realizer.

- A *parfactor/subset* PUD is a triple $(\mathcal{D}, \tau, J^\star)$ where $\mathcal{D}$ is a parfactor database, τ maps every identifier $i \in ids(R^{\mathcal{D}})$, where $R \in \mathbf{S}$, to a discrete distribution $\tau[i]$ over $tuples(R) \cup \{\bot\}$, and $J^\star$ is an ordinary database. As usual, $\bot$ means that no tuple is generated.
- A *parfactor/update* PUD is a triple $(\mathcal{D}, \kappa, J^\star)$ where $\mathcal{D}$ is a parfactor database, κ maps every identifier $i \in ids(R^{\mathcal{D}})$ and tuple $t \in tuples(R)$, where $R \in \mathbf{S}$, to a discrete distribution $\kappa[i,t]$ over $tuples(R)$, and $J^\star$ is an ordinary database.

In a parfactor/subset PUD $\mathcal{U} = (\mathcal{D}, \tau, J^\star)$, the probability $\mathcal{R}\circ\mathcal{I}(I, J)$ is then defined as follows. If I is not a subset of J, then $\mathcal{R}\circ\mathcal{I}(I, J) = 0$; otherwise:

$$\mathcal{R}\circ\mathcal{I}(I, J) \stackrel{\text{def}}{=} \mathcal{D}(I) \times \prod_{\substack{i \in ids(J) \setminus \\ ids(I)}} \tau[i](J[i]) \times \prod_{\substack{i \in ids(\mathcal{D}) \setminus \\ ids(J)}} \tau[i](\bot)$$

That is, $\mathcal{R}\circ\mathcal{I}(I, J)$ is the probability of I (i.e., $\mathcal{D}(I)$), multiplied by the probability that each new tuple of J is produced by τ (i.e., $\tau[i](J[i])$), multiplied by the probability that each tuple identifier i missing in J is indeed not produced (i.e., $\tau[i](\bot)$).

In a parfactor/update PUD $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$, the probability $\mathcal{R}\circ\mathcal{I}(I, J)$ is then defined as follows. If I is not an update of J, then $\mathcal{R}\circ\mathcal{I}(I, J) = 0$; otherwise:

$$\mathcal{R}\circ\mathcal{I}(I, J) \stackrel{\text{def}}{=} \mathcal{D}(I) \times \prod_{i \in ids(I)} \kappa[i, I[i]](J[i])$$

That is, $\mathcal{R}\circ\mathcal{I}(I, J)$ is the probability of I (i.e., $\mathcal{D}(I)$), multiplied by the probability that κ changes each tuple $I[i]$ to $J[i]$ (i.e., $\kappa[i, I[i]](J[i])$).

▶ **Example 3.** Figure 2 shows the intended database from Example 2 and two unclean versions obtained by a subset realizer and an update realizer. The subset realizer introduces a duplicate, while the update realizer introduces two typos. These correspond to two types of common errors in relational data. Our PUD framework can naturally model such cases.

In Section 5, we discuss connections between these PUD instantiations and the deterministic models. Finally, in Section 6, we provide more concrete cases of PUD instantiations.

4 Computational Problems

We define three computational problems over PUDs that are motivated by the need to clean and query unclean data, and learn the intention and realization models from observed data.

Data Cleaning. Given a PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, we wish to compute a Most Likely Intention (MLI) database I, given the observed unclean database $J^\star$. We refer to this problem as *data cleaning* in PUDs.

▶ **Definition 4** (Cleaning). *Let* $\mathbf{S}$ *be a schema and* $\mathbf{R}$ *a representation system for PUDs. The problem* $(\mathbf{S}, \mathbf{R})$-cleaning *is that of computing an* MLI *of a given PUD* $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$*, that is, computing a database* I *such that the probability* $\mathcal{U}^\star(I)$ *is maximal (or, equivalently, the probability* $\mathcal{R} \circ \mathcal{I}\,(I, J^\star)$ *is maximal).*

Probabilistic query answering. A PUD defines a probabilistic database – a probability space over the intensions I. The problem of *Probabilistic Query Answering* (*PQA*) is that of evaluating a query over this probabilistic database. We adopt the standard semantics of query evaluation over probabilistic databases [13, 40], where the confidence in an answer tuple is its marginal probability.

▶ **Definition 5** (PQA). *Let* $\mathbf{S}$ *be a schema,* Q *a query over* $\mathbf{S}$*, and* $\mathbf{R}$ *a representation system for PUDs. The problem* $(\mathbf{S}, Q, \mathbf{R})$-PQA *is the following. Given a PUD* $\mathcal{U}$ *and a tuple* $\mathbf{a}$*, compute the* confidence *of* $\mathbf{a}$*, that is, the probability* $\Pr_{I \sim \mathcal{U}^\star}(\mathbf{a} \in Q(I))$.

For now, we assume that both $\mathcal{I}$ and $\mathcal{R}$ are fully specified. We next define the problem of learning models $\mathcal{I}$ and $\mathcal{R}$ using training (potentially labeled) data.

