Discovering Functional Dependencies in Pay-As-You-Go Data Integration Systems

Daisy Zhe Wang
Michael Franklin
Luna Dong
Anish Das Sarma
Alon Halevy

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-119
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-119.html

August 15, 2009

Discovering Functional Dependencies in Pay-As-You-Go Data Integration Systems

Daisy Zhe Wang* , Luna Dong† , Anish Das Sarma‡ , Alon Halevy§ , and Michael J. Franklin*
*Univ. of California, Berkeley EECS and †AT&T Research and ‡Stanford University and §Google Inc.

ABSTRACT

Functional dependency is one of the most extensively researched subjects in database theory, originally for improving quality of schemas, and recently for improving quality of data. In a *pay-as-you-go* data integration system, where the goal is to provide best-effort service even without thorough understanding of the underlying domain and the various data sources, functional dependency can play an even more important role, applied in normalizing an automatically generated mediated schema, pinpointing sources of low quality, resolving conflicts in data from different sources, improving efficiency of query answering, and so on. Despite its importance, discovering functional dependencies in such a context is challenging: we cannot assume upfront domain knowledge for specifying dependencies, and the data can be dirty, incomplete, or even misinterpreted, so make automatic discovery of dependencies hard.

This paper studies how one can automatically discover functional dependencies in a pay-as-you-go data integration system. We introduce the notion of *probabilistic functional dependencies (pFDs)* and design Bayes models that compute probabilities of dependencies according to data from various sources. As an application, we study how to normalize a mediated schema based on the pFDs we generate. Experiments on real-world data sets with tens or hundreds of data sources show that our techniques obtain high precision and recall in dependency discovery and generate high-quality results in mediated-schema normalization.

1. INTRODUCTION

Functional dependency is one of the most thoroughly researched subjects in database theory [5]. Consider two sets of attributes $\mathbf{X}$ and $\mathbf{Y}$ in a relational schema. A functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ states that all tuples with the same value of $\mathbf{X}$ must have the same value of $\mathbf{Y}$. It is originally studied for improving quality of schema: according to functional dependencies, we can decompose a relation into several relations in certain normal forms, such as 3NF and BCNF. Recently, the topic is revisited for improving quality of data: the knowledge of functional dependencies can help remove dirtiness and resolve inconsistency existing in real-world data. Whereas the problem has been extensively studied for a single data source, how one can discover functional dependencies and apply them in various data management tasks when there are multiple data sources, typically heterogeneous and dirty, remains an open problem.

This paper studies discovery and possible applications of functional dependencies in a data integration system. In particular, we focus on a *pay-as-you-go* data integration system [6], where we may not fully understand the underlying domain and the data sources, but aim at providing best-effort services from the outset and improve them over time as we gain more knowledge of the data and the queries. Functional dependencies are especially helpful in such a system and we next list a few applications.

First, bootstrapping a pay-as-you-go data integration system requires a mediated schema, which captures the salient aspects of the domain, and mappings from source schemas to the mediated schema, which describe the semantic relationship between data from different sources. Recently techniques have been proposed to automatically create the mediated schema and the mappings (such as [18]); however, the created mediated schema typically is not perfect and not well normalized (if normalized at all). Knowledge of functional dependencies can help us normalize the mediated schema into relations that correspond to meaningful real-world entities and relationships, and so help users better understand the underlying data. In addition, observation of violation of some dependencies can help us pinpoint suspicious mappings between schemas. Second, the problem of dirtiness of data can be even more prominent in a pay-as-you-go system, where the data sources are only loosely coupled and are often of various quality. When we query such a system and integrate answers from different data sources, it is important to resolve conflicts from different sources to return clean answers to the user; the knowledge of functional dependencies can help in this aspect. Third, functional dependencies can often help us obtain more precise estimation of selectivity for query optimization and so improve efficiency in query answering.

Typically functional dependencies should be specified according to domain knowledge (and it is well known that data by themselves may suggest incorrect functional dependencies); however, this is often impossible in a pay-as-you-go system, where we cannot assume any upfront knowledge of the underlying domain and sometimes even domain experts can have uncertainty on some part of the domain. In such a setting, we have to discover functional dependencies from

data provided by various sources. The problem of automatic discovery of functional dependencies from data has been addressed in previous work [9, 11]; however, these techniques aim at a single data source and typically assume relatively clean and complete data. In a pay-as-you-go system, with presence of inconsistent information, missing data, different representations of the same object, misinterpretation of semantics of data (*i.e.*, wrong schema mappings), the functional dependencies we discover are at best probabilistic.

The key contribution of this paper is the concept of *probabilistic functional dependencies (pFDs)*, where the probability captures our uncertainty on correctness of the dependency in the underlying domain. We study how we can compute such probabilities automatically from data. Specifically, we first develop Bayes models that compute the probability of a functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ conditioned on our observation of data from various sources. Our models examine not only the tuples that observe or violate the dependency, but also the variety of the violation tuples on the value of $\mathbf{Y}$. We then enhance our basic models in two ways. First, we examine whether the set of functional dependencies for which we compute a high probability observes the transitivity rule and adjust the probabilities accordingly. Second, when the source schemas contain multiple tables, we consider the normalization already performed at the source side as stronger evidence of whether a functional dependency holds.

As an application of the pFDs we discover, we study how to normalize a mediated schema. We assume existence of a mediated schema generated automatically by existing techniques and study how we can normalize it to improve its understandability. Our goal in normalization is slightly different from that of traditional normalization: instead of trying to obtain a schema in a certain normal form, we aim at normalizing the schema into relations that correspond to real-world entities and relationships. We show how we can achieve this goal based on the pFDs we generate.

Specifically, this paper makes the following contributions:

- We formally define pFDs, describe Bayes models that generate pFDs from source data in a pay-as-you-to data integration system, and study various properties of our models.
- As an application, we study normalization of a mediated schema, most likely generated automatically, in a pay-as-you-go system. We present an algorithm that selects the best set of functional dependencies from the pFDs we generate and normalize accordingly.
- We experimented on data sources extracted from the web in different domains. Experimental results show that our pFD-discovery algorithm obtains high precision and recall and our normalization algorithm generates high-quality results.

This paper is organized as follows. Section 2 formally defines pFDs and the problems we solve in this paper. Section 3 describes two basic models that compute probability of a functional dependency according to source data. Section 4 describes two enhancements that improve results of the basic models. Section 5 studies normalization of a mediated schema in a pay-as-you-go system. Section 6 reports experimental results. Finally, Section 7 discusses related work and Section 8 concludes.

2. OVERVIEW

This section describes the several challenges we face in discovery of functional dependencies in a pay-as-you-go data integration system, formally defines the notion of probabilistic functional dependency and the problems this paper solves.

The main goal of this paper is *to generate a set of functional dependencies in a pay-as-you-go data integration system.* We consider the relational model, where each schema contains a set of relations and each relation contains a set of attributes. Let $\mathbf{X}, \mathbf{Y}$ be two subsets of attributes in the schema. We say $\mathbf{X}$ *functionally determines* $\mathbf{Y}$, denoted by $\mathbf{X} \rightarrow \mathbf{Y}$, if all tuples with the same value of $\mathbf{X}$ must have the same value of $\mathbf{Y}$. To simplify notations, our discussion focuses on a special case where both $\mathbf{X}$ and $\mathbf{Y}$ contain only a single attribute; however, the techniques we propose shall apply to the general case.

We assume existence of a *mediated schema* with a set of *mediated attributes* representing important attributes from the sources, and also the existence of matchings from source attributes to the mediated attributes. The mediated schema in a pay-as-you-go system is often automatically generated, so it is neither perfect nor necessarily well normalized. Indeed, one important motivation of finding functional dependencies is to facilitate normalization of an automatically generated mediated schema. For convenience, we assume we have preprocessed by replacing each source attribute by the mediated attribute it matches to; however, this step is not critical to our techniques.

We assume the sources are independent of each other and tuples in each source are independent of each other; however, our experiments on real-world data show that our techniques work well even on data sets that may violate these assumptions. For now we restrict ourselves to data sources that are in the same domain and contain only a few relations. It turns out that even for this setting, the problem is already hard. We defer considering more complex data sources to future work and the techniques we develop in this paper would be fundamental even in a more complex setting.

Challenges Discovering FDs usually requires an understanding of the domain and the data. However, the goal of a pay-as-you-go system is to provide some service from the outset even without a thorough understanding of the underlying domain. Thus, we cannot assume any upfront domain knowledge and have to rely on the source data for recognition of FDs. We face at least three challenges in this process:

- Real-world data are often dirty: there can be inconsistent and noisy data. Such dirtiness can prevent us from discovering correct FDs. For example, wrong information on publication year may lead to the wrong belief that paper ID does not functionally determine publication year.
- Data in a source are often incomplete: a source can provide only a small portion of data in the underlying domain. Such incompleteness can lead us to believe false FDs. For example, if a data source contains only publications in one conference, we can draw the wrong conclusion that publication year functionally determines conference name.
- The above challenges are aggravated when the sources are heterogeneous, as we can have wrong mappings from source schemas to the mediated schema, or do

not realize different representations of the same value (such as "California" and "CA").

