A Formal Framework for Probabilistic Unclean Databases

Christopher De Sa
Cornell University
Ithaca, NY, USA

Ihab F. Ilyas
University of Waterloo
Waterloo, ON, Canada

Benny Kimelfeld
Technion
Haifa, Israel

Christoper Ré
Stanford University
Stanford, CA, USA

Theodoros Rekatsinas
University of Wisconsin-Madison
Madison, WI, USA

ABSTRACT

Traditional modeling of inconsistency in database theory casts all possible "repairs" equally likely. Yet, effective data cleaning needs to incorporate statistical reasoning. For example, yearly salary of \$100k and age of 22 are more likely than \$100k and 122 and two people with same address are likely to share their last name (i.e., a functional dependency tends to hold but may occasionally be violated). 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 the actual database is realized through the introduction of noise.

Formally, a Probabilistic Unclean Database (PUD) is a triple that consists of a probabilistic database that we call the "intention", a probabilistic data transformator that we call the "realization", and a dirty database that we call the "observation". We define three computational problems in this framework: cleaning (find the most likely intention), probabilistic query answering (compute the probability of an answer tuple), and learning (find the most likely parameters given examples of clean and dirty databases). We illustrate the framework on concrete representations of PUDs, show that they generalize traditional concepts of repairs such as cardinality and value repairs, draw connection to consistent query answering, and prove tractability results. We further show that parameters can be learned in practical instantiations, and in fact, prove that under certain conditions we can learn directly from a single dirty database without any need for clean examples.

1. INTRODUCTION

We are in the golden era of data-centric applications. We are seeing widespread investments in complex analytics and machine-learning solutions that analyze large volumes of data and answer complex questions. Yet, there is a hidden catch: state-of-the-art systems rely on high-effort tasks that are involved in data collection and preparation. Data collection often introduces incomplete, erroneous, replicated, or conflicting data. A classic example is the integration of data from multiple sources with conflicting information and/or formats. Even when data is machine generated (e.g., server logs and sensor readings), errors may occur due to device malfunctions. It is *data cleaning* that has become the key development bottleneck in modern analytics [Cro16].

Inconsistent information is ubiquitous in data-driven applications. This problem was recognized already in the 1970s [Cod75], and since then an increasing body of work has focused on the problem of *automating* the management of inconsistent data [ABC99, Len02, LB07, Lib14]. Past theoretical research established fundamental results that concentrate on tractability boundaries of data cleaning and query answering in the presence of inconsistency [AK09, FKK15, KW17]. In their majority, these works adopt a deterministic interpretation of data consistency and cast all repairs equally likely. Here, a *repair* is a consistent database that is obtained from the dirty database by a set of operations that features some form of non-redundency [ABC99, AK09].

The theoretical developments have inspired practical tools that aim to automate data repairing [YEN+11, CIP13, WT14, Ily16]. To prioritize across possible repairs, the majority of tools use the notion of *minimality* [CM05, KL09] as an operational cleaning principle. Informally, minimality states that given two candidate sets of repairs, the one with fewer changes with respect to the original data is preferable. Intuitively, minimality introduces a strong prior that noise in an input (dirty) database is limited. Yet, this approach falls short of capturing statistical properties of clean and dirty data that can be valuable for data cleaning [Ily16].

Effective data cleaning needs to incorporate statistical reasoning. Our recent work on HoloClean [RCIR17] provides strong evidence that adopting a *probabilistic* perspective, casting data cleaning as *statistical learning and inference*, leads to significantly more effective cleaning methodologies. Specifically, our empirical results show that statistical learning doubles (on average) the accuracy of cleaning, compared to deterministic minimality-based solutions, and demonstrates that the probabilistic modeling allows to quantify the confidence on the precision of automated repairs.

Vision. Our experience with HoloClean, as well as prior art on probabilistic cleaning [AFM06, GdBS14], motivate the need to better understand the connections between statistical learning/inference and classical techniques for data repairing. To this end, we introduce a formal framework for probabilistic data cleaning built around the concept of *Probabilistic Unclean Databases (PUDs)* and use this framework to bridge classical results in data repairing and statistical learning.

This paper falls within the bigger vision of bridging database theory with learning theory as outlined in a recent survey [AAB+16]. Specifically, 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* [BHS+07]. 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. Examples include image segmentation and part-of-speech tagging. In all these problems structure is encoded via logic-based constraints [GRSY15] 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.

Probabilistic unclean databases. We introduce a formal framework for probabilisitic data cleaning. Our framework is based upon the concept of a *Probabilistic Unclean Database* (*PUD*). Intuitively, a PUD consists of a dirty database $J^\star$, along with a two-step generative model that describes how dirty databases are distributed; the first step samples a clean database, and the second introduces noise. We refer to the distribution that produces the intended clean database as the *intention model* (or just *intention* for short), and the distribution that introduces noise for realising the actual observed database is the *realization model* (or just *realization* for short). The PUD represents a probability distribution over the possible clean databases—this is the posterior probability given the observation $J^\star$.

More formally, a PUD is a triple $(\mathcal{I}, \mathcal{R}, J^\star)$, where $\mathcal{I}$, the intention, is a probabilistic database, and $\mathcal{R}$, the realization, maps every sample I of $\mathcal{I}$ to a probabilistic database $\mathcal{R}_I$ (obtained from I by introducing noise), and $J^\star$, the observation, is a dirty database. The models $\mathcal{I}$ and $\mathcal{R}$ define probability space $\mathcal{R}_\mathcal{I}$ over pairs (I, J), and the PUD $(\mathcal{I}, \mathcal{R}, J^\star)$ defines a probabilistic database over Is by conditioning $\mathcal{R}_\mathcal{I}$ on $J = J^\star$.

To make the case for the PUD framework, we define three computational tasks that are driven by real-life motivation. The first problem is *cleaning*, which we capture as a *maximum-likelihood inference*: given $(\mathcal{I}, \mathcal{R}, J^\star)$, compute a database I that maximizes the probability $\mathcal{R}_\mathcal{I}(I, J^\star)$; we call such I a *Most Likely Intention* (MLI). The second problem is *probabilistic query answering*, namely, evaluate a query Q on the probabilistic cleaning result induced by $(\mathcal{I}, \mathcal{R}, J^\star)$. Here, we adopt the traditional semantics of query answering over probabilistic databases as *marginal inference* where the goal is to determine the marginal probability of each possible query answer [DS04, SORK11]. The third problem is *PUD learning*: given a collection T of examples (I, J), find the parameters of $\mathcal{I}$ and $\mathcal{R}$ that maximize the *log-likelihood* (sum-of-logs/product of the probabilities) of T. For learning to be well defined, we assume that $\mathcal{I}$ and $\mathcal{R}$ have *parametric* representations $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$, where Ξ and Θ are vectors of symbolic parameters, such that actual $\mathcal{I}$ and $\mathcal{R}$ are obtained by assigning numeric values to the parameters.

The definition of a PUD is abstract, as it includes only the semantics of $\mathcal{I}$ and $\mathcal{R}$, and does not mention how they are represented. Statistical models are typically represented compactly, classic examples being graphical models and exponential (factor-graph) models that restrict the correlation between random variables. We illustrate our framework by instantiating it with two specific representation systems for PUDs. In both representations, the intention model is a probabilistic database that we call a *Markov Logic Database* (MLD), which we explain next. The two representations differ in their realization. In an *MLD/subset* PUD, the realizer can introduce new tuples; hence, the clean database is assumed to be a subset of $J^\star$. In an *MLD/update* PUD, the realizer can update table cells; hence, the clean database is assumed to be obtained from $J^\star$ by changing cell values.

In an MLD, each tuple is generated independently via a probabilistic *tuple generator*, and in addition, we have weak (and hard) constraints. The constraints affect the probability of a database in the sense of Markov random fields [Pea89], namely, each *grounding* of the rule adds a multiplicative factor to the probability. In an MLD, the constraints are *Denial Constraints* (DCs) [GGM92], phrased in tuple relational calculus, and each grounding is obtained by assigning tuples to the free variables. These constraints include the Functional Dependencies (FDs) and *conditional* FDs [BFG+07] that are traditionally used in data repairing [Ber11].

We establish several preliminary results on MLD/subset PUDs and MLD/update PUDs. In particular, we show configurations of the two where MLIs coincide with *minimum repairs*—*cardinality repairs* [LB07] in the case of MLD/subset, and *optimal V-repairs* [KL09] in the case of MLD/update. Consequently, for these configurations we immediately establish complexity results on PUDs from past literature on the complexity of computing a minimum repair [LB07, KL09, AK09, LK17]. In addition, we show that over MLD/subset PUDs, an MLI can be computed in polynomial time in a well studied scenario [KW17, AFM06, KP12], namely there is a single key constraint. Interestingly, unlike traditional (non-probabilistic) repairs, here the key constraint may be *weak* (e.g., two people are unlikely, but might, have the same first and last name), and the required MLI may, in fact, violate it.

We also give a result that relates probabilistic query answering on MLD/subset PUDs to the traditional *consistent* query answering [Ber11, ABC99] on cardinality repairs. A fundamental difference between the two is that in the latter, only the cardinality repairs determine the result (i.e., the set of consistent answers), while in the former, *every subset* of tuples is a possible world that can take a portion of the probability of an answer. We show that, under symmetry assumptions, the consistent answers for cardinality repairs coincide with the answers that approach (at the limit) %100 certainty as *(a)* the weight of the constraints approaches infinity,

and (b) the error probability (i.e., the probability that the realizer introduces new tuples) approaches zero.

For learning, we focus on MLD/update PUDs and establish the following. In the presence of labeled examples, learning the parameters of a PUD translates to a convex optimization problem; consequently, this problem can be solved in polynomial time *if* probabilistic inference (computation of marginal probabilities) is tractable. For example, we show that when tuples are are independent, learning is solvable in polynomial time via standard optimization methods such as Stochastic Gradient Descent (SGD). In fact, we show that SGD is (with high probability) guaranteed to recover the true parameters of a tuple-independent PUD within error ϵ with access to $O(\epsilon^{-2})$ labeled examples.

More importantly, we establish novel results on when the intention of a PUD can be learned from a *single dirty database* without access to any labeled clean examples. Specifically, we demonstrate that under low-noise conditions (i.e., the realizer does not cause too much corruption in the intended data), learning the intention model corresponds to a convex optimization problem. We also present results on when this convex optimization problem can be solved in polynomial time.

Organization. The remainder of the paper is organized as follows. We begin with preliminary definitions in Section 2. In Section 3 we present the concept of a PUD, including its instantiations as MLD/subset and MLD/update PUDs. We present the three fundamental problems in Section 4, and describe the results of our preliminary analysis in Section 5. We conclude with a discussion in Section 6. For space limitations, some of the proofs are in the Appendix.

2. PRELIMINARIES

We discuss preliminary definitions and terminology that we use throughout the paper.

2.1 Relational Databases

A *relation signature* (or simply *signature* for short) is a finite sequence $\alpha = (A_1, \ldots, A_k)$ of distinct *attributes*. When there is no risk of ambiguity, we view α as a *set*, rather than *sequence*, of attributes; hence, we may say $A \in \alpha$ to denote that A is an attribute that occurs in α. A *tuple* t over a signature α is a maps the attributes of α to atomic values (e.g., numbers and strings). We write $t.A$ instead of $t(A)$. If $\alpha = (A_1, \ldots, A_k)$, then we identify a tuple t over α with the sequence $(t.A_1, \ldots, t.A_k)$.

A *relation* over α is a finite set of tuples over α. A *relational schema* (or just *schema* for short) is a finite set $\mathbf{S}$ of *relation symbols*, where every relation symbol R is associated with a signature that we denote by $sig(R)$. We say that R is *k-ary* if $sig(R)$ consists of k attributes. A *database* D over a schema $\mathbf{S}$ associates with each relation symbol R a relation over $sig(R)$; we denote this instance by R^D. To facilitate the modeling of value updates, we assume that D associated with every tuple t of a unique *tuple identifier*. We denote by $ids(D)$ and $ids_R(D)$ the sets of all identifiers of tuples of D and R^D, respectively. For $i \in ids_R(D)$ we denote by $D[i]$ and $R^D[i]$ the tuple with identifier i. Hence, it holds that $R^D = \{R^D[i] \mid i \in ids_R(D)\}$. We also consider two versions of *weighted* databases:

- A *tuple-weighted* database D is one where every tuple identifier $i \in ids(D)$ has a weight $w_D(i)$.
- A *value-weighted* database D is one where every tuple identifier $i \in ids_R(D)$ is associated with a weight $w_D^{R.A}(i)$ for every relation symbol R and attribute A of R.

Unweighted databases are considered weighted databases with all weights being one.

A *query* Q over a schema $\mathbf{S}$ is associated with a relation signature $sig(Q)$ and maps every database D over $\mathbf{S}$ onto an instance over $sig(Q)$; we denote this instance by $Q(D)$. A query Q is *k-ary* if $sig(Q)$ is of length k. A query Q of arity zero is said to be *Boolean*, and in this case the result $Q(D)$ is either empty, corresponding to **false**, or the singleton $\{()\}$, corresponding to **true**. The statements $Q(D) = \{()\}$ and $Q(D) = \emptyset$ are also written as $D \models Q$ and $D \not\models Q$, respectively.

2.2 Constraints and Minimum Repairs

Let D and D' be databases over the same schema. We say that D' is a *subset of* D, denoted $D' \subseteq D$, if D' is obtained from D by deleting tuples; that is, for all relation symbols R we have $ids_R(D') \subseteq ids_R(D)$, and for all $i \in ids(D')$ we have $R^{D'}[i] = R^D[i]$. We say D' is an *update* of D, denoted $D \approx D'$, if D' is obtained from D by changing cell values; that is, for all relation symbols R we have $ids_R(D') = ids_R(D)$. We assume that weights do not change in subsets and updates.