PUD learning. For a PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, the models $\mathcal{I}$ and $\mathcal{R}$ are typically represented using numeric *parameters*. For example, the parameters of a parfactor/subset PUD $(\mathcal{D}, \tau, J^\star)$ are those needed to represent $\mathcal{D}$ (e.g., the weights of the constraints), and the parameters that define the distributions over the tuples in both $\mathcal{D}$ and τ. By a *parametric intention* we refer to an intension model $\mathcal{I}_\Xi$ with a vector Ξ of uninitialized parameters, and by $\mathcal{I}_{\Xi/\mathbf{c}}$ we denote the actual intention model where Ξ is assigned the values in the vector $\mathbf{c}$. Similarly, by a *parametric realization* we refer to a realization model $\mathcal{R}_\Theta$ with a vector Θ of uninitialized parameters, and by $\mathcal{R}_{\Theta/\mathbf{d}}$ we denote the actual realization model where Θ is set to $\mathbf{d}$.

Following the concept of *maximum likelihood estimation*, the goal in learning is to find the parameters that best explain (i.e., maximize the probability) of the training examples. In the *supervised* variant, we are given examples of both unclean databases and their clean versions; in the *unsupervised* variant, we are given only unclean databases.

▶ **Definition 6** (Learning). *Let* $\mathbf{S}$ *be a schema, and* $\mathbf{R}$ *a representation system for parametric intensions and realizations. In the following problems we are given, as part of the input, the parametric intention and realization models* $\mathcal{I}_\Xi$ *and* $\mathcal{R}_\Theta$*, respectively.*

- *In the* supervised $(\mathbf{S}, \mathbf{R})$-learning *problem, we are also given a collection* $(I_j, J_j)_{j=1}^n$ *of database pairs (intention-realization examples), and the goal is to find parameter values* $\mathbf{c}$ *and* $\mathbf{d}$ *that maximize* $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}\,(I_j, J_j)$ *for* $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ *and* $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$.
- *In the* unsupervised $(\mathbf{S}, \mathbf{R})$-learning *problem, we are also given a collection* $(J_j)_{j=1}^n$ *of databases (realization examples), and the goal is to find parameter values* $\mathbf{c}$ *and* $\mathbf{d}$ *that maximize* $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}\,(J_j)$ *for* $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ *and* $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$*, where* $\mathcal{R} \circ \mathcal{I}\,(J_j)$ *is the marginal probability of* J_j*, that is,* $\sum_I \mathcal{R} \circ \mathcal{I}\,(I, J_j)$.

Note that the summation in the unsupervised variant is over the sample space of the intention model $\mathcal{I}$. While the reader might be concerned about the source of many examples (I_j, J_j) and J_j in the phrasing of the learning problems, it is oftentimes the case that a single large example (I, J) (or just J in the unsupervised variant) can be decomposed into many smaller examples. This depends on the independence assumptions in the parametric models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ as we discuss in Section 6.1. In the next sections, we give preliminary results on the introduced problems, focusing on parfactor/subset and parfactor/update PUDs.

5 Cleaning and Querying Unclean Data

In this section, we draw connections between data cleaning in the PUD framework (MLIs) and traditional *minimum repairs*. We also give preliminary results on the complexity of cleaning. Finally, we draw a connection between probabilistic query answering and certain answers.

5.1 Generalizing Minimum Repairs

We now show that the concept of an MLI in parfactor/subset PUDs generalizes the concept of a minimum subset repair, and the concept of an MLI in parfactor/update PUDs generalizes the concept of an optimal update repair. Minimum subset repairs correspond to MLIs of PUDs with hard (or heavy) constraints. Minimum update repairs correspond to MLIs over PUDs that assume both hard (or heavy) constraints and assumptions of independence among the attributes. From the viewpoint of computational complexity, this means that finding an exact MLI is not easier than finding a minimum repair, which is often computationally hard [30]. Therefore, we should aim for *approximation* guarantees (which have clear semantics in the probabilistic setting) if we wish to avoid restricting the generality of the input.

Subset repairs and parfactor/subset PUDs. Recall that in parfactor/subset PUDs (as defined in Section 3.1), every intention I with a nonzero probability is a subset of the observed unclean database $J^\star$. In particular, every MLI is subset of $J^\star$. Our first result relates cleaning in parfactor/subset PUDs to the traditional minimum subset (or *cardinality*) repairs. This result states, intuitively, that the notion of an MLI in a parfactor/subset PUD coincides with the notion of a minimum subset repair if the weight of the formulas is high enough and the probability of introducing error is small enough.