Probabilistic functional dependency The key idea in our solution is to introduce the notion of *probabilistic functional dependency (pFD)*, which captures our uncertainty about whether an FD holds in the domain.

DEFINITION 2.1. (PROBABILISTIC FUNCTIONAL DEPENDENCY (PFD)) *Let $\mathcal{S}$ be a set of relational data sources, X and A be two different attributes in the mediated schema for $\mathcal{S}$. A* probabilistic functional dependency *is in the form of $X \rightarrow^p A$, where p indicates the likelihood that $X \rightarrow A$ holds in the underlying domain of the data sources in $\mathcal{S}$.* □

In this paper, we study how to generate pFDs from the source data in a pay-as-you-go data integration system. Our goal is to compute a high probability for a true FD and a low probability for a false FD.

Normalization of mediated schemas As an application, we study normalization of mediated schemas. Given a set of source schemas, [18] proposed a technique that generates a mediated schema with a single relation. In this paper, we study how we can normalize such a schema into multiple relations, where each relation describes an entity or a relationship between entities in the domain of the source schemas. We call such normalization *semantic normalization*.

Existing schema normalization techniques typically aim at normalizing the schema into a certain normal form. For example, the most common normal form, BCNF, requires that no attribute in a relation is functionally determined by a set of attributes that do not form a key of the relation. BCNF normalization finds all FDs that violate this condition and splits the relation accordingly. Our goal of semantic normalization is a bit different: each relation in our results should represent one and only one entity or relationship. Note that one subtle issue here is granularity of entities; we would like to avoid *oversplitting*: an entity with only a few attributes (too fine-grained) should be combined with its associated entity when appropriate to form a more coarse-grained entity. For example, rather than having a seperate relation Zipcode(zip, state) (zip functionally determines state), we would merge it with the Address relation.

3. BASIC MODELS FOR PFD GENERATION

Given an FD $D : X \rightarrow A$, we consider how to compute its probability, denoted by $P(D)$. In this computation, we need to take into account the possible noise (caused by dirtiness) and bias (caused by incompleteness) of data.

In this section, we describe two models for computing $P(D)$, the *per-value* model and the *per-tuple* model. The *per-tuple* model measures noise and bias at the tuple granularity; that is, what is the probability that each tuple contains noise or is biased on the value of A. The *per-value* model measures noise and bias at the value granularity; that is, for a particular value x of X, what is the probability that tuples with $X = x$ contain noise or are biased on the value of A. Both models apply Bayesian analysis to compute the probability of an FD based on the source data.

3.1 Per-tuple model

We start from the per-tuple model, which considers violation of an FD at the tuple granularity. We first define what we mean by a tuple *observing* or *violating* an FD.

If an FD $X \rightarrow A$ holds, for each value of X, there should be a single value of A. In real data sets, there can be multiple values of A because of noise on A or on X. Given a particular value x of X, we decide the *true value* of A in a particular data source S, denoted by $a(x, S)$, as the value of A that occurs in most of the tuples with value $X = x$. Accordingly, a tuple with value $X = x$ is considered as observing the FD if $A = a(x, S)$, and violating the FD otherwise. We denote by $\mathbf{O}(S)$ all tuples in S that observe the FD, and by $\mathbf{V}(S)$ all tuples that violate it. We denote their sizes by $|\mathbf{O}(S)|$ and $|\mathbf{V}(S)|$ respectively.

Let ϵ_{tn} be the probability that if an FD is true, a tuple contains noise and violates it. In other words, the probability that a tuple violates a true FD is

$$P(f \in \mathbf{V}(S)|D) = \epsilon_{tn}. \quad (1)$$

On the other hand, let ϵ_{tb} be the probability that if an FD is false, a tuple with $X = x$ happens to have value $A = a(x, S)$. In other words, the probability that a tuple observes a false FD is

$$P(f \in \mathbf{O}(S)|\bar{D}) = \epsilon_{tb}. \quad (2)$$

As we assume the tuples in a data source are independent of each other, the probability that data from source S observe or violate D can be computed as follows:

$$P(S|D) = \epsilon_{tn}^{|\mathbf{V}(S)|} \cdot (1 - \epsilon_{tn})^{|\mathbf{O}(S)|}; \quad (3)$$

$$P(S|\bar{D}) = \epsilon_{tb}^{|\mathbf{O}(S)|} \cdot (1 - \epsilon_{tb})^{|\mathbf{V}(S)|}. \quad (4)$$

Finally, as we assume the data sources are independent of each other, we can compute the probability that data from various sources in $\mathcal{S}$ observe or violate D by taking the product of that probability for each individual data source.

$$P(\mathcal{S}|D) = \Pi_{S \in \mathcal{S}} P(S|D); \quad (5)$$

$$P(\mathcal{S}|\bar{D}) = \Pi_{S \in \mathcal{S}} P(S|\bar{D}). \quad (6)$$

By applying Bayesian analysis, we can compute the probability of an FD conditioned on our observation of the source data. Specifically, if we denote $V = \Sigma_{S \in \mathcal{S}} |\mathbf{V}(S)|$, and $O = \Sigma_{S \in \mathcal{S}} |\mathbf{O}(S)|$, and denote the a-priori belief that D holds by $P(D)$ (we describe how we set the parameters at the end of this section), we have

$$P(D|\mathcal{S}) = \frac{P(D)P(\mathcal{S}|D)}{P(D)P(\mathcal{S}|D) + (1 - P(D))P(\mathcal{S}|\bar{D})} = \left(1 + (P(D)^{-1} - 1)(\frac{\epsilon_{tb}}{1 - \epsilon_{tn}})^O (\frac{1 - \epsilon_{tb}}{\epsilon_{tn}})^V\right)^{-1} \quad (7)$$

EXAMPLE 3.1. *Consider dependency $D : B \rightarrow C$ and the following two data sources.*

S_1	A	B	C
t_1	a1	b1	c1
t_2	a2	b1	c1
t_3	a3	b1	c1
t_4	a4	b1	c1
t_5	a5	b1	c1

S_2	A	B	C
t_6	a6	b1	c1'
t_7	a7	b2	c2
t_8	a7	b2	c3
t_9	a7	b2	c4

Among the nine source tuples, t_8 and t_9 violate D and the other 7 tuples observe it. If we set $\epsilon_{tn} = .1, \epsilon_{tb} = .4$ and $P(D) = .5$, we compute the following probability for $B \rightarrow C$ using the per-tuple model, showing that the FD is likely to be true.

$$P(D|\{S_1, S_2\}) = \left(1 + 1 \cdot (\frac{.4}{1 - .1})^7 \cdot (\frac{1 - .4}{.1})^2\right)^{-1} = .89. \quad \square$$

The per-tuple model has several nice properties: the more source tuples observing the FD, the higher probability shall we compute; the more source tuples violating the FD, the lower probability shall we compute; also, our belief of an FD will be strengthened if we have more data sources. We formally state the properties in the following theorem and give proofs in the full version of this paper.

THOREM 3.2. *Equation (7) has the following properties.*

1. *Under condition $\epsilon_{tb} + \epsilon_{tn} < 1$, if we fix V, $P(D|\mathcal{S})$ increases as O increases, and if we fix O, $P(D|\mathcal{S})$ decreases as V increases.*
2. *Assume each source contains o tuples that observe D and v tuples that violate D. Then, under condition*

$$\epsilon_{tb}^{o}(1-\epsilon_{tb})^{v} < \epsilon_{tn}^{v}(1-\epsilon_{tn})^{o}, \tag{8}$$

$P(D|\mathcal{S})$ increases as the number of sources increases; otherwise, $P(D|\mathcal{S})$ decreases as the number of sources decreases. □

Note that if $P(D) = .5$ (*i.e.*, we do not have any a-priori preference on whether D holds), under condition (8), $Pr(D|S) > .5$ for any single data source S, and we consider D as more likely to be true. Then, the more data sources we have, the higher is the probability computed for D and so our belief is strengthened.

3.2 Per-value model

The *per-value* model defines violation of an FD $D : X \rightarrow A$ for each value of X; that is, for each value x of X, it examines if the set of tuples with $X = x$ in a source S observe or violate the FD. According to the definition of functional dependency, these tuples observe D if they all have the same value of A and violate D otherwise. We denote by $\mathbf{X}^+(S)$ the set of values of X in S whose tuples observe D and by $\mathbf{X}^-(S)$ the set of values of X whose tuples violate D.

