Functional Dependency Generation and Applications in pay-as-you-go data integration systems[*]

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

ABSTRACT

Recently, the opportunity of extracting structured data from the Web has been identified by a number of research projects. One such example is that millions of relational-style HTML tables can be extracted from the Web. Traditional data integration approaches do not scale over such corpora with hundreds of small tables in one domain. To solve this problem, previous work has proposed *pay-as-you-go* data integration systems to provide, with little up-front cost, base services over loosely-integrated information. One key component of such systems, which has received little attention to date, is the need for a framework to gauge and improve the quality of the integration. We propose a framework based on functional dependencies(FDs). Unlike in traditional database design, where FDs are specified as statements of truth about all possible instances of the database; in web environment, FDs are not specified over the data tables. Instead, we generate FDs by counting-based algorithms over many data sources, and extend the FDs with probabilities to capture the inherent uncertainties in them. Given these probabilistic FDs, we show how to solve two problems to improve data and schema quality in a pay-as-you-go system: (1) pinpointing dirty data sources and (2) normalizing large mediated schemas. We describe these techniques and evaluate them over real-world data sets extracted from the Web.

1. INTRODUCTION

Recently, a number of research projects [4, 2, 6] have identified the exciting opportunity to collect structured data from the Web. For example, the WebTables project extracted millions of high-quality relational-style HTML tables from web pages [4]. Another example is the use of information extraction tools over web documents, which automatically construct corpora with large numbers of structured entities [2, 6]. As described in [16], traditional integration systems do not scale over such corpora, because (1) hundreds and thousands of data sources commonly exist for a given domain; (2) the data tables are usually small and incomplete; and (3) no one has control over all the sources.

One promising approach to solve the above problem is to use a pay-as-you-go data integration system [9], which provides, with little up-front cost, base services over loosely-integrated information. Such a system allows users, administrators, and programmers to focus their integration energies on the most fruitful parts of the data given limited human and machine resources. Previous work [16] has investigated building a pay-as-you-go data integration system over many structured data sources extracted from the Web. At the heart of the challenge of designing a pay-as-you-go system is the need for a framework to gauge and improve the quality of the information integration. However, little attention has been paid to the development of such a framework.

In this paper, we propose one framework based on functional dependencies (FDs) as a method for measuring and improving the quality of integration. Traditionally, FDs are statements of truth about all possible instances of the database, and are provided top-down as part of the database design process. They have been thoroughly researched and applied to improving schema quality through normalization [5, 3, 13, 14, 17] and to improving data quality in data cleaning [7, 8, 10]. In contrast, a pay-as-you-go integration system over a large corpus of tables extracted from the Web, do not have up-front knowledge of the FDs and an integrated view of the data must be created from bottom-up over many diverse data sources. To address this problem, we generate FDs from statistics over the data and extend the FDs with probabilities to capture the inherent uncertainty of the FDs learned in this manner.

The problem of automatic discovery of FDs from data has been addressed in previous work [11, 12]. Our approach is to extend the techniques such as TANE [11], which are designed for and work well on single large data tables, to handle corpus with many small data sources in one domain. We develop the notion of *probabilistic functional dependencies (pFDs)*. Given a mediated schema and the mappings from each source to the mediated schema, generated automatically [16], we describe counting-based algorithms for deriving pFDs and their probabilities from the data in various sources. Given these pFDs, we show how to solve two problems that arise in a pay-as-you-go integration system.

First, we show that the violation of pFDs by some data sources can help pinpoint data sources with low quality data. The types of dirtiness that can be discovered by checking pFDs includes: attributes with dummy values, entity ambiguities, nested attributes and incorrect schema mappings.

Second, just as deterministic FDs are used in traditional databases for schema normalization, we use the generated pFDs to help normalize a large automatically generated me-

[*]This work was supported in part by the NSF under grant NSF IIS-0415175.

diated schema into relations that correspond to meaningful real-world entities and relationships, to help users better understand the underlying data.

In this paper we describe and evaluate the techniques we developed for generating pFDs, and for discovering dirty data sources and normalizing large mediated schema using them. Our results show that these techniques obtain good precision and recall in functional dependency generation and obtain promising results for discovering dirty data sources and normalizing mediated schemas.