Let $\mathbf{S}$ be a schema, and let φ be a set of integrity constraints defined over $\mathbf{S}$. For example, φ can be a set of *Functional Dependencies* (FDs), *conditional FDs* [BFG+07], *denial constraints* (DCs) [GGM92], or referential constraints [Dat81]. If D *satisfies* φ, the we say that D is *consistent*. A *minimum subset repair* of D is a consistent subset D' of D with a minimal weighted difference $\sum_{i \in ids(D) \setminus ids(D')} w_D(i)$. A *minimum update repair* of D is a consistent update D' of D with a minimal $H(D, D')$, where $H(D, D')$, the (weighted) *Hamming distance* between D and D', is given by

$$H(D, D') \stackrel{\text{def}}{=} \sum_{R.A} \sum_{\substack{i| \\ D[i].A \neq D'[i].A}} w_D^{R.A}(i)\,.$$

In this paper, a DC φ is phrased in Tuple Relational Calculus (TRC) as

$$R_1(X_1) \wedge \cdots \wedge R_k(X_k) \;\Rightarrow\; \neg\psi(X_1, \ldots, X_k)$$

where each R_i is a relation symbol of $\mathbf{S}$, and ψ is a conjunction of comparisons of the form $X_i.A\ \theta\ X_j.B$ for a comparison operator θ (e.g., $=$, $\neq$, $<$, and so on). We may denote φ by $\varphi(X_1, \ldots, X_k)$ to specify its variables. For example, the FD $R : A \to B$ is expressed as the DC

$$R(X_1) \wedge R(X_2) \Rightarrow \neg(X_1.A = X_2.A \wedge X_1.B \neq X_2.B)\,.$$

A *violation* of a DC $\varphi(X_1, \ldots, X_k)$ in a database D is a sequence $(i_1, \ldots, i_k)$ of tuple identifiers of D such that the statement $\varphi(D[i_1], \ldots, D[i_k])$ is a false.

2.3 Probability Spaces

All probability spaces we use here are discrete. Formally, a *probability space* (or *distribution*) is a pair (Ω, π), where Ω is a countable set, called the *sample space*, and $\pi : \Omega \to [0,1]$, called the *probability mass function*, is a function such that $\sum_{o\in\Omega} \pi(o) = 1$. Each element in the sample space is called a *sample*. Let $\mathcal{P} = (\Omega, \pi)$ be a probability space. For $o \in \Omega$ we may denote the probability $\pi(o)$ by $\mathcal{P}(o)$. As conventional, if $\psi : \Omega \to \{\mathbf{true}, \mathbf{false}\}$ is a Boolean condition over samples, then $\Pr_{x\sim\mathcal{P}}(\psi(x))$ denotes the sum of $\pi(o)$ over all the samples $o \in \Omega$ with $\psi(o) = \mathbf{true}$. We also use the conventional notation of $\Pr_{x\sim\mathcal{P}}(\psi(x) \mid \varphi(x))$ to denote the probability of $\psi(x)$ *conditioned on* $\varphi(x)$.

2.4 Probabilistic Databases

A *probabilistic database* over a schema $\mathbf{S}$ is a probability space of (ordinary) databases over $\mathbf{S}$. We will use several representations of probabilistic databases. Throughout this section we fix the schema $\mathbf{S}$.

To illustrate our conceptual framework, the basic representation that we use for a probabilistic database is what we call a *Markov Logic Database* (MLD). Such a probabilistic database allows for uncertainty at the attribute level (via tuple generators), at the tuple level (via a tuple-selection probability), and at the database level (via Markov Logic). Next, we introduce this representation system. We begin with some notation.

Value and tuple generators. We first define mechanisms for generating random attribute values and random tuples. Let $\mathbf{S}$ be a schema.

- A *value generator* (*for* $\mathbf{S}$) is a function VG that maps every relation R and attribute A of R into a distribution, denoted $\mathsf{VG}_{R.A}$, over atomic values.
- A *tuple generator* (*for* $\mathbf{S}$) is a function TG that maps every relation symbol R into a distribution, denoted TG_R, over tuples of R.

One way of defining a tuple generator TG is via a value generator VG—a tuple in R is generated by independently sampling a value from $\mathsf{VG}_{R.A}$ for each attribute A. Nevertheless, in general, a tuple generator may introduce correlations across attributes. We do assume that, given a tuple t and an atomic value a, the probabilities $\mathsf{TG}_R(t)$ and $\mathsf{VG}_{R.A}(a)$ can be computed in polynomial time.

Identifier repository. Let $\mathbf{S}$ be a schema. An *identifier repository* (*for* $\mathbf{S}$) consists of pairwise-disjoint collections of tuple identifiers for each relation symbol. Formally, an identifier repository is a function ϱ that maps every relation symbol R into a finite set, denoted ϱ_R, of tuple identifiers, such that $\varrho_R \cap \varrho_S = \emptyset$ whenever $R \neq S$.

Coin allocation. Let $\mathbf{S}$ be a schema. A *per-table coin allocation* (*for* $\mathbf{S}$) is a function p that maps every relation symbol R into a probability (i.e., a number in $[0,1]$), denoted p_R. A *per-column coin allocation* (*for* $\mathbf{S}$) is a function p that maps every relation symbol R and attribute A of R into a probability, denoted $p_{R.A}$.

We can now define concrete representations of probabilistic databases.

TIID. Let $\mathbf{S}$ be a schema. A *database with independent, identically distributed tuples*, or *TIID* for short, is a triple $(\varrho, p, \mathsf{TG})$ such that ϱ is an identifier repository, p is a per-table coin allocation, and TG is a tuple generator. Semantically, a TIID $\mathcal{P} = (\varrho, p, \mathsf{TG})$ defines a probabilistic database via the following generative process. For each identifier i in ϱ, randomly choose, with probability p, whether or not to generate a tuple for i; if so, then generate the tuple using TG. More formally, the possible worlds are the databases D with $ids_R(D) \subseteq \varrho_R$ for all relation symbols R, and the probability $\mathcal{P}(D)$ of D is defined as

$$\mathcal{P}(D) \stackrel{\text{def}}{=\!=} \prod_{R\in\mathbf{S}} \mathcal{P}(R^D)$$

where each $\mathcal{P}(R^D)$ is given by

$$\mathcal{P}(R^D) \stackrel{\text{def}}{=\!=} \prod_{i\in ids_R(D)} (p_R \cdot \mathsf{TG}_R(D[i])) \times \prod_{\substack{i\in\varrho_R\setminus\\ ids_R(D)}} (1 - p_R)\,.$$

MLD. A *Markov Logic Database*, abbreviated *MLD*, is essentially a TIID with *weak constraints* that affect the probability of every possible world in the sense of Markov logic—every violation of a rule adds a factor to the probability. For the formal definition, let $\mathbf{S}$ be a schema. An MLD is a triple $(\mathcal{P}, \Phi, w)$ where:

- $\mathcal{P}$ is a TIID.
- Φ is a finite set of DCs.
- w is a *penalty function* that associates with every formula $\varphi \in \Phi$ a positive weight $w(\varphi)$.

Semantically, an MLD $\mathcal{M} = (\mathcal{P}, \Phi, w)$ defines a probabilistic database as follows: The possible worlds are those of $\mathcal{P}$. The probability $\mathcal{M}(D)$ of a possible world D is given by

$$\mathcal{M}(D) \stackrel{\text{def}}{=\!=} \frac{1}{Z} \times \mathcal{P}(D) \times \prod_{\varphi\in\Phi} e^{-w(\varphi)\cdot|V(D,\varphi)|}$$

where $V(D,\varphi)$ is the set of all violations of φ in D, and Z is a *normalization factor* (a.k.a. *partition function*) that normalizes the total sum of probabilities to one. We use the convention of $w(\varphi) = \infty$ to denote a *hard* constraint on which satisfaction we insist. (More formally, a hard constraint φ effectively restricts the probability space to be conditional on φ.)

3. PROBABILISTIC UNCLEAN DBS

We present our formal framework of unclean databases and example instantiations of our framework.

3.1 A Two-Actor Model For Unclean Data

We introduce a formal framework to model the process by which unclean data is generated. Our framework consists of two central notions: *(1)* an *intention* data-generation model, and *(2)* a noisy *realization* model. The intention model $\mathcal{I}$ corresponds to a data generating process that takes as input an identifier repository and generates a valid database I, typically with a bias

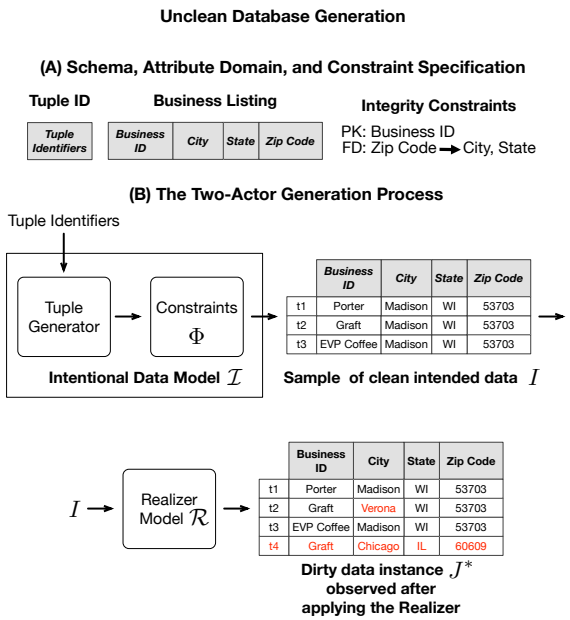


Figure 1: An example of a data generating PUD.

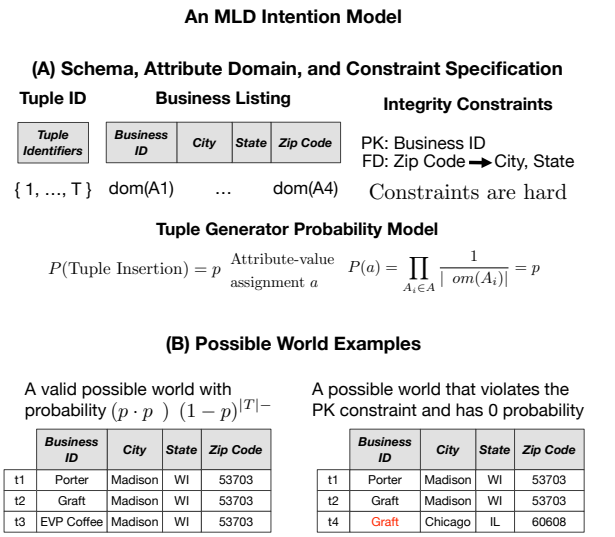


Figure 2: An example of a probabilistic intention model for the Database in our running example.

towards a collection Φ of constraints (e.g., DCs). The realization model $\mathcal{R}$ corresponds to a *noisy channel* that takes as input the valid database I and changes it in a stochastic manner to the instance $J^\star$ that we actually observe. Figure 1 illustrates this process.

Example 3.1. We use a running example from business listings. Figure 1(A) depicts the schema $\mathbf{S}$ of the example. The constraints consist of a primary key and a functional dependency. Figure 1(B) depicts the two-actor data generation process. First itention $\mathcal{I}$ outputs a valid database I containing three tuples. The realizer $\mathcal{R}$ takes as input this instance I, injects the new tuple $t4$ and updates the `City` value of tuple $t2$ from 'Madison' to 'Verona'. □

We consider processes $\mathcal{I}$ and $\mathcal{R}$ to be probabilistic. Specifically, $\mathcal{I}$, referred to as the *intention*, is a probability space over instances I of $\mathbf{S}$, that is, a probabilistic database over $\mathbf{S}$. Moreover, $\mathcal{R}$, referred to as the *realizer*, maps every possible I to a probabilistic database $\mathcal{R}_I$ over $\mathbf{S}$.

Example 3.2. Figure 2 (A) shows an instantiation of the intention $\mathcal{I}$ for our running example as a simple MLD that for each tuple identifier decides, independently with a fixed probability p, whether to include the tuple. The attribute values of tuples are assigned uniformly at random. The illustrated intention corresponds to an MLD that enforces the constraints as hard constraints. Figure 2 (B) shows two possible worlds and their probabilities. □

We also define the realization model $\mathcal{R}$, referred to as the *probabilistic realizer* hereafter, to be a function that maps every instance I of $\mathbf{S}$ to a probabilistic database $\mathcal{R}_I$ over $\mathbf{S}$. Finally we denote by J the *observed* or *unclean* database sampled by $\mathcal{R}_I$. Database J is an instance over $\mathbf{S}$.

Example 3.3. Figure 3 (A) describes a realizer that iterates over each cell in the input database instance I, independently with probability p (the coin allocation) changes the initial value of the cell to a new value. The new value is selected uniformly at random from the domain of the corresponding attribute. This is an example of a value generator. Figure 3 (B) shows two possible worlds sampled from this probabilistic database and their corresponding probabilities. □

3.2 Formal Definition

We now introduce the definition of a *Probabilistic Unclean Database* (PUD).

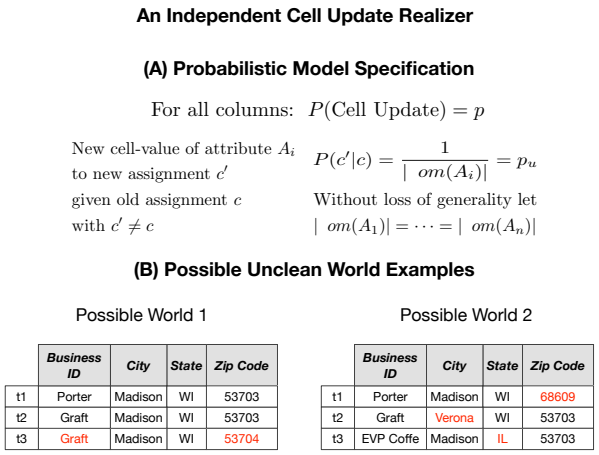
Definition 3.4. *Let* $\mathbf{S}$ *be a schema. A PUD (over* $\mathbf{S}$*) is a triple* $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ *where:*

1. $\mathcal{I}$ *is a probabilistic database referred to as the* intention*;*
2. $\mathcal{R}$*, the* realizer*, 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.*

A PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ is a probabilistic database over $\mathbf{S}$ with the following data generating process. First, sample an instance I from $\mathcal{I}$; this instance is what we consider the "intended" database that we seek to find. Yet, we do not see I, but rather the instance $J^\star$ that is sampled from $\mathcal{R}_I$ by possibly introducing noise into I.