We now consider how to compute the probability that tuples with a particular value $X = x$ observe or violate $X \rightarrow A$. We denote by ϵ_{vn} the probability that if D is true, tuples with the same value of X contain noise and introduce an extra value of A. If we denote by $\mathbf{A}(x, S)$ the set of values of A for tuples with value $X = x$, the probability that a value x violates a true FD is the probability that the tuples with $X = x$ introduce $|\mathbf{A}(x, S)-1|$ noise values. If we assume the noise values are independent of each other, the conditional probability should be

$$P(x \in \mathbf{X}^-(S)|D) = \epsilon_{vn}^{|\mathbf{A}(x,S)|-1}. \tag{9}$$

The probability that a value x does not violate a true FD is the probability that the tuples with $X = x$ do not introduce any number of noise values; thus,

$$P(x \in \mathbf{X}^+(S)|D) = 1 - \epsilon_{vn} - \cdots - \epsilon_{vn}^{\infty} = \frac{1-2\epsilon_{vn}}{1-\epsilon_{vn}}. \tag{10}$$

On the other hand, we denote by ϵ_{vb} the probability that if D is false, tuples with the same value of X are possibly incomplete and provide a single value of A. Thus,

$$P(x \in \mathbf{X}^+(S)|\bar{D}) = \epsilon_{vb}; \tag{11}$$

$$P(x \in \mathbf{X}^-(S)|\bar{D}) = 1 - \epsilon_{vb}. \tag{12}$$

Again, we assume independence of tuples in a data source and have

$$P(S|D) = \Pi_{X\in\mathbf{X}^-(S)}\epsilon_{vn}^{|\mathbf{A}(x,S)|-1} \cdot \Pi_{X\in\mathbf{X}^+(S)}\frac{1-2\epsilon_{vn}}{1-\epsilon_{vn}}; \tag{13}$$

$$P(S|\bar{D}) = (1-\epsilon_{vb})^{|\mathbf{X}^-(S)|} \cdot \epsilon_{vb}^{|\mathbf{X}^+(S)|}. \tag{14}$$

Finally, we assume independence of data sources and apply the Bayes Rule to compute the conditional probability $P(D|\mathcal{S})$. If we denote by $X^+ = \sum_{S\in\mathcal{S}}|\mathbf{X}^+(S)|$ and by $X^- = \sum_{S\in\mathcal{S}}|\mathbf{X}^-(S)|$, we have

$$P(D|\mathcal{S}) = (1 + (P(D)^{-1} - 1) \cdot (\frac{\epsilon_{vb}(1-\epsilon_{vn})}{1-2\epsilon_{vn}})^{X^+} \cdot \frac{(1-\epsilon_{vb})^{X^-}}{\Pi_{S\in\mathcal{S}}\Pi_{x\in X^-(S)}\epsilon_{vn}^{|\mathbf{A}(x,S)|-1}})^{-1} \tag{15}$$

EXAMPLE 3.3. *Continue with Example 3.1. In both sources, the value $b1$ satisfies $B \rightarrow C$; however, the value $b2$ in source S_2 violates $B \rightarrow C$. If we set $\epsilon_{vn} = .1, \epsilon_{vb} = .4$ and $P(D) = .5$, we obtain the following probability for $B \rightarrow C$ using the per-value model:*

$$P(D|\{S_1, S_2\}) = (1 + 1 \cdot (\frac{.4(1-.1)}{1-.1*2})^2 \cdot \frac{1-.4}{.1^2})^{-1} = .08.$$

We observe that the by-value model computes a significantly lower probability for $B \rightarrow C$, both because it does not differentiate the various tuples with $B = b1$, and because there are three different values of C for $B = b2$. □

The per-value model has similar properties as the per-tuple model. In addition, the number of values of A for a particular value of X affects the probability we compute: the higher variety of values of A for a value of X, the lower probability shall we compute. We next formally state the properties.

THOREM 3.4. *Equation (15) has the following properties.*

1. *If we fix X^+ and X^-, and fix $|\mathbf{A}(x, S)|$ for all x and $S \in \mathcal{S}$ except for x_0 and S_0, $P(D|\mathcal{S})$ decreases as $|\mathbf{A}(x_0, S_0)|$ increases.*
2. *Assume $|\mathbf{A}(x, S)|$ is the same for each x and $S \in \mathcal{S}$. Under condition $\epsilon_{vb} + 2\epsilon_{vn} < 1$, if we fix X^-, $P(D|\mathcal{S})$ increases as X^+ increases, and if we fix X^+, $P(D|\mathcal{S})$ decreases as X^- increases.*
3. *Assume each source contains o values of X that observe D and v values of X that violate D, and $|\mathbf{A}(x, S)|=a$ for each x and $S \in \mathcal{S}$. Then, under condition*

$$\epsilon_{vb}^{o}(1-\epsilon_{vb})^{v} < (\frac{1-2\epsilon_{vn}}{1-\epsilon_{vn}})^{o}(\epsilon_{vn})^{v(a-1)}, \tag{16}$$

$P(D|\mathcal{S})$ increases as the number of sources increases; otherwise, $P(D|\mathcal{S})$ decreases as the number of sources decreases. □

Per-tuple v.s. Per-value The per-tuple model and the per-value model are different in two aspects. First, the per-tuple model takes frequency of each value of X into account, but the per-value model does not; thus, a popular value of X will affect our belief of the probability of an FD much more in the per-tuple model than in the per-value model. Second, the per-value model takes into consideration the variety of values of A for each value of X, but the per-tuple model considers the various values just as randomly introduced noise

and so does not view a higher variety necessarily as more negative.

Each of the models has its strength. The per-tuple model is more suitable for data sets where the low quality of data is mainly caused by noise values. On the other hand, the per-value model is more appropriate for some skewed unnormalized data set, where a value of X gets popular just because the instance it represents is associated with a large number of instances of another entity type (such as the data set in Example 3.1).

We compared the two models in our experiments. On our data sets (details described in Section 6), the per-value model obtained better results than the per-tuple model.

3.3 Armstrong's Axioms

We now examine if the pFDs we discover have properties comparable to the *Armstrong's Axioms* for deterministic FDs. Based on this analysis, we discuss in the next section how we can improve the pFDs we generate.

On deterministic FDs, we have the following three rules, called Armstrong's Axioms:

- *Reflexivity:* If $\mathbf{X} \supseteq \mathbf{Y}$, then $\mathbf{X} \rightarrow \mathbf{Y}$;
- *Augmentation:* If $\mathbf{X} \rightarrow \mathbf{Y}$, then $\mathbf{XZ} \rightarrow \mathbf{YZ}$ for any $\mathbf{Z}$;
- *Transitivity:* If $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{Y} \rightarrow \mathbf{Z}$, then $\mathbf{X} \rightarrow \mathbf{Z}$.

Among them, we can easily find a comparable rule on pFDs for the reflexivity rule, formally stated as follows.

THOREM 3.5. *Let n be the total number of tuples in all data sources. If $\mathbf{X} \supseteq \mathbf{Y}$, then,*

- *in the per-tuple model, under condition $\epsilon_{tb} + \epsilon_{tn} < 1$, $P(D|\mathcal{S}) \rightarrow 1$ when $n \rightarrow \infty$.*
- *in the per-value model, under condition $\epsilon_{vb} + 2\epsilon_{vn} < 1$, $P(D|\mathcal{S}) \rightarrow 1$ when $n \rightarrow \infty$.* □

For the Augmentation rule, we can find a comparable rule for the per-tuple model, but not for the per-value model. We first state the results formally for the per-tuple model.

THOREM 3.6. *Let $\mathcal{S}$ be a set of data sources. Let $\mathbf{X} \rightarrow^{p} \mathbf{Y}$ and $\mathbf{XZ} \rightarrow^{p'} \mathbf{YZ}$ be two pFDs generated on $\mathcal{S}$ by the per-tuple model. Then, $p \leqslant p'$.* □

The following example shows that the same theorem does not hold for the per-value model.

EXAMPLE 3.7. *Consider the following relation.*

A	B	C
a	*b1*	*c1*
a	*b1*	*c2*
a	*b2*	*c1*
a	*b2*	*c2*

For FD $A \rightarrow B$, we have one violation value a. However, for FD $AC \rightarrow BC$, we have two violation values $(a, c1)$ and $(a, c2)$. Thus, by applying the per-value model, we compute lower probability for $AC \rightarrow BC$ than for $A \rightarrow B$. Note however that a similar analysis shows that we compute the same probability for the two FDs in per-tuple model. □

Finally, as the following example shows, using both models, we can compute a much lower probability for $\mathbf{X} \rightarrow \mathbf{Z}$ than that of $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{Y} \rightarrow \mathbf{Z}$.