2. GENERATING PROBABILISTIC FDS

This section introduces the key component of our solution: *probabilistic functional dependencies (pFDs)*. First, we formally define pFDs, then we discuss how to generate them from statistics on a set of data sources.

2.1 Definition

Functional dependencies (FDs) were originally developed as part of Relational Database theory for improving the quality of schemas. An FD $\bar{X} \rightarrow \bar{Y}$ indicates a relationship between sets of attributes $\bar{X}$ and $\bar{Y}$, such that any two entities that share a value for $\bar{X}$ must also share a value for $\bar{Y}$. FDs generalize the notion of a key in relational databases, and as such, provide important semantic clues about data consistency.

In a pay-as-you-go integration system, we cannot assume domain knowledge of the FDs. One way to generate FDs is by observing the data. However, the FDs we can infer from the data are inherently uncertain because (1) FDs are statements provided top-down as the truth over all data instances, whereas FDs learned bottom-up from some data instances might not hold in general; and (2) real-world data sources are often dirty with missing values, inconsistent values, etc. We define probabilistic functional dependencies to incorporate such uncertainty.

Definition 2.1. (Probabilistic Functional Dependency (pFD)) *Let R be a relation, $\bar{X}$ be a set of attributes in R, and A be an attribute in R. A* probabilistic functional dependency *is denoted by $\bar{X} \rightarrow^p A$, where p is the likelihood of $\bar{X} \rightarrow A$ being correct. A pFD is correct if it holds for all the data instances in the domain.* □

Next we describe how the probability of a pFD can be statistically computed from a set of data sources.

2.2 Computing probabilities

Single source: We first consider how to compute the probability of $\bar{X} \rightarrow A$ from a single source R. Ideally, if $\bar{X} \rightarrow A$ holds, all tuples with the same value of $\bar{X}$ should have the same value of A. However, as we may have noisy data, there can exist tuples whose A-value is different from that in the majority of tuples with the same value for $\bar{X}$. One way to compute the probability of $\bar{X} \rightarrow A$ is: first compute the fraction of such tuples for each distinct value of $\bar{X}$, and then compute the probability of $\bar{X} \rightarrow A$ using it. We call this PerValue algorithm.

1. First, for each distinct non-null [1] value V_X of $\bar{X}$, we find the non-null A-value V_A that occurs in the maximum number of tuples with value V_X for X. The probability that $\bar{X} \rightarrow A$ holds for tuples with value V_X, denoted by $Pr(\bar{X} \rightarrow A, V_X)$, is the fraction of tuples with value V_X and V_A over all tuples with value V_X. Formally, let $|V_A, V_X|$ be the number of tuples with values V_X for $\bar{X}$ and V_A for A, and $|V_X|$ be the number of tuples with values V_X for $\bar{X}$ and non-null value for A. We compute the probability as follows:

$$Pr(\bar{X} \rightarrow A, V_X) = \frac{|V_A, V_X|}{|V_X|}. \qquad (1)$$

2. Second, we compute the probability of $\bar{X} \rightarrow A$ as the average of probabilities of $\bar{X} \rightarrow A$ holding for each distinct non-null value of $\bar{X}$. Formally, let $\mathcal{D}_{\bar{X}}$ be all distinct values of $\bar{X}$ in R and $|\mathcal{D}_{\bar{X}}|$ be the size of $\mathcal{D}_{\bar{X}}$. We define the probability of $\bar{X} \rightarrow A$ as

$$Pr(\bar{X} \rightarrow A, R)_{\text{PerValue}} = \frac{\sum_{V_X \in \mathcal{D}_{\bar{X}}} Pr(\bar{X} \rightarrow A, V_X)}{|\mathcal{D}_{\bar{X}}|} \qquad (2)$$