▶ **Theorem 7.** *Let $(\mathcal{D}, \tau, J^\star)$ be a parfactor/subset PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. For $i \in ids(J^\star)$, assume that $\mathcal{K}[i](\bot) > 0$ and $\tau[i](J^\star[i]) > 0$, let $q(i) = \mathcal{K}[i](J^\star[i])/(\mathcal{K}[i](\bot) \cdot \tau[i](J^\star[i]))$, and assume that $q(i) \geq 1$. There is a number M such that if $w(\varphi) > M$ for all $\varphi \in \Phi$ then the following are equivalent for all $I \subseteq J^\star$:*

1. *I is an MLI.*
2. *I is a minimum subset repair of $J^\star$ w.r.t. the weight function $w(i) = \log(q(i))$.*

Note that in the theorem, $q(i)$ is the ratio between $\mathcal{K}[i](J^\star[i])$, namely the probability that $\mathcal{K}$ produces the ith tuple of $J^\star$, and $\mathcal{K}[i](\bot) \cdot \tau[i](J^\star[i])$, namely the probability that $\mathcal{K}$ does not generate the ith tuple of $J^\star$ but τ does.

Next, we draw a similar connection between minimum update repairs and MLIs of parfactor/update PUDs.

Update repairs and parfactor/update PUDs. We now turn our attention to update repairs. Recall that in a parfactor/update PUD (defined in Section 3.1), the intended clean database I is assumed to be an update of the observed unclean database $J^\star$. We establish a result analogous to Theorem 7, stating conditions under which MLIs for parfactor/update PUDs coincide with traditional minimum update repairs.

Let $\mathbf{S}$ be a schema, and let $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$ be a parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. We say that $\mathcal{U}$ is *attribute independent* if $\mathcal{K}$ and κ feature probabilistic independence among the attributes. More precisely, if $i \in R^{\mathcal{D}}$ for $R \in \mathbf{S}$ with $sig(R) = (A_1, \ldots, A_k)$, then we assume that $\mathcal{K}[i](a_1, \ldots, a_k)$ can be written as $\mathcal{K}[i](a_1, \ldots, a_k) = \prod_{j=1}^{k} \mathcal{K}_{A_j}[i](a_j)$ and, for $t =$

$(b_1, \ldots, b_k)$, that $\kappa[i,t](a_1, \ldots, a_k)$ can be written as $\kappa[i,t](a_1, \ldots, a_k) = \prod_{j=1}^{k} \kappa_{A_j}[i, b_j](a_j)$. In particular, the choice of the value a_j depends only on b_j and not on other values $b_{j'}$.

The following theorem states that the concept of an MLI of a parfactor/update PUD coincides with the concept of a minimum update repair when the PUD is attribute independent and, moreover, the weight of the integrity constraints is high.

▶ **Theorem 8.** *Let $\mathcal{U} = (\mathcal{D}, \tau, J^\star)$ be an attribute-independent parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$ such that $\mathcal{U}^\star(I) > 0$ for at least one consistent update I of $J^\star$. There is a number M such that if $w(\varphi) > M$ for all $\varphi \in \Phi$, then the following statements are equivalent for all $I \subseteq J^\star$:*

1. *I is an MLI.*
2. *I is a minimum update repair w.r.t. the weight function*

$$w(i.A, a) = -\log(\mathcal{K}_A[i](a) \cdot \kappa_A[i, a](J^\star[i].A)) \,.$$

Note that $\mathcal{K}_A[i](a) \cdot \kappa_A[i, a](J^\star[i].A)$ is the probability that a is produced by $\mathcal{K}$ for the cell $i.A$, and that a is then changed to $J^\star[i].A$ via κ. Also note that in the case where this product is zero, we slightly abuse the notation by assuming that the weight is infinity.

5.2 Complexity of Cleaning with Key Constraints

We now present a complexity result on computing an MLI of a parfactor/subset PUD in the presence of key constraints. The following theorem states that in the case of a single key constraint per relation (which is the common setup, e.g., for the analysis of certain query answering [26, 3, 24]), an MLI can be found in polynomial time. Note that we do not make any assumption about the parameters; in particular, it may be the case that an MLI violates the key constraints since the constraints are weak. Regarding the representation of the probability spaces $\mathcal{K}$ and τ, the only assumption we make is that, given a tuple t, the probabilities $\mathcal{K}[i](t)$ and $\tau[i](t)$ can be computed in polynomial time.

▶ **Theorem 9.** *Let $(\mathcal{D}, \tau, J^\star)$ be a parfactor/subset PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. If Φ consists of (at most) one key constraint per relation, and no relation of $J^\star$ has duplicate tuples, then an MLI can be computed in polynomial time.*

It is left for future investigation to seek additional constraints (e.g., functional dependencies) for which an MLI can be found in polynomial time. Note that Theorem 7 implies that (under conventional complexity assumptions) we cannot generalize the polynomial-time result to all sets of functional dependencies, since finding a minimum subset repair might be computationally hard [32, 30].

5.3 Probabilistic Query Answering

For probabilisitic query answering, we again focus on the parametric/subset PUDs, and now we draw a connection to *consistent query answering* over the cardinality repairs. Recall that a *consistent answer* for a query Q over an inconsistent database J is a tuple t that belongs to $Q(I)$ for every cardinality repair I of J.