EXAMPLE 3.8. *Consider the following relation.*

A	B	C
a1	*b1*	*c1*
a1	*b2*	*c2*
a2	*b3*	*c3*
a2	*b3*	*c4*

In the per-tuple model, there is one tuple violating $A \rightarrow B$ and one tuple violating $B \rightarrow C$, but two tuples violating $A \rightarrow C$. So the computed probability of $A \rightarrow C$ is lower than that of $A \rightarrow B$ and that of $B \rightarrow C$.

In the per-value model, there is one value of A that violates $A \rightarrow B$ and one value of B that violates $B \rightarrow C$, but two values of A that violate $A \rightarrow C$. So again, the computed probability of $A \rightarrow C$ is lower than that of $A \rightarrow B$ and that of $B \rightarrow C$. □

Note that since the augmentation rule and the transitivity rule do not necessarily hold, from pFDs that contain a single attribute on each side, we cannot imply the range of the probability of an FD that contains multiple attributes on one or both sides. If we are required to generate all pFDs, we often have to enumerate each possible FD and compute its probability.

3.4 Applying the basic models

There are two approaches to apply the basic models: one can directly apply them on the different sources, or one can also first merge all source data into a single relation and then apply the basic models on the result relation. We call the former the *separate-data* approach and the latter the *merge-data* approach.

Compared with *separate-data*, the *merge-data* approach has both advantages and disadvantages. On the one hand, merging data together can enrich our knowledge on various possible values of A for a particular value of X and remove some false positives. On the other hand, because of the object-identity issue, the same instances of A can be represented in different ways by different sources, or the same value of X can mean different instances in different sources; thus, simply merging all data may cause false negatives. As an example, consider the data sources in Example 3.1. If we merge data from S_1 and S_2, we will consider that value $b1$ violates $B \rightarrow C$; however, $c1$ and $c1'$ might actually be different representations of the same value. In addition, in applications where we have a huge number of sources or huge volume of source data, it is infeasible to merge all data or even find the majority value of A for each value of X across sources.

Our experiments show that on our data sets, the merge-data approach indeed computes low probabilities for some true FDs and the overall quality of the results is not as good as the separate-data approach.

Parameter settings Our models require setting parameters ϵ_{tb} and ϵ_{tn} (or ϵ_{vb} and ϵ_{vn}) and $P(D)$. We found that in real-world data, ϵ_{tb} is typically much higher than ϵ_{tn}; however, our experimental results show that our results are not sensitive to the initial setting of ϵ_{tb} and ϵ_{tn}. Also, we can re-compute these parameters according to the pFDs we discover and reapply the model with the new parameters until our results converge. We set $P(D) = .5$, as we do not assume any a-priori domain knowledge. As in many other Bayes analysis, our results are not sensitive to the a-priori probability at all.

4. IMPROVING PFD GENERATION

This section studies how we can improve the pFDs we generate according to the basic models. We first study how to improve the results by considering transitivity of FDs (Section 4.1). Then we study when there exist multi-table sources, how we can leverage the normalization already performed at the source side to improve pFDs we generate (Section 4.2).

4.1 Applying the transitivity rule

Although transitivity is one of the most important properties of deterministic FDs, as we show in Example 3.8, results of the basic models may not observe it. We now consider how we can apply the transitivity rule to improve the pFDs we generate.

Model Consider a mediated schema with mediated attributes $\overline{mA}$ and let n be the number of attributes in $\overline{mA}$. There are $n(n-1)$ non-trivial pFDs (a *trivial* pFD is $X \rightarrow X$) where each side contains a single attribute; we denote this set by $\mathcal{D}$. Now consider an FD $D : A \rightarrow B \in \mathcal{D}$. Our goal is to adjust $P(D)$ based on the probability of other FDs in $\mathcal{D}$.

We do the adjustment as follows. We enumerate all subsets of FDs in $\mathcal{D}-\{D\}$. For each such subset $\mathbf{D}$, we compute the probability that $\mathbf{D}$ contains all the true FDs except D, denoted by $P(\mathbf{D})$. We then re-compute the probability of D given the fact that only dependencies in $\mathbf{D}$ are true. The computation uses $P(D)$, computed by basic models, as the a-priori probability of D and the new result is denoted by $P(D|\mathbf{D})$. Finally, we compute the weighted sum of $P(D|\mathbf{D})$ over all subsets of $\mathcal{D} - \{D\}$ as the new probability of D:

$$P(D) = \sum_{\mathbf{D} \subseteq \mathcal{D}-\{D\}} P(D|\mathbf{D})P(\mathbf{D}). \quad (17)$$

We first describe given a subset $\mathbf{D}$, how we compute $P(\mathbf{D})$. If $\mathbf{D}$ violates the transitivity; that is, there exist attributes A, B and C, such that $A \rightarrow B \in \mathbf{D}$, $B \rightarrow C \in \mathbf{D}$, but $A \rightarrow C \notin \mathbf{D}$, then $P(\mathbf{D}) = 0$. Otherwise, as the probabilities we compute for the pFDs rely only on the source data and so are independent of each other, we can compute $P(\mathbf{D})$ as

$$P(\mathbf{D}) = \Pi_{D_0 \in \mathbf{D}} P(D_0) \cdot \Pi_{D_0 \in \mathcal{D}-\mathbf{D}-\{D\}}(1 - P(D_0)). \quad (18)$$

We then normalize the probability of each $\mathbf{D} \subseteq \mathcal{D} - \{D\}$ such that they add up to 1.

We next describe how we compute $P(D|\mathbf{D})$. Intuitively, there are several cases in which our belief of $D : A \rightarrow B$ will be affected.

- If $\mathbf{D}$ contains $A \rightarrow C$ and $C \rightarrow B$, then by the transitivity rule we know $A \rightarrow B$ must hold.
- If $\mathbf{D}$ contains $B \rightarrow C$ but not $A \rightarrow C$, then by the transitivity rule we know $A \rightarrow B$ cannot hold; similarly, if $\mathbf{D}$ contains $C \rightarrow A$ but not $C \rightarrow B$, then $A \rightarrow B$ cannot hold.
- If $\mathbf{D}$ contains $D_1 : A \rightarrow C$ and $D_2 : B \rightarrow C$, our belief of $A \rightarrow B$ will increase. This is because when $A \rightarrow B$ does not hold, the probability that both D_1 and D_2 hold is $P(D_1)P(D_2)$; however, if $A \rightarrow B$ holds, the probability becomes $P(D_2)$, which is higher unless $P(D_1) = 1$. By applying the Bayes rule, we can compute $P(D)$ as

$$\begin{aligned} P(D|D_1, D_2) &= \frac{P(D_1, D_2|D)P(D)}{P(D_1, D_2|D)P(D) + P(D_1, D_2|\bar{D})(1 - P(D))} \\ &= \left(1 + P(D_1)(\frac{1 - P(D)}{P(D)})\right)^{-1}. \end{aligned} \quad (19)$$

Similarly, if $\mathbf{D}$ contains $D_1' : C \rightarrow A$ and $D_2' : C \rightarrow B$, our belief of $A \rightarrow B$ will increase as well:

$$P(D|D_1', D_2') = \left(1 + P(D_2')(\frac{1 - P(D)}{P(D)})\right)^{-1}. \quad (20)$$

Based on the above analysis, we compute the probability of D conditioned on $\mathbf{D}$ as follows.

- If there exists attribute C, such that $A \rightarrow C \in \mathbf{D}$ and $C \rightarrow B \in \mathbf{D}$, then $P(A \rightarrow B|\mathbf{D}) = 1$.
- If there exists attribute C, such that $B \rightarrow C \in \mathbf{D}$ and $A \rightarrow C \notin \mathbf{D}$, then $P(A \rightarrow B|\mathbf{D}) = 0$.
- If there exists attribute C, such that $C \rightarrow A \in \mathbf{D}$ and $C \rightarrow B \notin \mathbf{D}$, then $P(A \rightarrow B|\mathbf{D}) = 0$.
- Otherwise, let $\mathbf{C}_{post}$ be the set of attributes s.t. for each $C \in \mathbf{C}_{post}$, both $A \rightarrow C \in \mathbf{D}$ and $B \rightarrow C \in \mathbf{D}$. Let $\mathbf{C}_{pre}$ be the set of attributes s.t. for each $C \in \mathbf{C}_{pre}$, both $C \rightarrow A \in \mathbf{D}$ and $C \rightarrow B \in \mathbf{D}$. If we denote $\beta = \frac{1-P(D)}{P(D)}$, then,

$$\begin{aligned} & P(D|\mathbf{D}) = P(D|\mathbf{C}_{post}, \mathbf{C}_{pre}) \\ = & \frac{P(\mathbf{C}_{post}|D)P(\mathbf{C}_{pre}|D)P(D)}{P(\mathbf{C}_{post}|D)P(\mathbf{C}_{pre}|D)P(D) + P(\mathbf{C}_{post}|\bar{D})P(\mathbf{C}_{pre}|\bar{D})P(\bar{D})} \\ = & \left(1 + \beta \cdot \Pi_{C \in \mathbf{C}_{post}} P(A \rightarrow C) \cdot \Pi_{C \in \mathbf{C}_{pre}} P(C \rightarrow B)\right)^{-1}. \end{aligned} \quad (21)$$

Note that when $\mathbf{C}_{post} = \mathbf{C}_{pre} = \emptyset$, $P(D|\mathbf{D}) = P(D)$.