[1]None of the attribute values in X is NULL.

```
GET-FDPROB $(R, X, A)$
SORT$(R, \{X, A\})$
$c \leftarrow t_1(X); \mid \pi(X) \mid \leftarrow 1; count(c) \leftarrow 0$
$c' \leftarrow t_1(X, A); count(c') \leftarrow 0; sum \leftarrow 0; maxCount(c) \leftarrow 0$
for each $t \in R$ do
   if $t(X) == c$ then
      $count(c) \leftarrow count(c) + 1$
      if $t(X, A) == c'$ then
         $count(c') \leftarrow count(c') + 1$
      else
         if $maxCount(c) < count(c')$ then
            $maxCount(c) \leftarrow count(c')$
         endif
         $c' \leftarrow t(X, A); count(c') \leftarrow 0$
      endif
   else
      $sum \leftarrow sum + maxCount(c)/count(c)$
      $c \leftarrow t(X); \mid \pi(X) \mid \leftarrow \mid \pi(X) \mid + 1$
      $count(c) \leftarrow 0; maxCount(c) \leftarrow 0$
endif endfor
return $\frac{sum}{|\pi_X|}$
```

Figure 1: The PerValue algorithm to compute the probability p of FD $\bar{X} \rightarrow A$ over a single source table R.

An alternative is to take the average of the probabilities $Pr(\bar{X} \rightarrow A, V_X)$ weighted by the frequency of each particular value of $\bar{X}$; in other words, the probability of $\bar{X} \rightarrow A$ on a value of $\bar{X}$ that occurs often should affect our belief of $\bar{X} \rightarrow A$ more than the probability on a less frequent value. We call this the PerTuple algorithm. Using the same notation as above, the PerTuple probability can be computed by:

$$Pr(\bar{X} \rightarrow A, R)_{\text{PerTuple}} = \frac{\sum_{V_X \in \mathcal{D}_{\bar{X}}} |V_A, V_X|}{\sum_{V_X \in \mathcal{D}_{\bar{X}}} |V_X|} \qquad (3)$$

We can compute the probability of a pFD in linear time in the size of the data, by scanning the source data and grouping by distinct $\bar{X}$ values. The detailed algorithm for PerValue is shown in Figure 1.

Multiple sources: When we have multiple data sources, one approach to computing the probability of an FD is to first merge data from different sources into one table, based on the mediated schema and the schema mapping, and then apply either the PerValue or the PerTuple algorithm.

Dom	#Src	Avg size	Keywords
People	45	63	*name*, one of *job* and *title*, and one of *organization, company* and *employer*
Bib	619	48	*author, title, year*, and one of *journal* and *conference*
Course	545	57	one of *course* and *class*, one of *instructor, teacher* and *lecturer*, and one of *subject, department* and *title*

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

We call this approach MergeData. This approach can significantly enrich the data upon which we generate the pFDs to avoid being biased by small source tables with incomplete information. However, it also introduces noise because the same entity can have different presentations in different sources, and because the effect of an incorrect schema mapping gets amplified with big source tables.

An alternative approach is to compute the probability an FD by first computing the probability of this FD on each source and then merging them over the mediated schema using the schema mapping. We call this approach MergeFD. Specifically, our computation is based on a single-table mediated schema M and a set of mappings between M and the source schemas. Consider an FD $\bar{X} \to A$ on M and let $\bar{S}$ be a set of data sources in which each attribute in $\bar{X}$ and A is mapped to a distinct attribute. We compute the probability of $\bar{X} \to A$ on each source schema in $\bar{S}$ and take the average as the resulting probability:

$$Pr(\bar{X} \to A, \bar{S})_{\text{MergeFD}} = \sum_{R \in \bar{S}} Pr(\bar{X} \to A, R) \qquad (4)$$

For simplicity, we restrict ourselves to FDs with only a single attribute on each side, denoted as $X \to A$, for the rest of this paper. Similar techniques as in [11] can be applied to compute FDs with multiple attributes efficiently.

2.3 Experiments

In order to evaluate the techniques we developed for generating pFDs over corpus with many data sources, we implemented the two algorithms, PerValue and PerTuple, for computing the probabilities of FDs over a single data source, and the MergeData and MergeFD approaches for combining either data or pFDs over multiple data sources. In this section, we compare the four possible ways to generate probabilistic FDs over three real datasets with many data sources.

Setup: First, we created the single-table mediated schema and the schema mappings for each of the three domains as would be automatically generated by [16]. Then we used the pFD generation algorithms to generate pFDs over the mediated schemas where both sides contain a single attribute. We implemented all the algorithms in Java along with Derby DBMS [1].