More formally, we denote by $\mathcal{R}_\mathcal{I}$ the bivariate distribution over pairs of instances of $\mathbf{S}$, where the probability of each pair (I, J) is given by:

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



	Business ID	City	State	Zip Code
t1	Porter	Madison	WI	53703
t2	Graft	Madison	WI	53703
t3	Graft	Madison	WI	53704

	Business ID	City	State	Zip Code
t1	Porter	Madison	WI	68609
t2	Graft	Verona	WI	53703
t3	EVP Coffe	Madison	IL	53703

Figure 3: An example of a realizer for the database in our running example. Both illustrated possible worlds are noisy realizations of intended possible world 1 shown in Figure 2.

In the probabilistic database defined by $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$, the probability of each instance I is given by

$$\mathcal{U}(I) \stackrel{\text{def}}{=\!=} \Pr_{(I',J')\sim\mathcal{R}_\mathcal{I}}(I' = I \mid J' = J^\star)\,;$$

that is, the probability conditioned on the random J being $J^\star$. In other words, we have the following.

$$\mathcal{U}(I) = \frac{\mathcal{R}_\mathcal{I}(I, J^\star)}{\sum_{I'} \mathcal{R}_\mathcal{I}(I', J^\star)}$$

For $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ to be well defined, we require $J^\star$ to have nonzero probability; that is, there exists I such that $\mathcal{R}_\mathcal{I}(I, J^\star) > 0$.

3.3 Example Instantiations of PUDs

Our definition of a PUD is abstract, and not associated with any specific representation model. We now present several examples of concrete representations of PUDS that we refer to later in the paper.

MLD/Subset PUD. We define an *MLD/subset PUD* to be such that the intended database I is produced by an MLD, and the realizer may choose the produce some of the tuples that I decided to exclude. For the formal definition, let $\mathbf{S}$ be a schema. An MLD/subset PUD is a triple $(\mathcal{D}, \mathsf{TG}, p, J^\star)$, where:

- $\mathcal{D}$ is an MLD;
- TG is a tuple generator;
- p is a per-table coin allocation;
- $J^\star$ is a database.

Semantically, the MLD/subset PUD $(\mathcal{D}, \mathsf{TG}, p, J^\star)$, with $\mathcal{D} = (\mathcal{P}, \Phi, w)$ and $\mathcal{P} = (\varrho, p', \mathsf{TG}')$, represents the PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, where $\mathcal{I}$ is the probability space defined by $\mathcal{D}$, and for all databases I the realizer $\mathcal{R}$ reconsiders all missing identifiers i from ϱ by deciding, using p, whether to insert a tuple for i, and if so, then the tuple is generated using TG. Therefore, $\mathcal{R}_\mathcal{I}(I, J)$ is given by

$$\mathcal{D}(I) \times \prod_{R\in\mathbf{S}} \Big(\prod_{\substack{i\in \\ ids_R(J)\setminus \\ ids_R(I)}} p_R \cdot \mathsf{TG}(J[i]) \Big) \times \Big(\prod_{\substack{i\in\varrho_R\setminus \\ ids_R(J)}} (1 - p_R) \Big)\,.$$

MLD/Update PUD. We define an *MLD/update PUD* to be such that the intended database I is produced by an MLD, and the realizer traverses every cell, randomly selects whether or not to change the cell, and if so selects a random value as replacement. Formally, an MLD/update PUD is a tuple $(\mathcal{D}, \mathsf{VG}, p, J^\star)$ where:

- $\mathcal{D}$ is an MLD;
- VG is a value generator;
- p is a per-atribute coin allocation;
- $J^\star$ is a database.

Semantically, the MLD/update PUD $(\mathcal{D}, \mathsf{VG}, p, J^\star)$ represents the PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, where $\mathcal{I}$ is the probability space defined by $\mathcal{D}$, and for all databases I the realizer $\mathcal{R}$ considers every tuple identifier i of I and each value in $I[i]$, randomly selects, using p, whether or not to change that value, and if so then it selects a new value using VG. Hence, $\mathcal{R}_\mathcal{I}(I, J)$ is given by

$$\mathcal{D}(I) \times \prod_{R\in\mathbf{S}} \prod_{A\in R} \left(\prod_{\substack{i\in ids_R(\mathcal{J})| \\ I[i].A\neq J[i].A}} p_{R.A} \cdot \mathsf{VG}_{R.A}(I[i]) \right)$$
$$\times \left(\prod_{\substack{i\in ids_R(J)| \\ I[i].A=J[i].A}} \Big(1 - p_{R.A} + p_{R.A} \cdot \mathsf{VG}_{R.A}(J[i])\Big) \right)$$

Example 3.5. Figure 3 illustrates an MLD/Update PUD. As shown, the per-relation coin allocation corresponds to a constant probability p for all attributes. The value generator VG of the realizer of this PUD corresponds to a uniform distribution over the finite domain of each attribute. Figure 3 shows two possible worlds sampled by the depicted PUD. According to our definitions the probability of our probability of the first world corresponds to $\mathcal{D}(I)\times(p\cdot p_u)^2\times(1-p+p\cdot p_u)^{10}$. □

4. COMPUTATIONAL PROBLEMS

We now define three computational problems over PUDs that are motivated by the need to clean and query unclean data, as well as to learn parameters of the underlying statistical models.

Cleaning. The first problem is computing a Most Likely Intension (MLI), serving as a best selection of a cleaning outcome (assuming we subscribe to the involved statistical models). We refer to this problem as *cleaning*.

Definition 4.1 (Cleaning). *Let $\mathbf{S}$ be a schema. An MLI of a PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ is a database I such that $\mathcal{U}(I)$, or equivalently $\mathcal{R}_\mathcal{I}(I, J^\star)$, is maximal. For a representation system $\mathbf{R}$ of PUDs, the problem $(\mathbf{S}, \mathbf{R})$-*cleaning *is that of computing an MLI of a given PUD.*

Probabilistic query answering. Recall that a PUD defines a probabilistic database—a probability space over the intensions I. The problem of *Probabilistic Query Answering*, or *PQA* for short, is that of evaluating a query over this probabilistic database. We adopt the standard semantics of query evaluation over probabilistic databases [DS04, SORK11], where the confidence in an answer tuple is its marginal probability.

DEFINITION 4.2 (PQA). *Let* $\mathbf{S}$ *be a schema,* Q *a query over* $\mathbf{S}$*, and* $\mathbf{R}$ *a representation system of PUDs. The problem* $(\mathbf{S}, Q, \mathbf{R})$-PQA *is the following. Given a PUD* $\mathcal{U}$ *and a tuple* $\mathbf{a}$ *over* $sig(Q)$*, compute* confidence $\Pr_{I\sim\mathcal{U}}(\mathbf{a} \in Q(I))$ *of* $\mathbf{a}$.

PUD learning. In a PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, the components $\mathcal{I}$ and $\mathcal{R}$ are typically represented using numeric *parameters*. For example, the parameters involved in the representation of an MLD/subset PUD $(\mathcal{D}, \mathsf{TG}, p, J^\star)$ consist of the parameters needed for representing $\mathcal{D}$ (e.g., the weights of the DCs), the parameters that define the tuple generator TG, and the probability p itself. We refer to such a PUD as *parameterized* and denote it by $(\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$, where Ξ and Θ are the vectors of parameters involved in defining $\mathcal{I}$ and $\mathcal{R}$, respectively. A *parametric PUD* is a PUD in which the parameters are not initialized, and hence, represented as variables (or *symbolic* parameters). We denote a parametric PUD as $(\mathcal{I}_\mathbf{x}, \mathcal{R}_\mathbf{y}, J^\star)$, or just $(\mathcal{I}_\mathbf{x}, \mathcal{R}_\mathbf{y})$ if $J^\star$ is irrelevant, where $\mathbf{x}$ and $\mathbf{y}$ are the corresponding vectors of variables.

Parameters are learned from training examples. In our case, the examples are pairs (I, J) that are assumed to be sampled i.i.d. (independently and identically distributed) from $\mathcal{R}^\star_{\mathcal{I}^\star}$ where $\mathcal{I}^\star = \mathcal{I}_{\Xi^*}$ and $\mathcal{R}^\star = \mathcal{R}_{\Theta^*}$ are the *real* (unknown) models. We adopt here the concept of *maximum likelihood estimation* for learning, where the goal is to find the parameters that best explain (i.e., maximize the probability) of the training examples.

DEFINITION 4.3 (LEARNING). *Let* $\mathbf{S}$ *be a schema, and* $\mathbf{R}$ *a representation system for parametrized PUDs. The problem* $(\mathbf{S}, \mathbf{R})$-learning *is: Given a parametric PUD* $(\mathcal{I}_\mathbf{x}, \mathcal{R}_\mathbf{y})$ *and a collection* $(I_1, J_1), \ldots, (I_n, J_n)$ *of training examples, find parameter vectors* Ξ *and* Θ *that maximize* $\prod_{i=1}^n \mathcal{R}_\mathcal{I}(I_i, J_i)$ *where* $\mathcal{I} = \mathcal{I}_\Xi$ *and* $\mathcal{R} = \mathcal{R}_\Theta$.

5. PRELIMINARY ANALYSIS

We discuss preliminary results and insights on the aforementioned computational problems. We focus on specific the representation systems for PUDs introduced by MLD/update and MLD/subset PUDs. A summary of our results on cleaning, PQA, and learning is presented in Table 1.

5.1 Cleaning

We first establish results on the problem of cleaning, that is, computing an MLI. Theorems 5.1 and 5.3 show how MLIs generalize traditional *minimum repairs*. Specifically, minimum repairs correspond to MLIs in settings where the intension constraints are essentially

Table 1: A summary of our preliminary results for Cleaning (MLI), PQA, and Learning for MLD/subset and MLD/update PUDs.

	MLD/subset PUD	MLD/update PUD
Cleaning (MLI)	*(1)* Connections to minimum repairs [Thm 5.1] *(2)* MLI is in P-time for key constraints [Thm 5.2]	Connections to minimum repairs [Thm. 5.3]
PQA	Connections to Consistent Query Answering [Thm. 5.4]	
Learning		*(1)* Convexity [Prop. 5.5, Thm. 5.10] *(2)* Complexity [Thm.5.6, Thm 5.10] *(3)* Convergence [Thm. 5.7]

hard contraints and, in the case of MLD/update PUDs, we have symmetry in the value and error generation models. Theorem 5.2 applies to MLD/subset PUDs, and it states that we can efficiently compute an MLI in the case of key constraints (either hard or weak).

5.1.1 MLD/Subset PUDs

Our first result relates cleaning in MLD/subset PUDs to the traditional minimum subset (a.k.a. *cardinality*) repairs. We consider an MLD/subset PUD $(\mathcal{D}, \mathsf{TG}, p, J^\star)$, where $\mathcal{D}$ is the MLD $\mathcal{D} = (\mathcal{P}, \Phi, w)$ and $\mathcal{P}$ is the TIID $(\varrho, p', \mathsf{TG}')$. For a relation symbol R and a tuple identifier $i \in ids(R^{J^\star})$, we use the following notation:

- $q(i)$ is the probability that tuple $J^\star[i]$ is generated by TG, assuming it is not in the intention.
$$q(i) \stackrel{\text{def}}{=\!=} p_R \cdot \mathsf{TG}_R(J^\star[i])$$
- $q'(i)$ is the probability that the tuple $J^\star[i]$ is generated by $\mathcal{P}$.
$$q'(i) \stackrel{\text{def}}{=\!=} p'_R \cdot \mathsf{TG}'_R(J^\star[i])$$
- $\overline{p'}(i)$ is the probability that $\mathcal{P}$ does not generate any tuple for i.
$$\overline{p'}(i) \stackrel{\text{def}}{=\!=} 1 - p'_R(i)$$

The following theorem states, intuitively, that the notion of an MLI in an MLD/subset PUD converges 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 5.1. *Let* $(\mathcal{D}, \mathsf{TG}, p, J^\star)$ *be an MLD/subset PUD where* $\mathcal{D}$ *is the MLD* $(\mathcal{P}, \Phi, w)$ *and* $\mathcal{P}$ *is the TIID* $(\varrho, p', \mathsf{TG}')$*. Suppose that for all* $i \in ids(D^\star)$ *we have* $q(i) < q'(i)/\overline{p'}(i)$*. There exists a number* M *such that if* $w_\varphi > M$ *for all* $\varphi \in \Phi$ *then the following are equivalent for all subsets* I *of* $J^\star$.