Finally, note that if the pFDs generated by the base models are of low quality, applying the transitivity rule can even worsen the results. However, our model works well on pFDs of reasonable quality, as it takes into account all possible subsets of pFDs and the probability of each of them.

Algorithm Our pFD-adjustment algorithm runs iteratively and proceeds in three steps.

1. Compute $P(D)$ for each $D \in \mathcal{D}$ using the by-tuple or by-value model.
2. For each $D \in \mathcal{D}$, adjust the probability of D according to Equation (17), using the probabilities computed in the last round as a-priori probabilities.
3. Repeat step 2 until the change of the probability of each pFD is below a threshold.

Note that adjusting the probability of each $D \in \mathcal{D}$ using Equation (17) requires enumerating all subsets of $\mathcal{D} - \{D\}$ and computing $P(D)$ conditioned on each subset, which takes exponential time. To simplify it, we divide pFDs in $\mathcal{D} - \{\mathcal{D}\}$ into three categories: each pFD whose probability is above a threshold θ_1 is considered as *existing* and is included in every subset $\mathbf{D}$; each pFDs whose probability is below a threshold θ_2 is considered as *non-existing* and is excluded from every subset $\mathbf{D}$; the rest of the FDs are considered as *uncertain*. Therefore, we only need to enumerate including or excluding uncertain FDs when we enumerate the subsets.

We next illustrate the algorithm using an example.

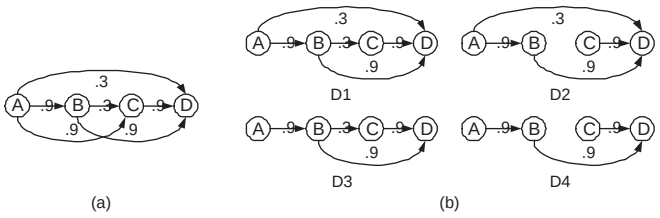

Figure 1: Example 4.1: (a) pFDs generated by a basic model. Each node represents an attribute, each edge represents a pFD with probability $p > .1$, and the weight of an edge represents the probability of the pFD; (b) subsets of pFDs considered for adjusting probability of $A \to C$ in the first round.

Table 1: Probabilities computed for pFDs in each round for Example 4.1. We only show the pFDs whose probabilities change in the iteration.

Rnd	$A \to D$	$B \to C$	$A \to B$	$C \to D$	$A \to C$	$B \to D$
0	.3	.3	.9	.9	.9	.9
1	1	.35	.97	.97	.98	.98
2	1	.36	.97	.97	.98	.98
3	1	.36	.97	.97	.98	.98

EXAMPLE 4.1. *Consider a mediated schema with attributes A, B, C and D. Figure 1(a) shows the pFDs we generate from the data. If we set $\theta_1 = .8$ and $\theta_2 = .2$, the pFDs $B \to C$ and $A \to D$ are uncertain and the rest of the pFDs represented in the graph are existing.*

Table 1 shows the probabilities we re-compute for the pFDs. The algorithm converges at the third round. The probability of $A \to D$ is raised to 1 because of the strong evidence from $A \to B, B \to D$ and $A \to C, C \to D$. The probabilities of the rest of the pFDs in the figure are also raised, but not significantly.

As an example, consider adjusting the probability of $A \to C$ in the first round. Figure 1(b) shows the four subsets of FDs we need to consider. Among them, the last two are invalid, and $\mathbf{D}_1$ and $\mathbf{D}_2$ have probability .3 and .7 respectively. As $\mathbf{D}_1$ contains $A \to B$ and $B \to C$, $P(A \to C|\mathbf{D}_1) = 1$. As $\mathbf{D}_2$ contains $A \to D$ and $C \to D$,

$$P(A \to C|\mathbf{D}_2) = (1 + .3 \cdot \frac{1 - .9}{.9})^{-1} = .97.$$

Thus, we adjust the probability of $A \to C$ to $P(A \to C) = .3 * 1 + .7 * .97 = .98$ in the first round. □

Finally, our iterative algorithm is similar to the *message-passing* algorithm in a loopy belief network in statistics [16]. It has been shown in the literature that the message-passing algorithm is not guaranteed to converge, but a number of researchers have reported excellent experimental results by using this propagation scheme in graphs that has loops [20]. All our experiments reached convergence within 4 iterations.

4.2 Multi-table sources

When a source schema contains multiple relations, some normalization has already been (often carefully) performed at the source side. We can leverage such normalization decision to improve the pFDs we generate.

In particular, consider two attributes X and A. According to their locations in the source tables, they fall into the following four categories.

1. *Case I:* X and A belong to the same relation and X is a key of the relation.

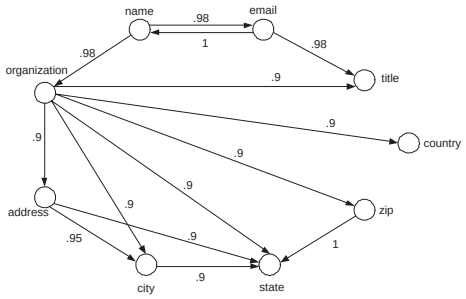

Figure 2: Example 5.1: pFDs for a given mediated schema. To avoid cluttering the graph, we omit pFDs whose probabilities are below .9; we also omit most of the pFDs from name or email to other attributes, whose probabilities range from .95 to 1.

2. *Case II:* X and A belong to the same relation but X is not a key of the relation.
3. *Case III:* X is a key of relation R_0 and A belongs to relation R_n; in addition, there exist relations $R_1, R_2, \ldots, R_{n-1}$, such that for each $i \in [0, n-1]$, R_i has a foreign key referring to a key of R_{i+1}.
4. *Case IV:* X and A belong to different relations but do not fall into Case III.

Intuitively, in Case I and III, it is more likely that $X \to A$, and in the other two cases, it is more likely that $X \not\to A$.

Recall that both the per-tuple model and the per-value model need to compute $P(S|D)$ for each source S. When S has multiple tables, we compute this probability differently: we observe the location of the two involved attributes and decide the probability that they fall into the particular case. The result is denoted by $P_m(S|D)$.

Formally, let ϵ_m be the probability that $D : X \to A$ is true but X and A fall into Case II and IV; in other words, ϵ_m is the probability of missing normalization. On the other hand, let ϵ_f be the probability that D is false but X and A fall into Case I and Case III; in other words, ϵ_f is the probability of false normalization. Thus, in Case I and III,

$$P_m(S|D) = 1 - \epsilon_m; \quad (22)$$
$$P_m(S|\bar{D}) = \epsilon_f. \quad (23)$$

In Case II and IV, if we denote by $P(S|D)$ and $P(S|\bar{D})$ the probabilities we compute using the basic models, we have

$$P_m(S|D) = \epsilon_m \cdot P(S|D); \quad (24)$$
$$P_m(S|\bar{D}) = (1 - \epsilon_f) \cdot P(S|\bar{D}). \quad (25)$$

Finally, if keys and foreign-keys are not specified with the schema, we can discover them by examining cardinality of attribute values and inclusion of values of one attribute in those of the key of another table. We ignore the details for space consideration.

5. NORMALIZING MEDIATED SCHEMAS

As an application of pFDs, we next consider how we can normalize a mediated schema, possibly generated automatically, in a pay-as-you-go data integration system.

Typically, a relation is normalized by finding FDs that violate the BCNF or 3NF normal form and splitting the relation accordingly. However, directly applying such normalization is inadequate in our context, as illustrated in the following example.

EXAMPLE 5.1. *Consider a mediated schema and a set of automatically generated pFDs, shown in Figure 2. If we consider the pFDs with a probability of no less than .9 as deterministic and apply BCNF normalization, we can get several normalization results, of which one is as follows.*

Business-contact(name, email, organization)
Org(organization, address, zip, country, title)
Addr(address, city)
City(city, state)

This result has several problems. First, organization → title, address → city *and* city → state *do not hold in general, so the result schema is imprecise: it incorrectly groups* title *with attributes for organization, and can fail in representing cities with the same name but in different states. Second,* address, city, state, zip, country *are all attributes that describe addresses; splitting them into several relations is not desired.* □

We next describe how we solve the problems.