Let $\mathbf{S}$ be a schema, $J^\star$ a database, and Φ a set of integrity constraints. Let $M = |ids(J^\star)|$. The *uniform* parfactor/subset PUD for $J^\star$ and Φ with the parameters p and u, denoted $\mathcal{U}_{p,u}(J^\star, \Phi)$ or just $\mathcal{U}_{p,u}$ if $J^\star$ and Φ are clear from the context, is the parfactor/subset PUD $(\mathcal{D}, \tau, J^\star)$ with $\mathcal{D} = (\mathcal{K}, \Phi, w)$ such that the following hold.

- For all $R \in \mathbf{S}$ we have $ids(R^{\mathcal{K}}) = ids(R^{J^\star})$ and $\mathcal{K}[i](x) = 1/M$ for every tuple identifier i and argument x in $R^{\mathcal{K}} \cup \{\bot\}$. The remaining mass (required to reaching 1) is given to an arbitrary tuple outside of $J^\star$.
- $w(\varphi) = u$ for every $\varphi \in \Phi$.
- $\tau[i](\bot) = p$, and $\tau[i](t) = (1-p)/M$ for every identifier i and tuple t in $R^{\mathcal{K}}$. Again, the remaining mass is given to an arbitrary tuple outside of $J^\star$.

Observe that $\mathcal{D}$ is defined in such a way that every subset I of $J^\star$ has the same prior probability $\mathcal{K}(I)$, namely $1/M^{|J^\star|}$.

The following theorem states that, for $\mathcal{U}_{p,u}$, the consistent answers are precisely the answers whose probability approaches one when all of the following hold: *(1)* the probability of introducing error (i.e., $1-p$) approaches zero; *and (2)* the weight of the weak constraints (i.e., u) approaches infinity, that is, the constraints strengthen towards hardness.

▶ **Theorem 10.** *Let $J^\star$ be a database, Φ a set of integrity constraints, Q a query, and t a tuple. The following are equivalent:*

1. *t is a consistent answer over the cardinality repairs.*
2. $\lim_{p \longrightarrow 1} \lim_{w \longrightarrow \infty} \Pr_{I \sim \mathcal{U}^\star_{p,u}}(t \in Q(I)) = 1$.

Therefore, Theorem 10 sheds light on the role that the consistent answers have in probabilisitic query answering over parfactor/subset PUDs.

6 Learning Probabilistic Unclean Databases

We now give preliminary results on PUD learning, focusing on parfactor/update PUDs. We begin by describing the setup we consider in this section and the representation system $\mathbf{R}$ we use to describe the parametric intention and realization models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$.

6.1 Setup

To discuss the learning of parameters, we need to specify the actual parametric model we assume. Let $\mathbf{S}$ be a schema, and let $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$ be a parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. Since we restrict the discussion to parfactor/update PUDs, we assume (without loss of generality) that the identifiers in $\mathcal{D}$ are exactly those in $J^\star$, that is, $ids(\mathcal{D}) = ids(J^\star)$.

In our setup, $\mathcal{K}$ of $\mathcal{D}$ and κ of $\mathcal{U}$ are expressed in a parametric form that allows us to define the parametric intention and realization models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ as *Gibbs* distributions. We refer to these as *Gibbs parfactor/update PUD models*, and define them as follows.

Parametric intention. To specify $\mathcal{K}$, we assume that for each relation symbol $R \in \mathbf{S}$, the probability $\mathcal{K}[i](t)$, with $i \in ids(R^{\mathcal{D}})$, is expressed in the form of an exponential distribution $\mathcal{K}[i](t) = \exp\left(\sum_{f \in F} w_f f(t)\right)$ where each $f \in F$ is an arbitrary function (*feature*) over t, and each weight w_f is a real number.

For example, a feature $f \in F$ may be a function that takes as input a tuple and returns a value in $\{-1, 1\}$. An example feature f can state that $f(t) = 1$ if $t[\texttt{gender}] = \texttt{female}$ and, otherwise, $f(t) = -1$. Another example is $f[t] = 1$ if $t[\texttt{zip}]$ starts with $\texttt{53}$ and $t[\texttt{state}] = \texttt{WI}$, and otherwise $f[t] = -1$. Additional examples of such features include the ones used in our prior work on HoloClean [34] to capture the co-occurrence probability of attribute value pairs. Each weight w_f corresponds to a parameter of the model. An assignment to these weights gives as a probability distribution, similarly to probabilistic graphical models [25].

The parameter vector Ξ of the parametric intention model $\mathcal{I}_\Xi$ consists of two sets of parameters: the weights w_f for each feature $f \in F$, and the weights $w(\varphi)$, which we write as w_φ for uniformity of presentation, for each constraint $\varphi \in \Phi$. Thus, the overall parametric intention model $\mathcal{I}_\Xi$ is expressed as a parametric Gibbs distribution.

We will take a special interest in the case where the integrity constraints are *unary*, which means that they have the form $\forall x[\gamma(x)]$, where γ is quantifier free; hence, a unary constraint is a statement about a single tuple. Examples of unary constraints are restricted cases of conditional functional dependencies [7, 16]. An example of such a constraint can be "age smaller than 10 cannot co-occur with a salary greater than \$100k."