1. *I is an MLI.*

2. *I is a minimum subset repair of $J^\star$ w.r.t. Φ under the tuple weight $w_{J^\star}(i) = \log\big(q'(i)/(q(i)\cdot\overline{p'}(i))\big)$.*

Proof. We compute the probability of an MLI I. For a relation symbol R and $i \in \rho_R$, denote $\overline{p}(i) \stackrel{\text{def}}{=\!=} 1 - p_R$ and $\overline{p'}(i) \stackrel{\text{def}}{=\!=} 1 - p'_R$. We have the following.

$$\begin{aligned}\mathcal{R}_{\mathcal{I}}(I, J^\star) &= \mathcal{D}(I) \times \prod_{\substack{i \in ids(J^\star)\setminus \\ ids(I)}} q(i) \times \prod_{i \in \varrho \setminus ids(J^\star)} \overline{p}(i) \\ &\sim \mathcal{D}(I) \times \prod_{\substack{i \in ids(J^\star)\setminus \\ ids(I)}} q(i) \qquad (1)\end{aligned}$$

The proportionality is due to the fact that the third factor in the first multiplication is the same for all I. If w_φ is large enough for all $\varphi \in \Phi$, then we can assume that every I that satisfies Φ has a higher probability than every I that violates Φ. Hence, we can assume that the MLI I satisfies Φ, and the following holds.

$$\mathcal{D}(I) \sim \mathcal{P}(I) \qquad (2)$$

Moreover, we have the following.

$$\mathcal{P}(I) = \prod_{i \in I} q'(i) \times \prod_{i \in \rho \setminus ids(I)} \overline{p'}(i) \sim \prod_{i \in I} q'(i) \times \prod_{i \in ids(J^\star)\setminus ids(I)} \overline{p'}(i) \qquad (3)$$

Here the proportionality is due to the fact that we divide the probability by the same factor for all I. From (1), (2) and (3) we conclude the following.

$$\begin{aligned}\mathcal{R}_{\mathcal{I}}(I, J^\star) &\sim \prod_{i \in I} q'(i) \times \prod_{i \in ids(J^\star)\setminus ids(I)} (q(i)\cdot\overline{p'}(i)) \\ &= \left(\prod_{i \in ids(J^\star)} q(i)\cdot\overline{p'}(i)\right) \times \left(\prod_{i \in I} \frac{q'(i)}{q(i)\cdot\overline{p'}(i)}\right) \\ &\sim \prod_{i \in I} \frac{q'(i)}{q(i)\cdot\overline{p'}} \qquad (4)\end{aligned}$$

Hence, I is a most likely repair if and only if I satisfies Φ and maximizes (4), which is the same as maximizing the sum $\sum_{i \in ids(I)} \log\big(q'(i)/(q(i)\cdot\overline{p'}(i))\big)$, as claimed. □

It follows from Theorem 5.1 that, under the conditions of the theorem, the complexity of computing an MLI is the same as that of finding a cardinality repair. In the case of a fixed set of FDs, the complexity of the latter is well known due to a dichotomy result by Livshits and Kimelfeld [LK17].

The following theorem considers MLD/subsets PUDs, and 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 [KW17, AFM06, KP12]), 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.

Theorem 5.2. *Let $(\mathcal{D}, \mathsf{TG}, p, J^\star)$ be an MLD/subset PUD where $\mathcal{D}$ is the MLD $(\mathcal{P}, \Phi, w)$. If Φ consists of (at most) one key constraint per relation and no relation of $J^\star$ contains duplicate tuples, then an MLI can be computed in polynomial time.*

Proof. Our goal is to compute a subset I of $J^\star$ that maximizes $\mathcal{R}_{\mathcal{I}}(I, J^\star)$. The proof is driven by several observations. The first observation is that, as shown in the proof of Theorem 5.1, we can ignore tuple identifiers that are not in $ids(J^\star)$. Second, as an FD does not involve more than a single relation, different relations are probabilistically independent:

$$\mathcal{R}_{\mathcal{I}}(I, J^\star) \sim \prod_{R \in \mathbf{S}} \mathcal{R}_{\mathcal{I}}(R^I, R^{J^\star})$$

Therefore, we can find an I with a maximal probability by finding an R^I with a maximal probability for each R separately. So, we can assume that $\mathbf{S}$ has a single relation symbol. Let R be the single relation symbol.

Denote by $\text{keys}(J^\star)$ the set of all tuples that are obtained from $J^\star$ by projecting on the key attributes. For $k \in \text{keys}(J^\star)$, let $J^\star_k$ and $I^\star$ be the restrictions of $J^\star$ and $I^\star$, respectively, to the tuples with the key value k. Call the pair $(J^\star_k, I_k)$ a *block*. Since Φ consists of only the key contraint, we have that different blocks are, again, probabilistically independent:

$$\mathcal{R}_{\mathcal{I}}(I, J^\star) \sim \prod_{k \in \text{keys}(J^\star)} \mathcal{R}_{\mathcal{I}}(I_k, J^\star_k)$$

Therefore, we can optimize for each block separately. For simplification assume that $(J^\star, I)$ is a single block.

Suppose that $J^\star$ contains m tuples. Then the number of tuples in I is in $\{0, 1, \ldots, m\}$. We can compute the most likely I with ℓ tuples for all $\ell \in \{0, 1, \ldots, m\}$, compute the probability of each such I, and take the I with the maximal probability. So, let ℓ be fixed. We complete the proof by showing how to compute an I that maximizes $\mathcal{R}_{\mathcal{I}}(I, J^\star)$, as well as the probability itself, among all subsets I of $J^\star$ with ℓ tuples.

Suppose that $\mathcal{P}$ is the TIID $(\varrho, p', \mathsf{TG}')$. Recall the definition of $\mathcal{R}_{\mathcal{I}}(I, J^\star)$ for our MLD/subset PUD. Denote our key constraint by φ, and by $V(I, \varphi)$ is the set of all violations of φ in I.

$$R_{\mathcal{I}}(I, J^\star) = \frac{1}{Z} \times \mathcal{P}(I) \times e^{-w(\varphi)\cdot|V(I,\varphi)|} \times \prod_{\substack{i \in ids(J^\star) \\ \setminus ids(I)}} q(i)$$

Here, Z is the partition function, and $q(i)$ is as defined in the beginning of this section. Since Z is the same for all I, we can simply ignore it and maximize (and compute) the unnormalized probability. Moreover, for our fixed ℓ it holds that $|V(I, \varphi)|$ is precisely $\ell(\ell - 1)$, since $J^\star$ has no duplicate tuples and, so, every pair of tuples is a violation. Denote $C_\ell = e^{-w(\varphi)\ell(\ell-1)}$. It then suffices to maximize the following:

$$C_\ell \times \mathcal{P}(I) \times \prod_{i \in ids(J^\star) \setminus ids(I)} q(i)$$
$$= C_\ell \times \left(\prod_{i \in ids(I)} q'(i) \right) \times \left(\prod_{i \in ids(J^\star) \setminus ids(I)} \overline{p'}(i) \right)$$
$$\times \left(\prod_{i \in ids(J^\star) \setminus ids(I)} q(i) \right)$$
$$= C_\ell \times \left(\prod_{i \in ids(J^\star)} \overline{p'}(i) \cdot q(i) \right) \times \left(\prod_{i \in ids(I)} \frac{q'(i)}{\overline{p'}(i) \cdot q(i)} \right)$$

Note that the second factor in the last product is the same for all I, so we again ignore it. Denote the number $\frac{q'(i)}{\overline{p'}(i) \cdot q(i)}$ by $f(i)$. It then suffices to find a subset I of $J^\star$, having ℓ tuples, with a maximal $\prod_{i \in ids(I)} f(i)$. But these are simply ℓ identifiers i with the maximal $f(i)$; hence, we sort the i's by descending $f(i)$ and select the top ℓ. The probability is proportional to C_ℓ times the product of the $f(i)$. Note that our assumption on the tractability of the generating functions allows us to actually compute $f(i)$ in polynomial time. This concludes the proof. □

5.1.2 MLD/Update PUDs

We discuss MLD/update PUDs and establish a result analogous to Theorem 5.1. We consider an MLD/update PUD $(\mathcal{D}, \mathsf{VG}, p, J^\star)$, where $\mathcal{D}$ is the MLD $(\mathcal{P}, \Phi, w)$ and $\mathcal{P}$ is the TIID $(\varrho, p', \mathsf{TG}')$. We say that TG' is *symmetric* if for every relation schema $R(A_1, \ldots, A_k)$ there are value domains $V_1, \ldots, V_k$ such that TG' selects tuples uniformly from the product $V_1 \times \cdots \times V_k$. Moreover, we say that TG' can *freely update* $J^\star$ if each domain V_i contains the corresponding column of $J^\star$, and is large enough to allow us to replace every value of $J^\star$ with a unique value.

For a relation R and tuple identifier $i \in ids(R^{J^\star})$, we use the following notation.

- For a relation symbol R, an attribute A of R and a tuple identifier $i \in ids_R(J^\star)$, we denote by $eq_{R.A}(i)$ the probability that the realizer leaves the value $R^I[i].A$ unchanged.

  $$eq_{R.A}(i) \stackrel{\text{def}}{=\!=} 1 - p_{R.A} + p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i])$$

  Here, $1 - p_{R.A}$ is the probability the realizer does not touch the cell, and $p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i])$ is the probability that it decides to resample the cell, but the resulting value is the same as the initial value.

- We define $r_{R.A}(i)$ as follows.

  $$r_{R.A}(i) \stackrel{\text{def}}{=\!=} p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i]) / eq_{R.A}(i)$$

Note that $r_{R.A}(i)$ is in $[0, 1]$, since $p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i]) \geq \mathsf{VG}_{R.A}(J^\star[i]) / eq_{R.A}(i)$. We have the following result, which we prove in the appendix.

THEOREM 5.3. *Let $(\mathcal{D}, \mathsf{VG}, p, J^\star)$ be an MLD/update PUD where $\mathcal{D}$ is the MLD $(\mathcal{P}, \Phi, w)$ and $\mathcal{P}$ is the TIID $(\varrho, p', \mathsf{TG}')$. Suppose that TG' is symmetric and can freely update $J^\star$. There exists a number M such that if $w_\varphi > M$ for all $\varphi \in \Phi$ then the following are equivalent for all updates I of $J^\star$.*

1. *I is an MLI.*
2. *I is a minimum update repair of $J^\star$ w.r.t. Φ under the value weight $w^{R.A}_{J^\star}(i) = -\log(r_{R.A}(i))$.*

5.2 Probabilistic Query Answering

For probabilisitic query answering, we focus on the instance MLD/subset PUDs and draw the following connection to *consistent query answering* over cardinality repairs. Consider an MLD/subset PUDs where each of the two tuple generators TG and TG' assigns to all tuples (over all relations) the same probability. Moreover, suppose that the coin p' of the intention is constantly 0.5, and the coin p of the realizer is the same on all relations. We call such an MLD/subset *strongly symmetric*. Then, the consistent answers are precisely the tuples whose probability approaches one when all of the following hold: (1) the probability of introducing error approaches zero; and (2) the weight of the weak constraints approaches infinity (i.e., the constraints are strengthening towards hard constraints).

THEOREM 5.4. *Let $\mathcal{U}_{p,w} = (\mathcal{D}, \mathsf{TG}, p, J^\star)$ be a strongly symmetric MLD/subset PUD, where $\mathcal{D}$ corresponds to the MLD $(\mathcal{P}, \Phi, w)$ and $\mathcal{P}$ is the TIID $(\varrho, 0.5, \mathsf{TG}')$. Let Q be a query, and $\mathbf{a}$ a possible answer. The following are equivalent.*

1. *$\mathbf{a}$ is a consistent answer over the cardinality repairs, w.r.t. Φ.*
2. *$\lim_{p \longrightarrow 0} \lim_{w \longrightarrow \infty} \Pr_{I \sim \mathcal{U}_{p,w}}(\mathbf{a} \in Q(I)) = 1$.*

PROOF. Recall the definition of the (unnormalized) probability $\mathcal{R}_{\mathcal{I}}(I, J^\star)$ of I.

$$\mathcal{D}(I) \times \prod_{R \in \mathbf{S}} \Big(\prod_{\substack{i \in \\ ids_R(J^\star) \setminus \\ ids_R(I)}} p_R \cdot \mathsf{TG}(J^\star[i]) \Big) \times \Big(\prod_{\substack{i \in \varrho_R \setminus \\ ids_R(J^\star)}} (1 - p_R) \Big)$$
$$= \mathcal{D}(I) \times \left(\prod_{R \in \mathbf{S}} \prod_{\substack{i \in \varrho_R \setminus \\ ids_R(I)}} (1 - p_R) \right) \times$$
$$\Big(\prod_{R \in \mathbf{S}} \prod_{\substack{i \in \\ ids_R(J^\star) \setminus \\ ids_R(I)}} \frac{p_R \cdot \mathsf{TG}(J^\star[i])}{(1 - p_R)} \Big) \sim \mathcal{D}(I) \times \prod_{\substack{i \in \\ ids_R(J^\star) \setminus \\ ids_R(I)}} \frac{p_R \cdot \mathsf{TG}(J^\star[i])}{(1 - p_R)}$$

Due to symmetry, we can write $\mathcal{R}_{\mathcal{I}}(I, J^\star)$ as follows for some $C \in (0, 1)$.

$$\mathcal{R}_{\mathcal{I}}(I, J^\star) \sim \mathcal{D}(I) \cdot C^{|J^\star \setminus I|}$$

For any constant p, as $w \longrightarrow \infty$ the mass of $\mathcal{D}$ concentrates on the databases that satisfy Φ, or in other

words, as $w \longrightarrow \infty$ the probability of satisfying Φ approaches 1. Hence, for fixed any fixed p, the probability $\mathcal{R}_{\mathcal{I}}(I, J^\star)$ concentrates around the consistent subsets. And, since $p' = 0.5$, all consistent subsets are equally likely.

Now, as p approaches zero, so does C. Hence, if I is a cardinality repair and I' is a consistent subset that is not a cardinality repair, the ratio between $R_{\mathcal{I}}(I, J^\star)$ and $\mathcal{R}_{\mathcal{I}}(I', J^\star)$ approaches infinity as p approaches zero.