Selecting pFDs A naive way of normalization is to generate all pFDs, prune some of them by applying a threshold, consider the remaining ones as deterministic, and apply normalization accordingly. However, the data can be dirty or incomplete and so the pFDs whose probabilities are above the given threshold may be incorrect. We thus do a further pruning according to the following heuristics.

HEURISTICS 5.2. *Each non-key attribute belongs to one and only one entity or relationship.* □

Following this heuristics, for each attribute A we only need to select one pFD that contains A on the right-hand side; other pFDs either can be implied by transitivity, or are unlikely to be correct. We thus prune pFDs as follows. We use threshold η as the minimum probability of a pFD being considered as likely to be true and δ as the maximum difference between two selected pFDs with the same attribute on the right-hand side. We prune a pFD $X \rightarrow^p A$ if one of the following three conditions holds.

- $p < \eta$;
- there exists attribute Y such that $Y \rightarrow^{p'} A, p' - p > \delta$;
- there exists attribute Y such that $Y \rightarrow^{p'} A, p' > p$, and there does not exist a set of attributes $Z_1, \ldots, Z_l, l \geqslant 0$, for which each of $X \rightarrow Z_1, Z_1 \rightarrow Z_2, \ldots, Z_l \rightarrow Y$ has a probability above η or each of $Y \rightarrow Z_l, \ldots, Z_2 \rightarrow Z_1, Z_1 \rightarrow X$ has a probability above η.

Consider Example 5.1. If we set $\eta = .9$ and $\delta = .05$, we prune pFDs not in the graph and also pFDs organization→title, organization→state, address→state, and city→state.

Avoiding oversplitting After the pruning, we consider the remaining pFDs as deterministic. Rather than applying BCNF normalization, we apply a dependency-preserving 3NF normalization, such that dependency between attributes are preserved and so an attribute will remain in the same relation as the attribute that represents the key of the object. Note that when we consider single-attribute FDs, the result of 3NF normalization is also in BCNF normal form.

The normalization requires recognizing keys of the schema. We consider an attribute K as a key if it is not determined by any other attribute, or if it is determined by K', but there is also an FD $K \rightarrow K'$.

```
NORMALIZATION (Med)
1  rels ← {Med}; n ← 0; cont ← true;
2  while cont do
3    cont ← false;n++;
4    for each R ∈ rels do
5      if #attrs in R > n + 1 then
6        cont ← true;
7        // Generate pFDs with n attributes on the left-hand side
8        pFDs ← GENERATEPFDS(R, n);
9        pFDs ← PRUNEPFDS(pFDs); // Prune inappropriate pFDs
10       keys ← FINDKEYS(R, pFDs); // Find key of the relation
11       // Remove FDs that can be derived by transitivity
12       pFDs ← MINSET(pFDs);
13       // 3NF Normalization
14       for each D : X̄ → A ∈ pFDs do
15         if X̄ ∉ keys then
16           Normalize R according to D;
17       endif endfor
18   endif endfor
19   rels ← (rels − {D}) ∪ {relations of normalization results};
20 endwhile
21 rels = MERGERELS(rels); //Merge tables to avoid splitting
22 return rels;
```

Figure 3: Algorithm NORMALIZATION: normalize a mediated schema.

Note that strictly following 3NF normalization can generate relations with only a few attributes. To avoid oversplitting, we examine each table T with no more than k (a pre-defined threshold) attributes; if there exists a table with the key of T, we merge T with that table.

Continue with Example 5.1. After we apply pruning, a dependency-preserving 3NF normalization obtains the following relations.

Business-contact(name, email, title, organization)
Org(organization, address, zip, country)
Addr(address, city)
Zip(zip, state)

If we set $k = 2$, we merge the last three tables into one and obtain the following results, effectively identifying two entities in the domain.

Business-contact(name, email, title, organization)
Org(organization, address, city, state, zip, country)

Algorithm NORMALIZATION (Figure 3) gives the complete algorithm for mediated-schema normalization. Note that the normalization first considers only pFDs where each side contains a single attribute. When there are no more selected pFDs that violate a 3NF, it explores pFDs where the left side contains multiple attributes. In this way, we can significantly reduce the number of multi-attribute pFDs for which we need to compute probability.

6. EXPERIMENTS

We now describe a set of experiments on real-world data sets extracted from the Web. Our first goal in these experiments is to examine the quality of the pFDs we generated; our second goal is to examine the quality of our normalization based on the pFDs.

6.1 Experimental setup

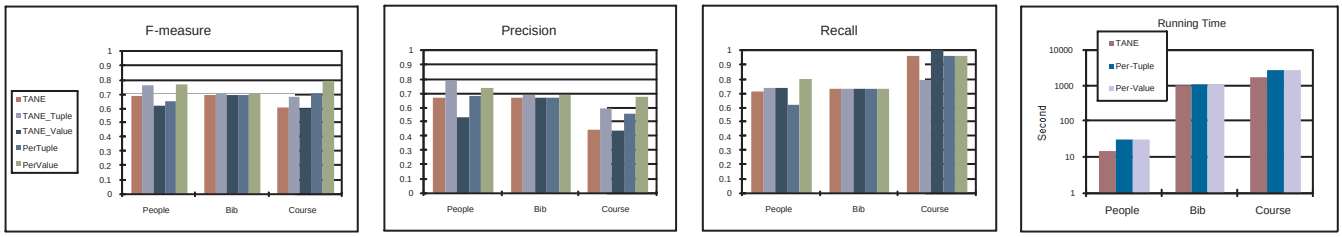

Figure 4: Quality of FDs and execution time generated by various methods in different domains.

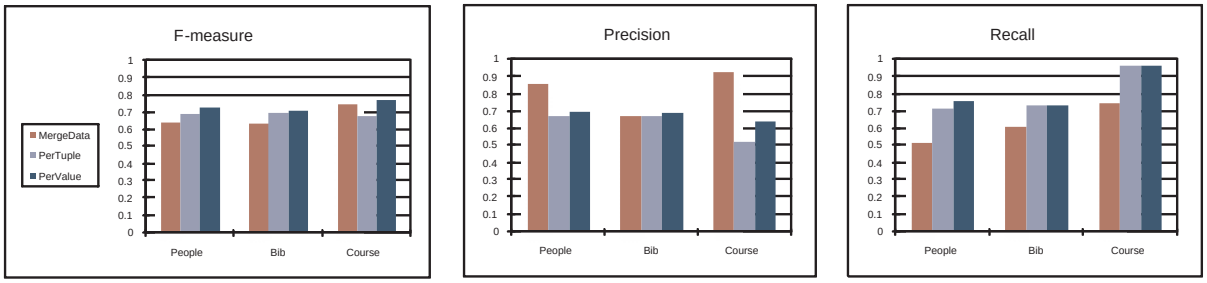

Figure 5: Precision, recall and F-measure of FDs generated by different basic models.

Table 2: Characteristics of data sources in each domain. For each domain, we report the number of sources, the average size (number of tuples) of the source tables, and the keywords that identify the domain.

Dom	#Src	Avg size	Keywords
People	49	63	*name*, one of *job* and *title*, and one of *organization, company* and *employer*
Bib	647	48	*author, title, year*, and one of *journal* and *conference*
Course	646	57	one of *course* and *class*, one of *instructor, teacher* and *lecturer*, and one of *subject, department* and *title*

We implemented the algorithms we described in this paper and built the ERDis (Entity/Relationship Discovery) system. Taking a set of data sources, ERDis first applies the methods described in [18] to automatically generate a single-table mediated schema and mappings from the source schemas to the mediated schema. ERDis then generates pFDs where both sides contain a single attribute in the mediated schema. According to the pFDs, ERDis normalizes the mediated schema into a set of relations. We implemented ERDis in Java and stored data using the Derby DBMS [1]. We conducted experiments on a Windows XP work station with Intel Core 1.66GHz CPU and 1GB memory.

We experimented on a set of HTML tables extracted from the Web and select the tables that have clear attribute labels. We considered tables from three domains: Business-Contact(People), Bibliography(Bib) and Course. For each domain, we identify tables in this domain by searching tables that contain certain keywords in the attribute labels (see Table 2). The number of tables in each domain vary from 49 to 200. The number of tuples in each source table varies from 6 to 1123, and on average is 53. For each domain, we manually normalized the automatically generated mediated schema according to our domain knowledge (shown in the second column of Table 5). Each mediated schema contains two or three relations and the number of attributes in each relation varies from 2 to 10. We considered each HTML table as a (single-table) data source and created a database for sources in each domain.