Parametric realization. We consider a parametric realization model that is similar to the parametric intention model presented above, which is again a parametric Gibbs distribution. For each relation symbol $R \in \mathbf{S}$ and every pair $(t, t') \in \text{tuples}(R) \times \text{tuples}(R)$, the probability $\kappa[i,t](t')$, with $i \in ids(\mathcal{D})$, is expressed in the form of the Gibbs distribution $\kappa[i,t](t') = \frac{1}{Z_\kappa(t)} \exp\left(\sum_{g \in G} w_g g(t,t')\right)$, where G is the set of features, and each g is an arbitrary function (*feature*) over (t,t'), each weight w_g is a real number, and $Z_\kappa(t)$ is a normalization constant defined as $Z_\kappa(t) = \sum_{t' \in \text{tuples}(R)} \exp\left(\sum_{g \in G} w_g g(t,t')\right)$. Hence, we get a parametric model for the probability distribution κ.

As an example, a feature function $g(t,t')$ may capture spelling errors: $g(t,t') = 1$ if $t'[\mathsf{city}]$ can be obtained by deleting one character from $t[\mathsf{city}]$, and otherwise, $g(t,t') = -1$. The parameter vector Θ of the parametric realization model $\mathcal{R}_\Theta$ consists of the set of weights w_g for each feature $g \in G$.

Assumptions. We make two assumption here. First, we assume that the attributes of all relation symbols in $\mathbf{S}$ take values over a finite and given set. This means that for each relation symbol $R \in \mathbf{S}$, $\text{tuples}(R)$ is also finite and given as input. Second, all features describing $\mathcal{K}$ and κ can be computed efficiently, that is, in polynomial time in the size of the input.

Obtaining examples for learning. For supervised learning, we require a training collection $(I_j, J_j)_{j=1}^n$ and for unsupervised learning, a training collection $(J_j)_{j=1}^n$. We would like to make the case that, oftentimes, a single large example can be broken down into many small examples. Recall that in our setup (Gibbs parfactor/update PUDs), cross-tuple correlations can be introduced only by the integrity constraints in Φ. Consider, for instance, the case where all constraints in Φ are unary. Then, we get *tuple-independent* parfactor/update PUDs, and each tuple identifier i can become an example database: $(I[i], J[i])$ in the supervised case, and $J[i]$ in the unsupervised case. For general constraints, cross-tuple correlations exist, and each example (I_j, J_j) and (J_j) can be obtained by taking correlated groups of tuples from $J^\star$ by considering different values for the attributes participating in each constraint $\varphi \in \Phi$. The number of tuples contained in each example depends on the constraints. This is a standard practice with parameterized probabilistic models as the ones we consider here [31].

6.2 Supervised Learning

We begin by considering supervised $(\mathbf{S}, \mathbf{R})$-learning. All results presented in this section build upon standard tools from statistical learning. We are given a collection $(I_j, J_j)_{j=1}^n$ of intention-realization examples, and the goal is to find parameter values $\mathbf{c}$ and $\mathbf{d}$ that maximize the likelihood of pairs $(I_j, J_j)_{j=1}^n$, that is, $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}(I_j, J_j)$ for $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ and $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$. To facilitate the analysis, we write this objective function as a sum over terms by considering the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n) = -\log \prod_{j=1}^n \mathcal{R} \circ \mathcal{I}(I_j, J_j) = -\sum_{j=1}^n \log \mathcal{R} \circ \mathcal{I}(I_j, J_j)$, and seek parameter values $\mathbf{c}$ and $\mathbf{d}$ that minimize it. For parfactor/update PUDs we have that $\mathcal{R} \circ \mathcal{I}(I, J) = \mathcal{D}(I) \times \prod_{i \in ids(I)} \kappa[i, I[i]](J[i])$ where $\mathcal{D}(I) = \mathcal{K}(I) \times 1/Z \times \exp(-\sum_{\varphi \in \Phi} w(\varphi) \times |V(\varphi, I)|)$. For the Gibbs parfactor/update PUD, $\mathbf{c}$

corresponds to an assignment of parameters w_f describing $\mathcal{K}$ and parameters $w_\varphi = w(\varphi)$ for the constraints $\varphi \in \Phi$. Similarly, $\mathbf{d}$ corresponds to an assignment of parameters w_g describing κ. We use $Z(\mathbf{c})$ to denote the partition function Z under the parameters $\mathbf{c}$. We have:

$$l(\mathbf{c}, \mathbf{d}; (I_j, J_j)_{j=1}^n) = -\sum_{j=1}^{n} \log \left(\mathcal{K}(I_j; \mathbf{c}) \ \times \ \exp \left(-\sum_{\varphi \in \Phi} w_\varphi \times |V(\varphi, I_j)| \right) \right) + n \log Z(\mathbf{c})$$
$$- \sum_{j=1}^{n} \sum_{i \in ids(I_j)} \log \left(\kappa[i, I[i]; \mathbf{d}](J_j[i]) \right) \tag{1}$$

where $\mathcal{K}(I_j; \mathbf{c})$ denotes that $\mathcal{K}$ is parametrized by $\mathbf{c}$ and $\kappa[i, I[i]; \mathbf{d}]$ denotes that κ is parametrized by $\mathbf{d}$.