We conclude that the total probability of the cardinality repairs approaches one as p approaches zero, and all cardinality repairs have the same probability. In particular, if $\mathbf{a}$ is a consistent answer, then its probability approaches one. Conversely, if $\mathbf{a}$ is *not* a consistent answer, then there is a contant portion of the probability that is missing for every p—this is the probability of a cardinality repair I in which $\mathbf{a} \notin Q(I)$. □

5.3 PUD Learning

We now consider the problem of PUD learning. We focus on MLD/update PUDs and present results on the convergence of learning and its computational complexity. Below, we first describe the model of parametric PUDs we consider for our analysis—recall that PUD learning in Section 4 is defined over parametric PUDs. We then distinguish between two learning setups: First we consider the case of *supervised learning* where we have access to labeled clean data, i.e., access to a collection of $(I, J^\star)$ pairs. Then we relax this condition and provide results for *unsupervised learning* where we only have access to the realized dirty database $J^\star$. Specifically, we provide preliminary results on when we can learn a PUD from dirty data alone. All proofs are deferred to Appendix B.1.

5.3.1 Markov Parametric PUDs

We instantiate our abstract definition of a parametric PUD from Section 4 to a concrete representation model. To this end, we focus on Markov networks [KF09] and consider that both Intention $\mathcal{I}$ and Realizer $\mathcal{R}$ correspond to log-linear models [WJ08]. We follow the typical parametrization scheme associated with log-linear models [KF09].

A key concept in this scheme is that of *sufficient statistics* [WJ08]. Formally, any real-valued function $u(X)$ of the observations in a random sample $X = X_1, \cdots, X_n$ is called a *statistic*. A sufficient statistic is described as follows: Consider a set X of i.i.d. data conditioned on an unknown parameter θ. A statistic $u(X)$ (e.g., a count of certain properties of X) is sufficient for parameter θ if $\Pr(x \mid u(X), \theta) = \Pr(x \mid u(X))$. For example, for an MLD probabilistic database the number of violations for a denial constraint φ is one of the sufficient statistics.

Given a parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta)$, we assume the parameter vector Ξ and Θ to have fixed dimensions $|\Xi|$ and $|\Theta|$ respectively. We use functions $u_{\mathcal{I}} : \Omega \to \mathbb{R}^{|\Xi|}$ and $u_{\mathcal{R}} : \Omega \to \mathbb{R}^{|\Theta|}$ to denote *the sufficient statistics of intention $\mathcal{I}_\Xi$ and realizer $\mathcal{R}_\Theta$*. Given these functions we call a PUD $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$ *Markov parametric* if the probability for a pair $(I, J^\star)$ assigned by the intention and the realizer of $\mathcal{U}$ are given by:

$$\mathcal{I}_\Xi(I) = \frac{1}{Z} \exp\left(\Xi^T \cdot u_{\mathcal{I}}(I)\right)$$

$$\mathcal{R}_\Theta(J^\star|I) = \frac{1}{Z(I)} \exp\left(\Theta^T \cdot u_{\mathcal{R}}(J^\star|I))\right)$$

That is, probabilities $\mathcal{I}_\Xi(I)$ and $\mathcal{R}_\Theta(J^\star|I)$ correspond to *log-linear* functions of the parameter vectors Ξ and Θ and the sufficient statistics $u_{\mathcal{I}}$ and $u_{\mathcal{R}}$ respectively.

When all tuple and value generators are parametric log-linear models with form $\exp(\sum_k w_k \cdot f_k(X))$ where $f_k(X)$ are functions of the input variables X, and w_k are the model coefficients, *all PUD instantiations described in Section 3.3 can be expressed as Markov parametric PUDs*. We refer the reader to Appendix A for more details on this equivalence. For all results presented below we assume Markov parametric PUDs.

5.3.2 Learning in the Presence of Training Examples

We focus on the problem of PUD learning in the presence of a collection $(I_1, J_1), \ldots, (I_n, J_n)$ of training examples. We will use T to denote this collection of training examples. The key component in our learning problem is the likelihood function, which corresponds to the objective function of an optimization problem. We now discuss the form of the likelihood function for Markov parametric PUDs, its properties, and their computational implications.

Instead of optimizing the likelihood directly, we maximize the negative log-likelihood:

$$nll(\Xi, \Theta : T) = -\log \prod_{i=1}^{n} \mathcal{R}_{\mathcal{I}}(I_i, J_i)$$

We have the following proposition with respect to the curvature of negative log-likelihood.

PROPOSITION 5.5. *Given a collection of training examples $(I_1, J_1), \ldots, (I_n, J_n)$ and a Markov parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta)$ the negative log-likelihood is a convex function of the parameter vector (Ξ, Θ).*

This result implies that the optimization objective for PUD learning under Markov parametric PUDs has no local optima. It does not, however, imply the uniqueness of a global optimum. For any global optimum to the MLE optimization we have that the gradient of our negative log-likelihood objective is zero. We can compute the gradient of the negative log-likelihood as follows [KF09]:

$$\frac{\partial}{\partial \theta_i} \frac{1}{n} nll(\Xi, \Theta : \mathcal{D}) = \boldsymbol{E}_\theta[f_i] - \boldsymbol{E}_{\mathcal{D}}[f_i]$$

where f_i corresponds to the i-th element of the overall feature vector $f = (u_{\mathcal{I}}, u_{\mathcal{R}})$, parameter vector $\theta = (\Xi, \Theta)$, and $u_{\mathcal{I}}$ and $u_{\mathcal{R}}$ are the sufficient statistics describing parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta)$.

For the maximum likelihood estimate, we have that $\boldsymbol{E}_{\mathcal{D}}[f_i] = \boldsymbol{E}_{\hat{\theta}}[f_i]$. That is, for parameters $\hat{\theta}$ the value for each feature f_i (e.g., the violations of a constraints

ϕ) relative to P_θ matches its empirical expectation in D. Unfortunately, there is no analytical form for θ. It is typical to resort to iterative, gradient-based methods that iteratively take steps in the parameter space to improve the objective [Min03].

However, computing the gradient requires computing the difference between a features's empirical count in the data and its expected count relative to our current parameterization θ. To compute the gradient, a full inference step is required at every iteration of the gradient descent procedure. For general Markov PUDs the computational cost of inference can be prohibitively expensive since inference can be #P-hard. Approximate inference methods such as MCMC methods [BC93, SWM12] or Belief propagation [WJW03] can be used to estimate the gradient in the aforementioned optimization.

We focus on cases where inference is tractable and thus can solve PUD learning with guarantees. Specifically, we focus on Markov parametric PUDs that correspond to *MLD/update PUDs with independent tuples*, i.e., the MLD of the PUD corresponds to a tuple-independent MLD. This means that each DC φ is defined over a single free variable. Examples of such denial constraints correspond to restricted cases of conditional functional dependencies [BFG+07, FGJK08] over the attribute values of a single tuple, such as "age smaller than 10 cannot co-occur with a salary greater than \$100k" and "zip codes staring with 53 imply that the state is Wisconsin." We have the next theorem for tuple-independent MLD/update PUDs:

THEOREM 5.6. *Let* $\mathbf{S}$ *be a schema and* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta)$ *be a parametric tuple-independent MLD/update PUD. Given a collection* $T = (I_1, J_1), \ldots, (I_n, J_n)$ *of labeled examples, the negative log-likelihood can be written as*

$$nll(\Xi, \Theta : D) = \sum_{k=1}^{n} \sum_{i \in ids(I_k)} l(I_k[i], J_k[i]; \Xi, \Theta),$$

where l *is a convex function of* Ξ *and* Θ.

As a corollary of Theorem 5.6, we note that it is easy to solve the PUD learning problem in this class, because we can use sum-of-components convex optimization techniques such as *stochastic gradient descent* (SGD) [BV04].

COROLLARY 5.7. *Let* $\mathbf{S}$ *be a schema.* $(\mathbf{S}, \mathbf{R})$*-learning is solvable in polynomial time for the class* $\mathbf{R}$ *of parametric tuple-independent MLD/update PUDs.*

So far we have shown that we can solve PUD learning for parametric tuple-independent MLD/update PUDs efficiently. The next step is to bound the error of the parameter estimates $\theta = (\Xi, \Theta)$ obtained by solving PUD learning with respect to the optimal parameters $\theta^\star = (\Xi^\star, \Theta^\star)$ of the data generating PUD. We show that for parametric tuple-independent MLD/update PUDs the estimated error is bounded as a function of the size of the example databases.

THEOREM 5.8. *Let* $\mathbf{S}$ *be a schema and* $\mathcal{U} = (\mathcal{I}_{\Xi^\star}, \mathcal{R}_{\Theta^\star})$ *be a parametric tuple-independent MLD/update PUD. For training collections* T *sampled from* $\mathcal{U}$*, estimators* (Ξ, Θ) *are asymptotically normal:*

$$\sqrt{|T|}\,(\Xi - \Xi^\star) \to \mathcal{N}\left(0, \Sigma^2_{\Xi^\star}\right) \text{ as } |T| \to \infty$$
$$\sqrt{|T|}\,(\Theta - \Theta^\star) \to \mathcal{N}\left(0, \Sigma^2_{\Theta^\star}\right) \text{ as } |T| \to \infty$$

where $\Sigma^2_{\Xi^\star}$ *and* $\Sigma^2_{\Theta^\star}$ *correspond to the asymptotic variance of estimates* Ξ *and* Θ.

The detailed proof of this Theorem together as well as the definitions for $\Sigma^2_{\Xi^\star}$ and $\Sigma^2_{\Theta^\star}$ are presented in Appendix B.4 and follow the form of standard proofs for asymptotic normality [EH78]. Intuitively, the above theorem states that the MLE estimators not only converge to the true unknown parameters, but they converge fast enough, at a rate $1/\sqrt{|T|}$. Importantly, this result means that as the size of the database increases, the quality of our parameter estimates becomes arbitrarily good. That is, a size of $|T| = O(\epsilon^{-2})$ will suffice to achieve error ϵ.

5.3.3 Unsupervised Learning from a Dirty Database

Until now we focused on PUD learning in the presence of labeled clean data. We now show preliminary results on MLD/update PUDs for cases where we have no training data but we only have access to a *single dirty database*. In other words, we study the unsupervised version of PUD learning where we only get a parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$ and we want to recover parameters Ξ and Θ without any additional information such as labeled data. The results below provide preliminary insights into the conditions under which unsupervised PUD learning can be solved efficiently with guarantees on the estimation error for Ξ and Θ.

As with the supervised case, we first analyze the curvature of negative log-likelihood. Here, the negative log-likelihood is defined as:

$$nll(\Xi, \Theta : J^\star) = -\log\left(\sum_{I \in \Omega} \mathcal{I}_\Xi(I) \cdot \mathcal{R}_\Theta(J^\star | I)\right)$$

where $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ follow the form for Markov parametric PUDs introduced in Section 5.3.1. Similar to Section 5.3.2, we focus on MLD/update PUDs and study when the negative log-likelihood is convex.

For all unsupervised learning results we require that the realizer does not introduce too much noise. More formally, we require the following condition to hold for all unsupervised learning results:

Low-Noise Condition. Consider a class of parametric MLD/update PUDs $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$. We say that $\mathcal{U}$ satisfies the low-noise condition with respect to a probability p (e.g., $p = 0.01$) if it is the case that for every possible (I, J) in the class and every tuple identifier $i \in \varrho$ the realizer introduces an error into tuple $I[i]$ with probability at most p. That is, $\Pr(J[i] = I[i] | I) \geq 1 - p$.

This condition dictates an upper bound on the per-attribute coin allocation of the realizer of MLD/update

PUDs as a function of probability p. As a result low values of p enforce that the realizer does not introduce too much corruption in the intention database. We now show that for MLD/update PUDs and under the low-noise condition the negative log-likelihood is a convex function of the parameters Ξ of our realizer. condition we now show that :

THEOREM 5.9. *Given a schema* $\mathbf{S}$ *and a Markov parametric MLD/update PUD* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$ *where* Ξ *belongs in a compact convex set, there exists some fixed* $p > 0$ *such that whenever the low-noise condition holds with respect to* p*, the negative log-likelihood* $nll(\Xi, \Theta : J^\star)$ *is a convex function of* Ξ*.*

While this theorem states that under low noise the negative log-likelihood is a convex function of Ξ the overall negative log-likelihood is non-convex. Convexity only provides us with guarantees on the existence of a global optimum but does not give us guarantees on the complexity of learning. Next, we study the complexity of unsupervised PUD learning.

In general, we can optimize the negative log-likelihood using non-convex optimization methods [Ber99]. Specifically, if we assume simple update realizers, e.g., a realizer with the same coin allocation per attribute, and assume that p is bounded, we can solve PUD learning via simple methods such as stochastic gradient descent or quasi-Newton methods. These methods optimize for the single coin allocation parameter of the realizer via simple grid-search while solving a series of convex optimization problems for $nll(\Xi, \Theta : J)$ for different fixed instantiations of Θ. Recently, SGD has been shown to converge for similar simple non-convex problems [SRO15, MWCC17]. For the above to hold p should be bounded.

We now show that p is always bounded in the case of tuple-independent MLD/update PUDs but in the case of MLD/update PUDs with arbitrary denial constraints this is not always the case. From the proof of Theorem 5.9, we have that for intention $\mathcal{I}$ with sufficient statistics $u_\mathcal{I}$ and the low noise condition probability p:

$$p \cdot \max_{J \in \Omega} \|u_\mathcal{I}(J)\|^2 \prec \delta \cdot I$$

where $\delta > 0$ is a constant and $u_\mathcal{I}(J)$ is the sufficient statistics of $\mathcal{I}$ computed over J. For details please see Appendix B.1. For this condition to hold probability p may have to take a small value that depends on *the size of the tuple identifier repository* ϱ which might not be known. For example, in the presence of denial constraints, sufficient statistics $u_\mathcal{I}$ entail the number of violations for denial constraints, which in the worst case is proportional to the size of repository ϱ.