For pFD generation, we implemented both the per-tuple model and the per-value model, and also implemented several baseline methods for comparison:

- TANE: Extend the method proposed in TANE [9] to multiple sources. Specifically, let $\mathcal{S}$ be the set of data sources and let $D : X \rightarrow A$ be the FD we examine. For each source $S \in \mathcal{S}$, find the set $\mathbf{O}(S)$ of tuples in S that observe D (as described in Section 3.1). Compute the confidence of D as
$$P(D) = \frac{\sum_{S\in\mathcal{S}} |\mathbf{O}(S)|}{\sum_{S\in\mathcal{S}} |S|}.$$
- TANETuple: The same as TANE except that we compute the confidence on each source and then take the average. In other words,
$$P(D) = Avg_{S\in\mathcal{S}} \frac{|\mathbf{O}(S)|}{|S|}.$$
- TANEValue: The same as TANETuple except that for each source, we compute the confidence as the percentage of values of X for which the tuples observe D. (This method has similar spirit as PerValue).
- MergeData: First merge data from different sources, then apply the per-tuple model.
- PerTuple: Apply the per-tuple model and then adjust the results by applying the transitivity rule.
- PerValue: Apply the per-value model and then adjust the results by applying the transitivity rule.

For MergeData, PerTuple and PerValue, we set $\epsilon_{tn} = \epsilon_{vn} = .01, \epsilon_{tb} = \epsilon_{vb} = .2$ and $P(D) = .5$ (the a-priori probability of a dependency). When we apply adjustment according to the transitivity rule, we set $\theta_1 = 0.9$ and $\theta_2 = 0.5$.

We measured quality of the generated FDs as follows. For each domain, we manually generated the set of true FDs over the mediated schema. For each method, we took the set of FDs whose computed probability is above .8 and compared it with the golden standard. We reported *precision, recall* and *F-measure* of our results. Let $\mathbf{D}_G$ be the set of FDs in the golden standard and $\mathbf{D}_R$ be the set of automatically

Table 3: Improvement of applying the transitivity rule over the basic models. We report the number of FDs on which we changed decisions.

Domain	Per-value			Per-value w. transitivity						
	Quality			Quality			#FDs w. inc pr		#FDs w. dec pr	
	F-msr	Prec	Rec	F-msr	Prec	Rec	Total	Correct	Total	Correct
People	.723	.694	.756	.766	.735	.8	5	4	5	3
Bib	.706	.686	.727	.706	.686	.727	0	0	0	0
Course	.767	.639	.958	.793	.676	.958	0	0	2	2

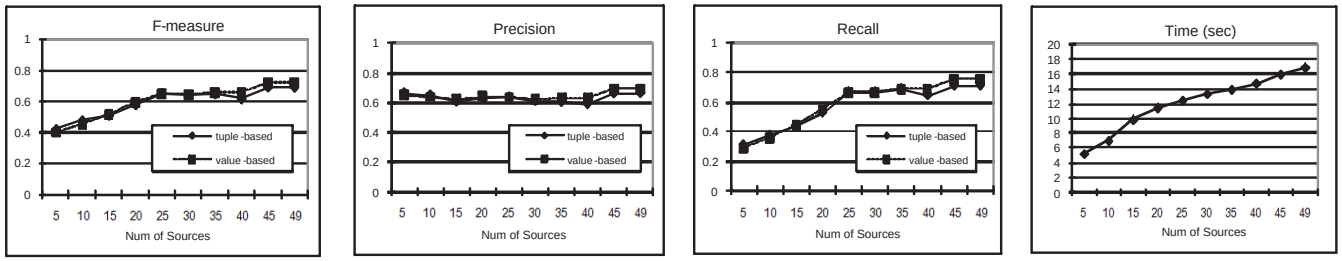

Figure 6: Effect of more sources on quality of the pFDs we generate and on execution time.

generated FDs. Then, precision is defined as $P = \frac{|\mathbf{D}_G \cup \mathbf{D}_R|}{|\mathbf{D}_R|}$, recall is defined as $R = \frac{|\mathbf{D}_G \cup \mathbf{D}_R|}{|\mathbf{D}_G|}$, and F-measure is computed as $F = \frac{2PR}{P+R}$.

We describe the various methods we implemented for normalization and measurement of results in Section 6.3.

6.2 Generating pFDs

Figure 4 shows the precision, recall and F-measure of results of different FD-generation methods. We observe that PERVALUE obtains the highest F-measure, on average .75, in all domains. It has higher precision and recall than PERTUPLE in all domains and on average increases precision by 10.3% and increases recall by 9.5%. Among the various adaptions of the TANE model, TANETUPLE obtains the best results: on the People and Bib domains, the F-measure of its results is the same as or slightly worse than PERVALUE; however, on the Course domain, the F-measure of PERVALUE is still 16.9% better than that of TANETUPLE. In fact, PERVALUE obtains better or the same precision and recall compared with TANETUPLE in all domains, except a lower precision in the People domain. This is because the People domain contains fewer sources and there happens to be big data sources that observe a false FD; PERVALUE gives higher weight to larger tables but TANETUPLE treats tables of different sizes as the same. We also observe that although TANEVALUE and PERVALUE both consider violation of FDs at the value granularity, PERVALUE considers variety of violations and so obtains much higher precision than TANEVALUE, on average 33% higher.

Figure 4 also shows the execution time of TANE, PERTUPLE and PERVALUE (TANETUPLE and TANEVALUE have similar execution time to TANE). Given a data set with around 650 sources, PERTUPLE and PERVALUE finished in about 45 minutes. Because of applying the transitivity rule, our algorithms took about half time longer than TANE; however, finding FDs is often a one-time process and the execution time is still acceptable.

We next examine contributions of different components of our algorithm to the quality of the results.

Basic models: We first compared results of different basic models, namely, per-tuple, per-value, and merge-data (Figure 5). To isolate performance of the basic models from enhancements, we did not apply the transitivity rule in this set of experiments. We observe that PERVALUE obtains higher precision and similar recall compared with PERTUPLE in all domains. MERGEDATA typically obtains higher precision but much lower recall, and so lower F-measure, because of the object-identity problem. Indeed, the recall of PERVALUE is 35% higher than that of MERGEDATA, and the F-measure of PERVALUE is 13% higher.

Applying transitivity rule: We next examined improvements of applying enhancements over the basic models. Table 3 shows the effect of applying the transitivity rule over the per-value model. We observe that applying the transitivity rule in most cases increases the probability of a true FD while decreases the probability of a false dependency. It increases both precision and recall; in particular, it increases both precision and recall by 5% in the People domain, and increases the precision by 5% in the Course domain.

Effect of more sources: We also examined the effect of having more data sources on the quality of the FDs we generated. We randomly ordered sources in the People domain, ran our models initially on the first 5 sources, and then gradually added more sources. We applied the per-value model and the per-tuple model without transitivity enhancements. The results are plotted in Figure 6. We observe that having more sources tend to improve recall of the discovered FDs for both models and also improve precision of the per-value model slightly. Overall, the F-measure increased by 62.2% for per-tuple model and by 80.9% for per-value model. We also observed that execution time increased linearly with the number of sources, showing that our algorithm scales well when we have more sources.

Effect of parameters: We varied parameters in the model. We observed that when we vary $\epsilon_{vb}(\epsilon_{tb})$ from .2 to .6, $\epsilon_{vn}(\epsilon_{vb})$ from .000001 to .1, and $P(D)$ from .3 to .8, the F-measure of PERVALUE and PERTUPLE do not change much.

Multi-table Sources: Finally, we examined the effect of multi-table sources on the quality of pFDs. In each domain, we randomly selected half of the sources and applied BCNF according to FDs in the golden standard, and left the rest of the sources unchanged. We applied our algorithm and set $\epsilon_m = \epsilon_f = .1$. Table 4 shows the improvement of the results in each domain. We observe that leveraging normalization

Table 5: Comparison of manually generated mediated schemas and those generated by ERDIS. Misclassified attributes in the results of ERDIS are highlighted. ERDIS obtained high-quality mediated schemas.

Dom	Golden standard	Results of ERDIS	#(Mis attrs)	F-msr	Precision	Recall
People	R1(name, email, phone, fax, title, organization) R2(organization, address, city, state, zip, country)	R1(name, email, phone, fax, title, *address*, organization) R2(organization, city, state, zip, country)	1	.85	.84	.87
Bib	R1(paperID, title, abstract, authors, journal, volume, issue, year, subject, source) R2(journal, issn, eissn)	R1(paperID, title, abstract, authors, journal, volume, issue, year, source) R2(journal, issn, eissn, *subject*)	1	.78	.94	.67
Course	R1(courseID, catelogID, section, time, days, term, location, instructor, fee) R2(catelogID, title, subject, credits) R3(instructor, institution)	R1(courseID, catalogID, section, time, days, term, instructor) R2(catalogID, title, subject, credits, *location*) R3(instructor, institution, *fee*)	2	.63	.60	.66

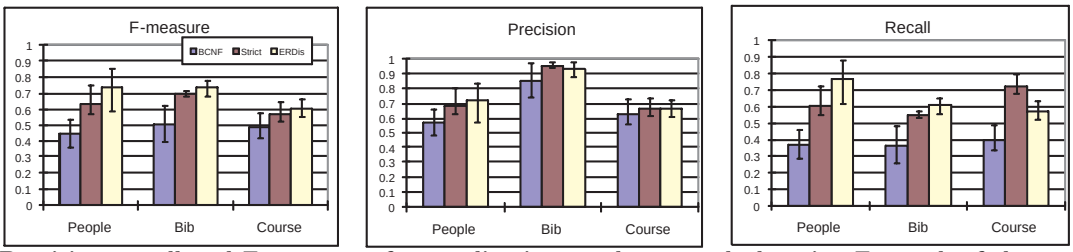

Figure 7: Precision, recall and F-measure of normalization results on each domain. For each of them, we show the average, minimum and maximum value.