Our goal becomes to minimize the expression in (1). It is well-known from the ML literature that there is no analytical solution to such minimization problems, and one needs to use iterative *gradient-based* methods [25]. We investigate whether gradient-based methods can indeed find a global minimum, and whether computing the gradient of this objective during each iteration is tractable. For the first question, the answer is positive.

▶ **Proposition 11.** *$l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ is a convex function of (Ξ, Θ).*

This proposition implies that the optimization objective for supervised learning has only global optima. Hence, it is guaranteed that any gradient-based optimization method will converge to a global optimum. Next, we study when the gradient of $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ with respect to $\mathbf{c}$ and $\mathbf{d}$ can be computed efficiently.

To compute the gradient of $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$, one has to compute $\frac{\partial}{\partial c_l} \log Z(\mathbf{c})$. To compute this derivative a full inference step is required [25]. This is because computing this gradient amounts to computing the expected value for each feature (corresponding to each parameter c_l) according to the distribution defined by $\mathbf{c}$ [25, Proposition 20.2]. However, marginal inference is often #P-hard [18]. In our setup, the constraints in Φ correspond to features, and it is not clear whether the gradient can be efficiently computable. In general, one can still estimate the aforementioned gradient by using approximate inference methods such as *Markov Chain Monte Carlo* (MCMC) methods [9, 39] or *belief propagation* [41]. While effective in practice, these methods do not come with guarantees on the quality of the obtained solution. Next, we focus on an instance of PUD learning where exact inference is tractable (linear on tuples(R)), hence, we can compute the exact gradients of the aforementioned optimization objective efficiently.

Tuple independence. We focus on Gibbs parfactor/update PUD models where all constraints in Φ are unary. Here, $(I_j, J_j)_{j=1}^n$ corresponds to a collection of n examples, each of which having one tuple identifier. We use $I_j[0]$ and $J_j[0]$ to denote the tuples in the example (I_j, J_j). In the extended version of our paper [36], we show that the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ factorizes as $l(\mathbf{c}, \mathbf{d}; (I_j, J_j)_{j=1}^n) = \sum_{j=1}^n l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$, where each $l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$ is a convex function of Ξ and Θ (see the extended version of our paper [36]). Moreover, we show that the gradient of each $l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$ can be evaluated in time linear to tuples(R) where R is the relation corresponding to the tuple identifier associated with example (I_j, J_j) (see the extended version of our paper [36]). Hence, we get the following.

▶ **Theorem 12.** *Given a training collection* $(I_j, J_j)_{j=1}^n$ *and a Gibbs parfactor/update PUD model with unary constraints, the exact gradient of* $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ *can be evaluated in* $O(n \cdot \max_{R \in \mathbf{S}} |\text{tuples}(R)|)$ *time.*

The above theorem implies that convex-optimization techniques such as *stochastic gradient descent* [8] can be used to scale to large PUD learning instances (i.e., for large n).

A question that arises is about the number of examples (i.e., n) required to learn a PUD model. To answer this question, we study the convergence of *supervised* $(\mathbf{S}, \mathbf{R})$*-learning* for Gibbs parfactor/update PUD models with unary constraints. For that, we view the collection $(I_j, J_j)_{j=1}^n$ as independent and identically distributed (i.i.d.) examples, drawn from a distribution that corresponds to a Gibbs parfactor/update PUD model with unary constraints and true parameters $\mathbf{c}^\star$ and $\mathbf{d}^\star$. By the law of large numbers, the *Maximum Likelihood Estimates* (MLE) $\mathbf{c}$ and $\mathbf{d}$ are guaranteed to converge to $\mathbf{c}^\star$ and $\mathbf{d}^\star$ in probability. This means that for arbitrarily small $\epsilon > 0$ we have that $P(|\mathbf{c} - \mathbf{c}^\star| > \epsilon) \to 0$ as $n \to \infty$. The same holds for $\mathbf{d}$. Moreover, we show that the MLE $\mathbf{c}$ and $\mathbf{d}$ satisfy the property of *asymptotic normality* [27]. Intuitively, asymptotic normality states that the estimator not only converges to the unknown parameter, but it converges fast enough at a rate of $1/\sqrt{n}$. This implies that to achieve the error ϵ for $\mathbf{c}$ and $\mathbf{d}$, one only needs $n = O(\epsilon^{-2})$ training examples.