In contrast to generic MLD/update PUDs, we show that for tuple-independent PUDs p is independent of the size of repository ϱ. Moreover, due to tuple-independence the negative log-likelihood decomposes to a sum over a series of convex functions, thus, we can solve PUD learning efficiently (in polynomial time) with guarantees using standard sum-of-components convex optimization techniques such as SGD. We have that:

THEOREM 5.10. *Given a schema* $\mathbf{S}$ *and a parametric tuple-independent PUD* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$*, parameter* p *for Theorem 5.9 to hold is independent of the number of tuple identifiers in repository* ϱ *and the negative log-likelihood can be written as*

$$nll(\Xi, \Theta : J^\star) = \sum_{i \in ids(J^\star)} l_i(J^\star[i]; \Xi, \Theta),$$

where each l_i *is a convex function of* Ξ *(keeping* Θ *fixed).*

We focus again on MLD/update PUDs with denial constraints. From the expression on p and $\|u_\mathcal{I}(J)\|^2$ presented above, we have: If we assume that the number of denial constraint violations is bounded for any J sampled by a given PUD $\mathcal{U}$ then p will still be independent of the size of repository ϱ. Thus, we can use non-convex optimization methods as those for tuple-independent MLD/update PUDs to solve PUD learning. Finally, we point out that here the negative log-likelihood does not decompose to a sum over independent random variables, thus, computing the gradient (which requires performing inference) can be hard in the worst case. Nonetheless, approximate inference methods as the ones discussed in Section 5.3.2 can be used.

6. CONCLUDING REMARKS

Taking inspiration from our experience with the HoloClean system [RCIR17], we introduced the concept of a PUD that represents a probabilistic unclean database as the posterior probability space of a two-step generative model: an *intention* that generates a clean database, and a *realization* that introduces noise. We defined three fundamental problems in the framework: cleaning (comping an MLI), probabilistic query answering (marginal inference), and PUD learning (parameter estimation). We introduced the MLD/subset PUDs and MLD/update PUDs as instantiations of our framework. Our preliminary analysis focused on showing how these two PUD representations generalize their deterministic minimum-repair counterparts, MLI computation under (weak) primary-key constraints, and PUD learning along with associated guarantees of convexity that allow to deploy standard learning toolkits.

This paper opens up many research directions for future exploration. One is investigating the complexity of cleaning in more general configurations than the ones covered here. Moreover, in cases where finding an MLI is computationally hard, it is of natural interest to find *approximate* MLIs 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 certain fact). Finally, an important direction is to devise learning algorithms for more general cases than those covered 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 generic noisy realizers than the ones discussed in this paper.

7. REFERENCES

[AAB+16] 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.

[ABC99] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM, 1999.

[AFM06] Periklis Andritsos, Ariel Fuxman, and Renée J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, page 30. IEEE Computer Society, 2006.

[AK09] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM, 2009.

[BC93] 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.

[Ber99] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[Ber11] Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[BFG+07] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755. IEEE, 2007.

[BHS+07] 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.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CIP13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[CM05] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1):90–121, 2005.

[Cod75] E. F. Codd. Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

[Cro16] CrowdFlower. Data Science Report. http://visit.crowdflower.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf, 2016.

[Dat81] C. J. Date. Referential integrity. In *VLDB*, pages 2–12. VLDB Endowment, 1981.

[DS04] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

[EH78] Bradley Efron and David V Hinkley. Assessing the accuracy of the maximum likelihood estimator: Observed versus expected fisher information. *Biometrika*, 65(3):457–483, 1978.

[FGJK08] 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.

[FKK15] 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.

[GdBS14] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. In *BUDA*, 2014.

[GGM92] Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

[GRSY15] Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How hard is inference for structured prediction? In *ICML*, pages 2181–2190. JMLR.org, 2015.

[Ily16] Ihab F. Ilyas. Effective data cleaning with continuous evaluation. *IEEE Data Eng. Bull.*, 39:38–46, 2016.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[KL09] Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361, pages 53–62. ACM, 2009.

[KP12] 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.

[Kul97] Solomon Kullback. *Information theory*

and statistics. Courier Corporation, 1997.

[KW17] 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.

[LB07] 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.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, New York, NY, USA, 2002. ACM.

[Lib14] Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *PODS*, pages 1–13, New York, NY, USA, 2014. ACM.

[LK17] Ester Livshits and Benny Kimelfeld. The complexity of computing a cardinality repair for functional dependencies. *CoRR*, abs/1708.09140, 2017.

[Min03] Thomas P. Minka. Algorithms for maximum-likelihood logistic regression. *Technical Report, Carnegie Mellon University*, 2003.

[MWCC17] 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 arXiv:1711.10467*, 2017.

[Pea89] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[RCIR17] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011.

[SRO15] 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.

[SWM12] 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.

[WJ08] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, January 2008.

[WJW03] 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.

[WT14] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468. ACM, 2014.

[YEN+11] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

APPENDIX

A. MLD PUDS AS MARKOV PARAMETRIC PUDS

We demonstrate how parametric MLD/update PUDs can be written as Markov parametric PUDs. Recall that:

$$\mathcal{R}_{\mathcal{I}}(I,J) = \mathcal{I}_{\Xi}(I) \cdot \mathcal{R}_{\Theta}(J|I)$$

We first focus on intention $\mathcal{I}_{\Xi}$ and consider that it corresponds to an MLD $\mathcal{I} = (\mathcal{P}, \Phi, w)$. We show that such an MLD follows the general form of Markov parametric PUDs when the tuple generator TG_R for every relation symbol R in TIID $\mathcal{P}$ of $\mathcal{M}$ is a parametric log-linear model with:

$$\mathsf{TG}_R[D[i]] = \exp\left(\sum_{k_R} w_{k_R} \cdot f_{k_R}(D[i])\right)$$

Recall that for an MLD $\mathcal{I} = (\mathcal{P}, \Phi, w)$ we have that:

$$\mathcal{I}(I) \stackrel{\text{def}}{=\!=} \frac{1}{Z} \times \mathcal{P}(I) \times \prod_{\varphi \in \Phi} e^{-w(\varphi)\cdot|V(I,\varphi)|}$$

where $Z = \sum_{I' \sim \Omega} \prod_{\varphi \in \Phi} e^{-w(\varphi)\cdot|V(I',\varphi)|}$. First, by following the log-exp re-write rule we can rewrite $\mathcal{P}(I)$ as:

$$\mathcal{P}(I) = \exp\left(\sum_{R \in \mathbf{S}} \left(\log(p_R)|ids_R(I)| + \sum_{i \in ids_R(I)} \sum_{k_R} w_{k_R} \cdot f_{k_R}(I[i]) + \log(1-p_R)|\varrho_R \setminus ids_R(I)|\right)\right)$$

We can also rewrite:

$$\prod_{\varphi \in \Phi} e^{-w(\varphi)\cdot|V(I,\varphi)|} = \exp\left(\sum_{\varphi \in \Phi} -w(\varphi) \cdot |V(I,\varphi)|\right)$$

Eventually we can write $\mathcal{I}(I)$ as:

$$\mathcal{I}(I) = \frac{1}{Z} \exp \sum_{R \in \mathbf{S}} \left(\log(p_R)|ids_R(I)| + \sum_{i \in ids_R(I)} \sum_{k_R} w_{k_R} \cdot f_{k_R}(I[i]) + \log(1-p_R)|\varrho_R \setminus ids_R(I)|\right)$$
$$\times \exp \sum_{\varphi \in \Phi} (-w(\varphi) \cdot |V(I,\varphi)|)$$

Given the above, concatenating all parameters $\log(p_R)$, w_{k_R}, $\log(1-p_R)$, and $-w(\phi)$ into a single vector we obtain the parameter vector Ξ of intension $\mathcal{I}$, while quantities $|ids_R(I)|$, $f_{k_R}(\cdot)$, $|i \in \varrho_R \setminus ids_R(I)|$, $|V(I,\varphi)|$ form the sufficient statistics for intention $\mathcal{I}$.

We now focus on the realizer. First we consider a subset realizer. Recall that:

$$\mathcal{R}(J,I) = \prod_{R \in \mathbf{S}} \Big(\prod_{\substack{i \in \\ ids_R(J) \setminus \\ ids_R(I)}} p_R \cdot \mathsf{TG}(J[i]) \Big) \times \Big(\prod_{\substack{i \in \varrho_R \setminus \\ ids_R(J)}} (1-p_R) \Big).$$

Following the log-exp rewrite trick we have that:

$$\mathcal{R}(J|I) = \exp\left(\sum_{R \in \mathbf{S}} \left(\log(p_R)|i \in ids_R(J) \setminus ids_R(I)| + \sum_{\substack{i \in \\ ids_R(J) \setminus \\ ids_R(I)}} \sum_{k_R} w_{k_R} \cdot f_{k_R}(J[i]) + \log(1-p_R)|i \in \varrho_R \setminus ids_R(J)|\right)\right)$$

Given the above, concatenating all parameters $\log(p_R)$, w_{k_R}, and $\log(1-p_R)$ into a single vector we obtain the parameter vector Θ of realizer $\mathcal{R}$, while quantities $|i \in ids_R(J) \setminus ids_R(I)|$, $f_{k_R}(\cdot)$, and $|i \in \varrho_R \setminus ids_R(J)|$ form the sufficient statistics of realizer $\mathcal{R}$. It is easy to see that taking the product of $\mathcal{I}_I$ and $\mathcal{R}(J|I)$ corresponds to a Markov parametric PUD.

A similar rewriting process can be followed for update realizers described in Section 3.3. The only difference is that we require that the value generator VG corresponds to a parametric log-linear model with:

$$\mathsf{VG}_R.A[D[i]] = \exp\left(\sum_{k_{R.A}} w_{k_{R.A}} \cdot f_{k_{R.A}}(D[i])\right)$$

Given this we have the following rewriting for an update realizer:

$$\mathcal{R}(J|I) = \exp \sum_{R\in\mathbf{S}} \sum_{A\in R} (\log(p_{R.A})|i \in ids_R(J) \mid I[i].A \neq J[i].A|) \times \exp \sum_{R\in\mathbf{S}} \sum_{A\in R} \sum_{\substack{i\in ids_R(J)|\\ I[i].A\neq J[i].A}} \sum_{k_{R.A}} w_{k_{R.A}} \cdot f_{k_{R.A}}(J[i])$$

$$\times \exp \sum_{R\in\mathbf{S}} \sum_{A\in R} \sum_{\substack{i\in ids_R(J)|\\ I[i].A=J[i].A}} \log\left(1 - p_{R.A} + p_{R.A} \cdot \mathsf{VG}_{R.A}(J[i])\right)$$

Given the above, it is easy to see that combining $\mathcal{I}$ with an update realizer corresponds to a Markov parametric PUD as well.

B. PROOFS

In this section, we provide proofs for that are missing from the body of the paper.

B.1 Proof of Theorem 5.3

THEOREM 5.3. *Let $(\mathcal{D}, \mathsf{VG}, p, J^\star)$ be an MLD/update PUD where $\mathcal{D}$ is the MLD $(\mathcal{P}, \Phi, w)$ and $\mathcal{P}$ is the TIID $(\varrho, p', \mathsf{TG}')$. Suppose that TG' is symmetric and can freely update $J^\star$. There exists a number M such that if $w_\varphi > M$ for all $\varphi \in \Phi$ then the following are equivalent for all updates I of $J^\star$.*

1. *I is an MLI.*
2. *I is a minimum update repair of $J^\star$ w.r.t. Φ under the value weight $w^{R.A}_{J^\star}(i) = -\log(r_{R.A}(i))$.*

PROOF. With our notation, we can phrase $\mathcal{R}_\mathcal{I}(I, J^\star)$ as follows.

$$\mathcal{R}_\mathcal{I}(I, J^\star) = \mathcal{D}(I) \times \prod_{R.A} \Big(\prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A\neq J^\star[i].A}} p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i]) \times \prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A=J^\star[i].A}} eq_{R.A}(i) \Big) \tag{5}$$

As in the case of Theorem 5.1, if w_φ is large enough for all $\varphi \in \Phi$, then we can assume that every I that satisfies Φ has a higher probability than every I that violates Φ. Hence, we can assume that the MLI I satisfies Φ, in which case we get that $\mathcal{D}(I) \sim \mathcal{P}(I)$. Since TG' is symmetric, we view it as an independent application of a value generator VG':

$$\mathcal{P}(I) = \prod_{R.A} \prod_{i\in ids_R(J^\star)} \mathsf{VG}'_{R.A}(I[i])$$

Hence, by combining with (5) we get the following.

$$\mathcal{R}_\mathcal{I}(I, J^\star) \sim \prod_{R.A} \Big(\Big(\prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A\neq\\ J^\star[i].A}} \Big(\mathsf{VG}'_{R.A}(I[i]) \cdot p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i]) \Big) \times \Big(\prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A=J^\star[i].A}} \mathsf{VG}'_{R.A}(J^\star[i]) \cdot eq_{R.A}(i) \Big) \Big) \Big)$$

$$= \left(\prod_{R.A} \prod_{i\in ids(J^\star)} \mathsf{VG}'_{R.A}(J^\star[i]) \cdot eq_{R.A}(i) \right) \times \left(\prod_{R.A} \prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A\neq J^\star[i].A}} \frac{\mathsf{VG}'_{R.A}(I[i]) \cdot p_{R.A} \cdot \mathsf{VG}_{R.A}(J^\star[i])}{\mathsf{VG}'_{R.A}(J^\star[i]) \cdot eq_{R.A}(i)} \right)$$

In the last product, the first factor is the same for all I, and hence can be ignored. Moreover, symmetry of TG' implies that $\mathsf{VG}'_{R.A}(I[i]) = \mathsf{VG}'_{R.A}(J^\star[i])$. Hence, we get the following.

$$\mathcal{R}_\mathcal{I}(I, J^\star) \sim \prod_{R.A} \prod_{\substack{i\in ids_R(J^\star)|\\ I[i].A\neq J^\star[i].A}} r_{R.A}(i)$$

Hence, an MLI corresponds to an update repair with a minimum sum of $-\log(r_{R.A}(i))$ over all updated values. □

B.2 Proof of Proposition 5.5

PROPOSITION 5.5. *Given a collection of training examples $(I_1, J_1), \ldots, (I_n, J_n)$ and a Markov parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta)$ the negative log-likelihood is a convex function of the parameter vector (Ξ, Θ).*