Table 4: Improvement of considering normalization at the source side.

	Single-table sources			Multi-table sources		
Domain	F-msr	Prec	Rec	F-msr	Prec	Rec
People	.766	.735	.8	.8	.8	.8
Bib	.706	.686	.727	.743	.703	.788
Course	.793	.676	.958	.842	.727	1

at the source side can improve our results: it improves the F-measure by 5.3% on average; on the Bib domain, it improves recall by 8.4%; on the People and Course domains, it improves precision by 15.3% and 13.8% respectively.

6.3 Normalization

We implemented three normalization methods:

- BCNF selects all pFDs whose probability is no less than .9 and applies BCNF normalization;
- STRICT prunes FDs as described in Section 5 and applies dependency-preserving 3NF normalization;
- ERDIS applies Algorithm NORMALIZATION and is different from STRICT by avoiding oversplitting.

In STRICT and ERDIS, we set $\eta = .9, \delta = .02$, and in ERDIS, we set $k = 2$.

We compared the results of each method with the golden standard we manually created. As normalization essentially can be viewed as clustering attributes into entities, we measure the quality of the results using *precision*, *recall* and *F-measure*, standard for clustering. Specifically, let $\mathbf{A}$ be the set of attributes in the mediated schema. Let S be the schema generated by one of our algorithms and G be the golden standard. For each pair of attributes $A, A' \in \mathbf{A}$ that occur in the same relation in S, we say they are correctly clustered if they also occur in the same relation in G. We denote by *Corr(S,G)* the number of correctly clustered attribute pairs, by *Total(S)* the number of attribute pairs that occur in the same relation in S, and by *Total(G)* the number of attribute pairs that occur in the same relation in G. Then, we define precision as $P = \frac{Corr(S,G)}{Total(S)}$, recall as $R = \frac{Corr(S,G)}{Total(G)}$, and F-measure as $F = \frac{2PR}{P+R}$. Note that depending on the order of pFDs we apply for normalization, we can generate different results in different runs. We ran each method 10 times and report the average, minimum and maximum values.

Table 5 shows the normalization results of ERDIS (among multiple possible results, the table shows the most frequent one). We observe that although the F-measure is not high, the normalization results are actually very close to the golden standard. This is because there are only two or three relations in the golden standard in each domain, so once we put one attribute into a wrong relation, the precision and recall will be reduced significantly.

Figure 7 shows the quality of normalization results by each method. We observe that ERDIS has the highest precision and recall; on average it improves the precision over BCNF by 13.7% and improves the recall by 73.3%. Avoid oversplitting can also improve quality of the results: ERDIS improves the F-measure over STRICT by 9.3% on average.

7. RELATED WORK

Our work is close to two bodies of work: generation of functional dependencies from data and creation of mediated schemas in data integration systems.

Generating functional dependencies: The idea of deriving FDs from data was originally proposed in TANE [9] and CORDS [11], and they call the results *approximate FDs* or *soft FDs*. Given a relation R, both systems compute the probability (or confidence) of an FD $X \rightarrow A$ as the maximal percentage of R's tuples that satisfy the FD. Our

strategy is different from theirs in several aspects. First, we develop probabilistic models suitable for multiple data sources. In our model, consensus among data sources will strengthen our belief (Theorem 3.2), but a simple extension of the TANE model to multiple sources does not have this feature. Second, we develop the per-value model, which considers violation of an FD at the value granularity and consider the variety of A values in the violation tuples. Third, our algorithm adjusts the FDs discovered from the data by considering transitivity of FDs and by leveraging possible normalizations already performed at the source side; these are not considered in TANE or CORDS. We describe experimental comparison between our models and the methods in TANE and CORDS in Section 6.

Recently, there has been work on discovering *conditional FDs* from data [7]. A conditional FD associates an FD with a *pattern tableau* stating the conditions of tuples that observe the FD [5]. Discovery of conditional FDs thus focuses on generating the best condition tableau. The notion of pFD that we propose has different goals: the probability of an FD captures our uncertainty on whether an FD holds in the underlying domain, given the possible noise and incompleteness of data from different sources. Our pFD discovery algorithms thus focus on applying probabilistic analysis on the observation of whether data from various sources observe or violate an FD.

Creating mediated schemas

While there has been recent interest in developing techniques for improving mediated schemas in a data integration system, these techniques are limiting in at least one of the following aspects: (1) they are not fully automatic, and (2) they create an un-normalized single-table mediated schema.

A large body of previous work [2, 3, 10, 12, 14, 17, 15, 19] focused on semi-automatically merging source schemas to create a mediated schema, where ambiguity needs to be resolved by users. More recently Chiticariu [4] developed a system that enumerates multiple promising mediated schemas for a user to select from to solve the entity mis-alignment problem in presence of multi-table schemas.

Sarma et al. [18] proposed automatically generating a probabilistic mediated schema from source schemas and then consolidating them to provide a single-table mediated schema; however, the mediated schema is not normalized. He and Chang [8] considered co-occurrence of attributes in source schemas to automatically create a *generative model* that is maximally consistent with the source schemas. The results, again, are single-relation schemas and there is no discussion of normalization of the result. Finally, Magnani et al.[13] proposed using probabilistic relationships between entities to generate a set of alternative mediated schemas; however, they did not consider categorizing attributes into entities.

We study the problem of normalizing a (possibly automatically generated) mediated schema. Our normalization is based on the pFDs we generate from data and is fully automatic.

8. CONCLUSION AND FUTURE WORK

This paper presents a technique that discovers FDs from heterogeneous data sources in a pay-as-you-go data integration system. We propose the notion of probabilistic functional dependencies and develop Bayes models to automatically compute probabilities of FDs according to data from different data sources. As an application, we apply the pFDs we discover for normalization of automatically generated mediated schemas. Experimental results show high precision and recall of the FDs we discover and high quality of our normalized mediated schemas.

There are several directions to explore for future work. First, we would like to consider how to improve scalability of FD discovery by sampling in presence of large data sources. Second, we would like to extend our work for generation of mediated schemas in data integration systems where data can be in multiple underlying domains and each domain can be complex (with many types of objects and relationships).

9. REFERENCES

[1] Apache Derby. http://db.apache.org/derby/.

[2] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. In *ACM Computing Surveys*, pages 323–364, 1986.

[3] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *Proc. of EDBT*, 1992.

[4] L. Chiticariu, M. A. Hernandez, P. G. Kolaitis, and L. Popa. Semi-automatic schema integration in clio. In *Proc. of VLDB*, 07. Demonstration description.

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

[6] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4):27–33, 2005.

[7] Lukasz Golab, Howard J. Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[8] B. He and K. C. Chang. Statistical schema matching across web query interfaces. In *Proc. of ACM SIGMOD*, 2003.

[9] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[10] R. Hull. Relative information capacity of simple relational database schemata. In *Proc. of ACM PODS*, 1984.

[11] I. F. Ilyas, V. Markl, P. J. Haas, P. G. Brown, and A. Aboulnaga. Cords: Automatic generation of correlation statistics in db2. In *Proc. of VLDB*, 2004.

[12] L. A. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *Proc. of EDBT*, 1990.

[13] M. Magnani, N. Rizopoulos, P. Brien, and D. Montesi. Schema integration based on uncertain semantic mappings. *Lecture Notes in Computer Science*, pages 31–46, 2005.

[14] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. of VLDB*, 1993.

[15] A. Motro. Superviews: Virtual integration of multiple databases. In *IEEE Transactions on Software Engineering*, pages 785–798, 1987.

[16] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. 1988.

[17] R. Pottinger and P. Bernstein. Creating a mediated schema based on initial correspondences. In *IEEE Data Eng. Bulletin*, pages 26–31, Sept 2002.

[18] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proc. of ACM SIGMOD*, 2008.

[19] J. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase integrating heterogenous distributed database systems. In *Proc. of AFIPS*, 1981.

[20] Y. Weiss. Correctness of local probability propagation in graphical models with loops, 1998.