▶ **Theorem 13.** *Consider a training collection* $(I_j, J_j)_{j=1}^n$ *drawn i.i.d. from a Gibbs parfactor/update PUD model with unary constraints and true parameters* $\mathbf{c}^\star$ *and* $\mathbf{d}^\star$*. The maximum likelihood estimates* $\mathbf{c}$ *and* $\mathbf{d}$ *satisfy* asymptotic normality*, that is,*

$$\sqrt{n}\,(\mathbf{c} - \mathbf{c}^\star) \to \mathcal{N}\left(0, \Sigma^2_{\mathbf{c}^\star}\right) \text{ as } n \to \infty \text{ and } \sqrt{n}\,(\mathbf{d} - \mathbf{d}^\star) \to \mathcal{N}\left(0, \Sigma^2_{\mathbf{d}^\star}\right) \text{ as } n \to \infty$$

where $\Sigma^2_{\mathbf{c}^\star}$ *and* $\Sigma^2_{\mathbf{d}^\star}$ *are the asymptotic variance of the estimates* $\mathbf{c}$ *and* $\mathbf{d}$*.*

Note that both the *multivariate Gaussian* distribution $\mathcal{N}(\mu, \Sigma^2)$ and the *asymptotic variance* are defined in classic statistics literature [27].

6.3 Unsupervised Learning

We now present preliminary results for unsupervised $(\mathbf{S}, \mathbf{R})$-learning. We are given a training collection $(J_j)_{j=1}^n$ and seek to find $\mathbf{c}$ and $\mathbf{d}$ that minimize the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (J_j)_{j=1}^n) = -\sum_{j=1}^n \log \sum_I \mathcal{R} \circ \mathcal{I}(I, J_j)$. Again, there is no analytical solution for finding optimal $\mathbf{c}$ and $\mathbf{d}$. Hence, one needs to use iterative gradient-based approaches, and again the questions of convexity and gradient computation arise. In the general case, this function is *not* necessarily convex. Hence, gradient-based methods are not guaranteed to converge to a global optimum. However, one can still solve the corresponding optimization problem using non-convex optimization methods [6]. Nevertheless, we show next that when realizers do not introduce *too much error*, we can establish guarantees.

Low-noise condition. Consider a Gibbs parfactor/update PUD model. We say that a PUD defined by $\mathcal{I}_{\Xi/\mathbf{c}}$ and $\mathcal{R}_{\Theta/\mathbf{d}}$ satisfies the *low-noise condition* with probability p if the realizer introduces an error with probability at most p. That is, for all intensions I and identifiers $i \in ids(I)$, it is the case that $\Pr(J[i] = I[i] \mid I) \geq 1 - p$. We have the following.

▶ **Theorem 14.** *Consider a Gibbs parfactor/update PUD model where* Ξ *takes values from a compact convex set. Given a training collection* $(J_j)_{j=1}^n$*, there exists a fixed probability* $p > 0$ *such that, under the low-noise condition with probability* p*, the negative log-likelihood* $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (J_j)_{j=1}^n)$ *is a convex function of* Ξ*.*

Hence, in certain cases, it is possible to find a global optimum of the overall negative log-likelihood. For that, the low-noise condition should hold with probability p that is also bounded, that is, it cannot be arbitrarily large. We can show that, if the low-noise condition holds with probability p, it is indeed bounded for Gibbs parfactor/update PUDs with unary constraints (see the extended version of our paper [36]). We then find a global optimum as follows. We assume a simple parametric realization $\mathcal{R}_\Theta$, that is, a model for which we can efficiently perform *grid search* over the space of parameter values $\mathbf{d}$. To find the global optimum for the negative log-likelihood, we solve a series of convex optimization problems over Ξ for different fixed $\Theta = \mathbf{d}$. For each of these problems, we are guaranteed to find a corresponding global optimum $\mathbf{c}$, and by performing a grid search we are guaranteed to find the overall global optima $\mathbf{c}$ and $\mathbf{d}$. This approach has been shown to converge for similar simple non-convex problems [37, 33].

Finally, similarly to supervised learning, the negative log-likelihood for fixed $\Theta = \mathbf{d}$ decomposes into a sum of convex losses over $(J_j)_{i=1}^n$ where each example J_j contains a single tuple. We use $J_j[0]$ to denote that tuple. We have that $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}, (J_j)_{j=1}^n) = \sum_{j=1}^n l'(\mathbf{c}, \mathbf{d}; J_j[0])$ where $\mathbf{d}$ is fixed. We show that the gradient of each $l'(\mathbf{c}, \mathbf{d}; J[0])$ can be evaluated in polynomial time to tuples(R) where R is the relation corresponding to the tuple identifier associated with the example (J_j).

It is left for future work to find sufficient conditions for p to be bounded for PUD models with more general constraints, as well as the complexity and convergence aspects.

7 Concluding Remarks

Taking inspiration from our experience with the HoloClean system [34], we introduced the concept of Probabilistic Unclean Databases (PUDs), a framework for unclean data that follows a noisy channel approach to model how errors are introduced in data. We defined three fundamental problems in the framework: cleaning, probabilistic query answering, and PUD learning (parameter estimation). We introduced PUD instantiations that generalize the deterministic concepts of subset repairs and update repairs, presented preliminary complexity, convergence, and learnability results.