PROOF. The negative log-likelihood of a Markov parametric PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta)$ can be written as:

$$-\log \mathcal{R}_\mathcal{I}(I, J) = -\log \mathcal{I}_\Xi(I) - \log \mathcal{R}_\Theta(J|I).$$

To show that this is convex, we will show that each of the components of this sum is convex. First, we have that the negative log-likelihood for the intention model can be written as:

$$-\log \mathcal{I}_\Xi(I) = -\log\left(\frac{1}{Z}\exp\left(\Xi^T \cdot u_\mathcal{I}(I)\right)\right) = \log(Z) - \Xi^T \cdot u_\mathcal{I}(I) = \log\left(\sum_{I' \in \Omega}\exp\left(\Xi^T \cdot u_\mathcal{I}(I')\right)\right) - \Xi^T \cdot u_\mathcal{I}(I).$$

As a function of Ξ, this is convex because it is the sum of a LogSumExp function with an affine function, which is well-known to be convex [BV04]. Similarly, for the realizer,

$$\begin{aligned}-\log \mathcal{R}_\Theta(J|I) &= -\log\left(\frac{1}{Z(I)}\exp\left(\Theta^T \cdot u_\mathcal{R}(J|I)\right)\right) = \log(Z(I)) - \Theta^T \cdot u_\mathcal{R}(J|I)\\ &= \log\left(\sum_{J' \in \Omega}\exp\left(\Theta^T \cdot u_\mathcal{R}(J'|I)\right)\right) - \Theta^T \cdot u_\mathcal{R}(J|I).\end{aligned}$$

This is convex for the same reason as the intention model. So the entire minus-log-likelihood expression is convex. □

B.3 Proof of Theorem 5.6

THEOREM 5.6. *Let* $\mathbf{S}$ *be a schema and* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta)$ *be a parametric tuple-independent MLD/update PUD. Given a collection* $T = (I_1, J_1), \ldots, (I_n, J_n)$ *of labeled examples, the negative log-likelihood can be written as*

$$nll(\Xi, \Theta : D) = \sum_{k=1}^{n}\sum_{i \in ids(I_k)} l(I_k[i], J_k[i]; \Xi, \Theta),$$

where l *is a convex function of* Ξ *and* Θ.

PROOF. In Appendix A we showed that parametric MLD PUDs can be written as Markov parametric PUDs. We have that the negative log-likelihood of observing database (I, J) is:

$$-\log \mathcal{R}_\mathcal{I}(I, J) = -\log \mathcal{I}_\Xi(I) - \log \mathcal{R}_\Theta(J|I).$$

We will first show that (1) the negative log-likelihood decomposes as a sum of functions over individual tuples in I and (2) each such function is convex. First, we focus on the intention model. First, from Appendix A we have that:

$$\begin{aligned}\mathcal{I}(I) = \frac{1}{Z}\exp\sum_{R \in \mathbf{S}}&\left(\log(p_R)\cdot|ids_R(I)| + \sum_{i \in ids_R(I)}\sum_{k_R} w_{k_R}\cdot f_{k_R}(I[i]) + \log(1-p_R)\cdot|\varrho_R \setminus ids_R(I)|\right)\\ &\times \exp\sum_{\varphi \in \Phi}(-w(\varphi)\cdot|V(I,\varphi)|)\end{aligned}$$

where $Z = \sum_{I' \in \Omega}\prod_{\varphi \in \Phi} e^{-w(\varphi)\cdot|V(I',\varphi)|}$. We can rewrite the above expression as:

$$\begin{aligned}\mathcal{I}(I) &= \frac{1}{Z}\times\exp\sum_{R \in \mathbf{S}}\left(\sum_{i \in ids_R(I)}\left(\log(\frac{p_R}{1-p_R}) + \sum_{k_R} w_{k_R}\cdot f_{k_R}(I[i])\right) + \sum_{i \in \varrho_R}\log(1-p_R)\right)\\ &\quad\times\exp\sum_{\varphi \in \Phi}(-w(\varphi)\cdot|V(I,\varphi)|)\\ &= \frac{1}{Z}\times\exp\sum_{R \in \mathbf{S}}\sum_{i \in \varrho_R}\log(1-p_R)\times\exp\sum_{R \in \mathbf{S}}\left(\sum_{i \in ids_R(I)}\left(\log(\frac{p_R}{1-p_R}) + \sum_{k_R} w_{k_R}\cdot f_{k_R}(I[i])\right)\right)\\ &\quad\times\exp\sum_{\varphi \in \Phi}(-w(\varphi)\cdot|V(I,\varphi)|)\\ &= \frac{1}{Z}\times\text{Const}\times\exp\sum_{i \in ids(I)} f_i(I[i])\times\exp\sum_{\varphi \in \Phi}(-w(\varphi)\cdot|V(I,\varphi)|)\end{aligned}$$

where $f_i(\cdot)$ is an affine function of $I[i]$. For the negative log-likelihood we have that:

$$
\begin{aligned}
-\log \mathcal{I}(I) &= -\log\left(\frac{1}{Z} \times \text{Const} \times \exp \sum_{i \in ids(I)} l_i(I[i]) \times \exp \sum_{\varphi \in \Phi} (-w(\varphi) \cdot |V(I,\varphi)|)\right) \\
&= \log(Z) - \log(\text{Const}) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} (-w(\varphi) \cdot |V(I,\varphi)|) \\
&= \log\left(\sum_{I' \in \Omega} \prod_{\varphi \in \Phi} e^{-w(\varphi) \cdot |V(I',\varphi)|}\right) - \log(\text{Const}) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} (-w(\varphi) \cdot |V(I,\varphi)|) \\
&= \log\left(\sum_{I' \in \Omega} \exp \sum_{\varphi \in \Phi} -w(\varphi) \cdot |V(I',\varphi)|\right) - \log(\text{Const}) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} (-w(\varphi) \cdot |V(I,\varphi)|)
\end{aligned}
$$

Since we focus on tuple-independent PUDs every formula φ has only one free variable. Therefore, we can let $grd(i,\varphi)$ denote that formula grounded with tuple id i, and rewrite our expression as:

$$
-\log \mathcal{I}(I) = \log\left(\sum_{I' \in \Omega} \exp \sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I') \\ I' \models grd(i,\varphi)}} -w(\varphi)\right) - \log(\text{Const}) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I) \\ I \models grd(i,\varphi)}} (-w(\varphi))\,.
$$

Since $I \models grd(i,\varphi)$ if and only if $I|_{\{i\}} \models grd(i,\varphi)$, it follows that

$$
-\log \mathcal{I}(I) = \log\left(\sum_{I' \in \Omega} \exp \sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I') \\ I'|_{\{i\}} \models grd(i,\varphi)}} -w(\varphi)\right) - \log(\text{Const}) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I) \\ I|_{\{i\}} \models grd(i,\varphi)}} (-w(\varphi))\,.
$$

Now, if we look at the argument of the logarithm above more closely, we notice that the body of the sum can be factored in terms of expressions that only depend on $I'|_{\{i\}}$. It follows by sum separation that:

$$
\sum_{I' \in \Omega} \exp\left(\sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I') \\ I'|_{\{i\}} \models grd(i,\varphi)}} -w(\varphi)\right) = \prod_{i \in \varrho} \sum_{I[i] \in \Omega} \exp\left(\sum_{\varphi \in \Phi} \sum_{I'|_{\{i\}} \models grd(i,\varphi)} -w(\varphi)\right)
$$

Therefore,

$$
-\log \mathcal{I}(I) = \sum_{i \in \varrho} \log\left(\sum_{I[i] \in \Omega} \exp\left(\sum_{\varphi \in \Phi} \sum_{I'|_{\{i\}} \models grd(i,\varphi)} -w(\varphi)\right)\right) - \log(C) - \sum_{i \in ids(I)} f_i(I[i]) - \sum_{\varphi \in \Phi} \sum_{\substack{i \in ids(I) \\ I|_{\{i\}} \models grd(i,\varphi)}} (-w(\varphi))
$$

$$
= \sum_{i \in \varrho}\left(\log\left(\sum_{I[i] \in \Omega} \exp\left(\sum_{\varphi \in \Phi} \sum_{I'|_{\{i\}} \models grd(i,\varphi)} -w(\varphi)\right)\right) + f_i(I[i]) \cdot \mathbf{1}_{i \in I}[i] + \sum_{\varphi \in \Phi} \sum_{I|_{\{i\}} \models grd(i,\varphi)} (-w(\varphi))\right) - \log(C)
$$

where $\log(C) = \log(\text{Const})$. Similarly the negative log-likelihood of the realizer can be written as:

$$
\begin{aligned}
-\log(\mathcal{R}(J|I)) = &\sum_{i \in ids(J)} \left(\sum_{R \in \mathbf{S}} \sum_{A \in R} \left(\left(\log(p_{R.A}) + \sum_{k_{R.A}} w_{k_{R.A}} \cdot f_{k_{R.A}}(J[i])\right) \mathbf{1}_{I[i].A \neq J[i].A}[i]\right)\right) \\
&\times \sum_{i \in ids(J)} \left(\sum_{R \in \mathbf{S}} \sum_{A \in R} \log\left(1 - p_{R.A} + p_{R.A} \cdot \mathsf{VG}_{R.A}(J[i])\right) \cdot \mathbf{1}_{I[i].A = J[i].A}[i]\right)
\end{aligned}
$$

As shown the negative log-likelihood for $\mathcal{I}(I)$ decomposes into a sum over a constant plus terms each of which depends only on $(I|_{\{i\}}, J|_{\{i\}})$. Each component of the sum corresponds to a convex function as it is composed by the sum of a log-exp function with affine functions. □

B.4 Proof of Theorem 5.8

Before we present the proof for this theorem we discuss the tools used to proof asymptotic normality. Here, we rely on the notion of *Fisher information* of the available training data [Kul97]. The Fisher information determines the amount of information that observed database instances carry about the unknown parameters Ξ and Θ. Intuitively, Fisher information can be interpreted as a measure of how quickly the distribution density will change when we slightly change a parameter in θ near the optimal θ^*.

Next, we define the Fisher information of a Markov parametric PUD (see Section 5.3.1). Recall that $u_{\mathcal{I}}$ and $u_{\mathcal{R}}$ correspond to the sufficient statistics of the intention and realizer of a PUD and are defined in the beginning of Section 5.3.1.

Definition B.1. *The* intention Fisher information *of a Markov parametric PUD is:*

$$\mathcal{I}(\Xi) = \mathbf{Cov}_{I\sim\mathcal{I}_\Xi}\left[u_{\mathcal{I}}(I)\right].$$

Similarly, the realizer Fisher information *of a parametric PUD is:*

$$\mathcal{I}(\Theta) = \mathbf{E}_{I\sim\mathcal{I}_\Xi}\left[\mathbf{Cov}_{J\sim\mathcal{R}_\Theta(\cdot|I)}\left[u_{\mathcal{R}}(J|I)\right]\right].$$

For general parameter learning, the Fisher information matrices can be *singular*, i.e., our observations carry no information about the parameters in some direction. Two conditions that lead to singular Fisher information matrices being singular are: (1) the parameters of our PUD model are *redundant* (e.g., we can have the same formula listed twice with different weights) or (2) there is a parameter in our PUD model that has no effect on the distribution (e.g., we have a parameter associated with a formula that always evaluates to true). Notice that both cases described above correspond to misspecified parametric PUDs.

We now show that for any Markov parametric PUD the Fisher information matrices are positive definite, thus, not singular. Given that the Fisher information matrices are not singular, and thus invertible, we show the *asymptotic normality* of the MLE estimates Ξ and Θ for tuple independent MLD/update PUDs.

Lemma B.2. *For any PUD, if the sufficient statistics function $u_{\mathcal{I}}$ considered as a matrix $u_{\mathcal{I}} : \mathcal{R}^{|\Omega|\times d}$ is always full-rank, and similarly for $u_{\mathcal{R}}(\cdot|I)$, and all parameters (Ξ, Θ) are finite, then*

$$\mathcal{I}(\Xi) \succ 0 \text{ and } \mathcal{I}(\Theta) \succ 0.$$

That is that $\mathcal{I})(\Xi)$ and $\mathcal{I}(\Theta)$ are positive definite and thus invertible.

Proof of Lemma B.2. For the intention model, the Fisher information is the covariance of the sufficient statistics function $u_I(I)$. The only way this matrix could be singular is if there is some unit vector ϕ such that

$$\mathbf{Var}_{I\sim\mathcal{I}_\Xi}[\phi^T u_{\mathcal{I}}(I)] = 0.$$

This, in turn, will only happen if $\phi^T u_I(I)$ is constant across all I on which $\mathcal{I}_\Xi$ is supported. Since Ξ is finite, $\mathcal{I}_\Xi$ is supported everywhere on Ω. This means that if we define $\bar{u}_I$ as

$$\bar{u}(I) = u_{\mathcal{I}}(I) - \phi\phi^T u_{\mathcal{I}}(I),$$

then $\bar{u}(I)$ will also be a sufficient statistics function for the same PUD. But, $\bar{u}(I)$ is rank deficient, because $\phi^T\bar{u} = 0$. But this cannot happen, since we supposed that any sufficient statistics matrix would be full rank. Therefore, the Fisher information is positive definite, which is what we wanted to prove. □

Given that the Fisher information matrices are invertible we can now proceed to show asymptotic normality. Before we proceed with our theorem, we focus on notation that is used in the next theorem. Since we focus on tuple independent MLD/update PUDs, a collection $T = (I_1, J_1),\ldots,(I_n, J_n)$ of labeled examples can be viewed as a single pair (I, J) whose tuples correspond to the union of tuples of all examples in T. Furthermore, each tuple in (I, J) can be viewed as an independent single-tuple database example. In the theorem below, we use quantities $\mathcal{I}_1(\Xi^*)$ and $\mathcal{I}_1(\Theta^*)$ to denote the Fisher information of a parametric PUD over a single-tuple database. We have for asymptotic normality:

Theorem 5.8. *Let $\mathbf{S}$ be a schema and $\mathcal{U} = (\mathcal{I}_{\Xi^\star}, \mathcal{R}_{\Theta^\star})$ be a parametric tuple-independent MLD/update PUD. For training collections T sampled from $\mathcal{U}$, estimators (Ξ, Θ) are asymptotically normal:*

$$\sqrt{|T|}\left(\Xi - \Xi^\star\right) \to \mathcal{N}\left(0, \Sigma^2_{\Xi^\star}\right) \text{ as } |T| \to \infty$$
$$\sqrt{|T|}\left(\Theta - \Theta^\star\right) \to \mathcal{N}\left(0, \Sigma^2_{\Theta^\star}\right) \text{ as } |T| \to \infty$$

where $\Sigma^2_{\Xi^\star} = \mathcal{I}_1(\Xi^\star)^{-1}$ and $\Sigma^2_{\Theta^\star} = \mathcal{I}_1(\Theta^\star)^{-1}$ correspond to the asymptotic variance of estimates Ξ and Θ and $\mathcal{I}_1(\cdot)$ denotes the Fisher information of a parametric PUD over a single-tuple database.