This paper opens up many research directions for future exploration. One is to investigate the complexity of cleaning in more general configurations than the ones covered here. Moreover, in cases where probabilistic cleaning is computationally hard, it is of natural interest to find *approximate* repairs that have a probability (provably) close to the maximum. Another direction is the complexity of probabilistic query answering and approximation thereof, starting with the most basic constraints (e.g., primary keys) and queries (e.g., determine the marginal probability of a fact). Finally, an important direction is to devise learning algorithms for cases beyond the ones we discussed here. In particular, it is of high importance to understand when we can learn parameters without training data, based only on the given dirty database, under more general noisy realization models than the ones discussed in this paper.

References

1 Serge Abiteboul, Marcelo Arenas, Pablo Barceló, Meghyn Bienvenu, Diego Calvanese, Claire David, Richard Hull, Eyke Hüllermeier, Benny Kimelfeld, Leonid Libkin, Wim Martens, Tova Milo, Filip Murlak, Frank Neven, Magdalena Ortiz, Thomas Schwentick, Julia Stoyanovich, Jianwen Su, Dan Suciu, Victor Vianu, and Ke Yi. Research Directions for Principles of Data Management (Abridged). *SIGMOD Record*, 45(4):5–17, 2016. doi:10.1145/3092931.3092933.

2 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM, 2009.

3 Periklis Andritsos, Ariel Fuxman, and Renée J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *ICDE*, page 30. IEEE Computer Society, 2006.

4 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *PODS*, pages 68–79. ACM, 1999. doi:10.1145/303976.303983.

5 Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

6 Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

7 Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional Functional Dependencies for Data Cleaning. In *ICDE*, pages 746–755. IEEE, 2007.

8 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

9 N. E. Breslow and D. G. Clayton. Approximate Inference in Generalized Linear Mixed Models. *Journal of the American Statistical Association*, 88(421):9–25, 1993.

10 Marco Calautti, Leonid Libkin, and Andreas Pieris. An Operational Approach to Consistent Query Answering. In *PODS*, pages 239–251. ACM, 2018.

11 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1):90–121, 2005.

12 Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic Data Cleaning: Putting Violations into Context. In *ICDE*, pages 458–469, 2013.

13 Nilesh N. Dalvi and Dan Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

14 C. J. Date. Referential Integrity. In *VLDB*, pages 2–12. VLDB Endowment, 1981.

15 Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. Dichotomies in the Complexity of Preferred Repairs. In *PODS*, pages 3–15, New York, NY, USA, 2015. ACM.

16 Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, June 2008.

17 Terry Gaasterland, Parke Godfrey, and Jack Minker. An Overview of Cooperative Answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

18 Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How Hard is Inference for Structured Prediction? In *ICML*, pages 2181–2190. JMLR.org, 2015.

19 Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The Most Probable Database Problem. In *BUDA*, 2014.

20 Ihab F. Ilyas. Effective Data Cleaning with Continuous Evaluation. *IEEE Data Eng. Bull.*, 39:38–46, 2016.

21 Abhay Kumar Jha, Vibhor Rastogi, and Dan Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.

22 Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

23 Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361, pages 53–62. ACM, 2009.

24 Phokion G. Kolaitis and Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

25 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

26 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.

27 Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

28 Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, New York, NY, USA, 2002. ACM.

29 Leonid Libkin. Incomplete Data: What Went Wrong, and How to Fix It. In *PODS*, pages 1–13, New York, NY, USA, 2014. ACM.

30 Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing Optimal Repairs for Functional Dependencies. In *PODS*, pages 225–237. ACM, 2018.

31 Ben London, Bert Huang, Ben Taskar, and Lise Getoor. Collective Stability in Structured Prediction: Generalization from One Example. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 828–836, 17–19 June 2013.

32 Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*, pages 179–193, 2007.

33 Cong Ma, Kaizheng Wang, Yuejie Chi, and Yuxin Chen. Implicit Regularization in Nonconvex Statistical Estimation: Gradient Descent Converges Linearly for Phase Retrieval, Matrix Completion and Blind Deconvolution. *arXiv preprint*, 2017. `arXiv:1711.10467`.

34 Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB*, 10(11), 2017.

35 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.

36 Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A Formal Framework For Probabilistic Unclean Databases. *CoRR*, abs/1801.06750, 2018. `arXiv:1801.06750`.

37 Christopher De Sa, Christopher Ré, and Kunle Olukotun. Global Convergence of Stochastic Gradient Descent for Some Non-convex Matrix Problems. In *ICML*, volume 37 of *JMLR Proceedings*, pages 2332–2341. JMLR.org, 2015. URL: `http://jmlr.org/proceedings/papers/v37/sa15.html`.

38 Prithviraj Sen, Amol Deshpande, and Lise Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

39 Sameer Singh, Michael Wick, and Andrew McCallum. Monte Carlo MCMC: Efficient Inference by Approximate Sampling. In *MNLP-CoNLL*, pages 1104–1113. Association for Computational Linguistics, 2012.

40 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011.

41 Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. Tree-reweighted belief propagation algorithms and approximate ML estimation via pseudo-moment matching. In *AISTATS*, January 2003.

42 Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468. ACM, 2014.

43 Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.