The proof of this Theorem follows the form of standard proofs for asymptotic normality [EH78].

Proof of Theorem 5.8. We will prove this by the standard proof technique that is used to prove asymptotic normality. Recall that the gradient of the negative log-likelihood of an exponential family model with sufficient statistics u and parameters θ is:

$$\nabla f(\theta) = -\nabla_\theta \log \left(\frac{\exp(\theta^T u(x))}{\sum_{y \in \Omega} \exp(\theta^T u(y))} \right) = -u(x) + \frac{\sum_{y \in \Omega} \exp(\theta^T u(y)) u(y)}{\sum_{y \in \Omega} \exp(\theta^T u(y))} = -u(x) + \mathbf{E}_{y \sim \pi_\theta} [u(y)]$$

and the Hessian (the matrix that corresponds to the second-order partial derivatives) is

$$\begin{aligned} \nabla^2 f(\theta) &= \frac{\sum_{y \in \Omega} \exp(\theta^T u(y)) u(y) u(y)^T}{\sum_{y \in \Omega} \exp(\theta^T u(y))} - \frac{\sum_{y \in \Omega} \exp(\theta^T u(y)) u(y)}{\sum_{y \in \Omega} \exp(\theta^T u(y))} \cdot \frac{\sum_{y \in \Omega} \exp(\theta^T u(y)) u(y)^T}{\sum_{y \in \Omega} \exp(\theta^T u(y))} \\ &= \mathbf{E}_{x \sim \pi_\theta} \left[u(x) u(x)^T \right] - \mathbf{E}_{x \sim \pi_\theta} [u(x)] \, \mathbf{E}_{x \sim \pi_\theta} [u(x)]^T = \mathbf{Cov}_{x \sim \pi_\theta} [u(x)] \, . \end{aligned}$$

In the limit, we have that the gradient at the true parameter values is, for training examples $x_1, \ldots, x_N$,

$$\nabla f(\theta) = \mathbf{E}_{y \sim \pi_\theta} [u(y)] - \frac{1}{N} \sum_{i=1}^{N} u(x_i).$$

By a Taylor expansion, we expand the negative log-likelihood about the true parameters $\theta^\star$ and obtain that:

$$\nabla f(\theta) = H(\theta - \theta^*).$$

From the above and Lemma B.2, which states that the Fisher information is positive definite and thus invertible, it follows that

$$\theta - \theta^* = \mathcal{I}(\theta)^{-1} \left(\mathbf{E}_{y \sim \pi_\theta} [u(y)] - \frac{1}{N} \sum_{i=1}^{N} u(x_i). \right) = (\mathbf{Cov}_{x \sim \pi_\theta} [u(x)])^{-1} \left(\mathbf{E}_{y \sim \pi_\theta} [u(y)] - \frac{1}{N} \sum_{i=1}^{N} u(x_i). \right)$$

where $\mathcal{I}(\theta) = \mathbf{Cov}_{x \sim \pi_\theta} [u(x)]$ is the Fisher information of the model for x. This has expected value 0, and covariance

$$\mathbf{Cov} [\theta - \theta^*] = \frac{1}{N} \mathbf{Cov}_{x \sim \pi_\theta} [u(x)]^{-1} \, .$$

If we plug in our Markov parametric intention model to the above we have that:

$$\mathbf{Cov}[\Xi - \Xi^\star] = \frac{1}{|T|} \mathbf{Cov}_{I \sim \mathcal{I}_\Xi} [u_{\mathcal{I}}(I)]^{-1}$$

Therefore:

$$\mathbf{Cov}[\Xi - \Xi^\star] = \frac{1}{|T|} \mathcal{I}_1(\Xi^\star)^{-1} \text{ as } |T| \to \infty.$$

For the conditional model, we have:

$$\pi(X|I) = \frac{\exp(\theta^T u(X|I))}{\sum_Y \exp(\theta^T u(Y|I))}.$$

The gradient of the negative log-likelihood is

$$\nabla_\theta - \log \pi(X|I) = -u(X|I) + \frac{\sum_Y \exp(\theta^T u(Y|I)) u(Y|I)}{\sum_Y \exp(\theta^T u(Y|I))} = -u(X|I) + \mathbf{E}_{Y \sim \pi(\cdot|I)} [u(Y|I)] \, .$$

The Hessian is

$$\begin{aligned} \nabla^2_\theta - \log \pi(X|I) &= \frac{\sum_Y \exp(\theta^T u(Y|I)) u(Y|I) u(Y|I)^T}{\sum_Y \exp(\theta^T u(Y|I))} - \left(\frac{\sum_Y \exp(\theta^T u(Y|I)) u(Y|I)}{\sum_Y \exp(\theta^T u(Y|I))} \right) \left(\frac{\sum_Y \exp(\theta^T u(Y|I)) u(Y|I)}{\sum_Y \exp(\theta^T u(Y|I))} \right)^T \\ &= \mathbf{Cov}_{Y \sim \pi(\cdot|I)} [u(Y|I)] \, . \end{aligned}$$

In expectation over I, this will be

$$\nabla^2 f(\theta) = \mathbf{E}_I \left[\mathbf{Cov}_{Y \sim \pi(\cdot|I)} [u(Y|I)] \right] .$$

This will leave us with a typical gradient using samples of

$$\nabla f(\theta) = \frac{1}{N} \sum_{k=1}^{N} \left(\mathbf{E}_{Y \sim \pi(\cdot|I_k)} [u(Y|I_k)] - u(X_k|I_k) \right) .$$

This will have expected value 0, and covariance

$$\mathbf{Cov}\left[\nabla f(\theta)\right] = \mathbf{Cov}\left[\frac{1}{N}\sum_{k=1}^{N}\left(\mathbf{E}_{Y\sim\pi(\cdot|I_k)}\left[u(Y|I_k)\right] - u(X_k|I_k)\right)\right] = \frac{1}{N}\mathbf{Cov}\left[\mathbf{E}_{Y\sim\pi(\cdot|I_1)}\left[u(Y|I_1)\right] - u(X_1|I_1)\right]$$
$$= \frac{1}{N}\mathbf{E}_{I_1}\left[\mathbf{Cov}_{X_1}\left[\mathbf{E}_{Y\sim\pi(\cdot|I_1)}\left[u(Y|I_1)\right] - u(X_1|I_1)\right]\right] = \frac{1}{N}\mathbf{E}_{I_1}\left[\mathbf{Cov}_{X_1}\left[u(X_1|I_1)\right]\right].$$

So in this setting,

$$\mathbf{Cov}\left[\theta - \theta^*\right] = \frac{1}{N}\mathbf{E}_z\left[\mathbf{Cov}_{x\sim\pi_\theta}\left[u(x|z)\right]^{-1}\right].$$

If we plug in our Markov parametric realizer model to the above we have that:

$$\mathbf{Cov}[\Theta - \Theta^\star] = \frac{1}{|T|}\mathbf{E}_{I\sim\mathcal{I}_\Xi}\mathbf{Cov}_{J\sim\mathcal{R}_\Theta(\cdot|I)}\left[u_\mathcal{R}(J|I)\right]^{-1}$$

Therefore:

$$\mathbf{Cov}[\Theta - \Theta^\star] = \frac{1}{|T|}\mathcal{I}_1(\Theta^\star)^{-1} \text{ as } |T|\to\infty.$$

This concludes the proof. □

B.5 Proof of Theorem 5.9

THEOREM 5.9. *Given a schema* $\mathbf{S}$ *and a Markov parametric MLD/update PUD* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$ *where* Ξ *belongs in a compact convex set, there exists some fixed* $p > 0$ *such that whenever the low-noise condition holds with respect to* p*, the negative log-likelihood* $nll(\Xi, \Theta : J^\star)$ *is a convex function of* Ξ*.*

PROOF. Considering a PUD $(\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$ for which we want to solve PUD learning having access only to $J^\star$ and not training examples. Specifically, we focus on the MLE estimate of Ξ. We have that the negative log-likelihood with respect to Ξ is:

$$nll(\Xi) = -\log\left(\sum_{I\in\Omega}\mathcal{I}_\Xi(I)\cdot\mathcal{R}_\Theta(J^\star|I)\right) = -\log\left(\sum_{I\in\Omega}\frac{\exp(\Xi^T u_\mathcal{I}(I))}{\sum_{J\in\Omega}\exp(\Theta^T u_\mathcal{I}(J))}\cdot\mathcal{R}_\Theta(J^\star|I)\right).$$

The gradient of this is

$$\nabla f(\Xi) = \frac{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))\cdot u_\mathcal{I}(J)}{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))} - \frac{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)u_\mathcal{I}(I)}{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)}$$

and the Hessian is

$$\begin{aligned}\nabla^2 f(\Xi) = &\frac{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))\cdot u_\mathcal{I}(J)\cdot u_\mathcal{I}(J)}{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))} - \left(\frac{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))\cdot u_\mathcal{I}(J)}{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))}\right)\left(\frac{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))\cdot u_\mathcal{I}(J)}{\sum_{J\in\Omega}\exp(\Xi^T u_\mathcal{I}(J))}\right)^T \\ &- \frac{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)\cdot u_\mathcal{I}(I)\cdot u_\mathcal{I}(I)}{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)} \\ &+ \left(\frac{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)\cdot u_\mathcal{I}(I)c\dot{u}_\mathcal{I}(I)}{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)}\right)\left(\frac{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)\cdot u_\mathcal{I}(I)\cdot u_\mathcal{I}(I)}{\sum_{I\in\Omega}\exp(\Xi^T u_\mathcal{I}(I))\cdot\mathcal{R}_\Theta(J^\star|I)}\right)^T \\ &= \mathbf{Cov}[u_\mathcal{I}(I)] - \mathbf{Cov}[u_\mathcal{I}(I)|J^\star].\end{aligned}$$

By the argument in Lemma B.2, we know that $\mathbf{Cov}[u_\mathcal{I}(I)] \succ 0$ on all Ξ. It follows by continuity that there exists a δ such that on Ξ, $\mathbf{Cov}[u_\mathcal{I}(I)] \succ \delta I$. On the other hand, if we have a $\mathcal{R}_\Theta(J^\star|I)$ such that for each tuple identifier $i \in I$ $I[i] = J^\star[i]$ with probability at least $1-p$, then

$$\mathbf{Cov}[u_\mathcal{I}(I)|J^\star] \preceq p\max_{J\in\Omega}\|u_I(J)\|^2.$$

It follows that we can choose a p small enough that the Hessian is always positive definite on Ξ, which means f is convex. This completes the proof. Notice that for the Hessian to be positive definite it must be that $p\cdot\max_{J\in\Omega}\|u_\mathcal{I}(J)\|^2 \prec \delta I$ which in turn means that the value of p depends on the maximum value of the sufficient statistics $u_\mathcal{I}$ when computed over J. □

B.6 Proof of Theorem 5.10

THEOREM 5.10. *Given a schema* $\mathbf{S}$ *and a parametric tuple-independent PUD* $\mathcal{U} = (\mathcal{I}_\Xi, \mathcal{R}_\Theta, J^\star)$*, parameter* p *for Theorem 5.9 to hold is independent of the number of tuple identifiers in repository* ϱ *and the negative log-likelihood can be written as*

$$nll(\Xi, \Theta : J^\star) = \sum_{i \in ids(J^\star)} l_i(J^\star[i]; \Xi, \Theta),$$

where each l_i *is a convex function of* Ξ *(keeping* Θ *fixed).*

PROOF. According to the proof of Theorem 5.9 we have that for the Theorem to hold it must be that

$$p \cdot \max_{J \in \Omega} \|u_{\mathcal{I}}(J)\|^2 \prec \delta I$$

Also if the PUD is tuple-independent we have from the proof of Theorem 5.6 that:

$$-log(\mathcal{I}_\Xi(I)) = \sum_{i \in \varrho} \Bigg(\log \left(\sum_{I[i] \in \Omega} \exp \left(\sum_{\varphi \in \Phi} \sum_{I'|_{\{i\}} \models grd(i,\varphi)} -w(\varphi) \right) \right) + f_i(I[i]) \cdot \mathbf{1}_{i \in I}[i] + \sum_{\varphi \in \Phi} \sum_{I|_{\{i\}} \models grd(i,\varphi)} (-w(\varphi)) \Bigg) - \log(C)$$

Given the above it is easy to see that the maximum value for $\|u_I(J)\|^2$ for any $J \in \Omega$ scales independently of the number of tuples in repository ϱ and depends only on the statistics that describe the tuple generator TG of intention $\mathcal{I}_\Xi$. Moreover, given a fixed realizer $\mathcal{R}_\Theta$ the negative log-likelihod decomposes to a sum over convex functions over individual tuple identifiers. □