Each dataset is a set of HTML tables extracted from the Web. We considered tables from three domains: Business-Contact(People), Bibliography(Bib) and Course. For each domain, we identified tables in this domain by searching for tables that contain certain keywords in the attribute labels (see Table 2). The number of tables per domain varies from 45 to 619. The number of tuples in each source table varies from 6 to 1123, and on average is 53. Each mediated schema contains 11 to 14 attributes.

For each pFD generation algorithm, we took the set of FDs whose computed probability is above threshold $\tau = 0.8$ and compared it with a golden standard generated manually by the authors from their domain knowledge. We reported *precision, recall* and *f-measure*. Let $\mathbf{D}_G$ be the set of FDs in the golden standard and $\mathbf{D}_R$ be the set of automatically 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 $F = \frac{2PR}{P+R}$.

Results: Figure 2 shows the precision, recall and f-measure results of the pFDs generated by four different pFD generation algorithms – MergeFD-PerTuple, MergeFD-PerValue, MergeData-PerTuple and MergeData-PerValue.

As can be seen, in these experiments, the MergeFD-PerTuple algorithm generates the set of pFDs with the highest f-measure across all three domains. The results also show that the PerTuple algorithms usually outperform the PerValue algorithms. This is because by taking into account the frequency of each value of X, the per-tuple model is more suitable for data sets where the low quality of data is mainly caused by dirty values, which is the case in our three data sets. In addition, the MergeFD approach performs better than MergeData approach. This is because merging data from heterogenous sources introduces more noise and inconsistencies.

Table 1 shows for each domain, the number of generated functional dependencies that have probability above 0.8, and some examples among those generated pFDs. As can be seen, the number of pFDs automatically generated by the MergeFD-PerTuple algorithm is between 34 and 42, which is hard to generate all manually. Moreover, the sample pFDs show that useful pFDs are generated which capture the quality and characteristics of the data.

Lastly, we also ran experiments to test the sensitivity of the setting of the threshold τ. The results not shown here, indicate that the f-measure of the generated pFDs is rather stable when varying the threshold between 0.5 and 0.95 for these three domains.

3. APPLICATIONS OF PROBABILISTIC FDS

In this section we use pFDs to solve two problems that arise in pay-as-you-go systems. The first problem is data cleaning, where dirty data sources are discovered by validating pFDs. The second problem is improving the mediated schema, by normalizing into multiple schemas, each representing an entity or a relationship in the domain.

3.1 Data Cleaning

It is well known that FDs can be used as quality constraints in databases. In a pay-as-you-go data integration system, pFDs generated from the data sources can help pinpoint low quality data sources.

Given a set of high probability pFDs $\bar{F}$ over the mediated schema, a data source R and a mapping from the source to the mediated schema, we report on the data sources that violate one or more pFDs in $\bar{F}$. The violation of dependencies can be caused by one of the following types of dirtiness in data: (1) dummy or default values (e.g., the value 'Email' throughout email column) ; (2) entity ambiguities (e.g., two different values referring to the same entity), and (3) nested

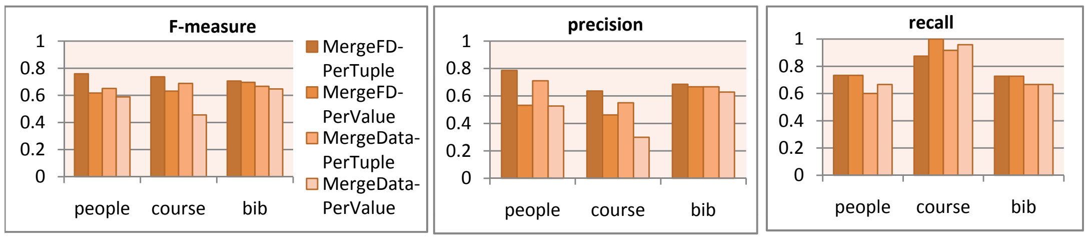

Figure 2: F-measure, Precision and Recall of different pFD generation algorithms.

Domain	Num pFDs generated	Example generated pFDs
people	42	name→ organization, organization→ address, email→ name, fax→ address
bib	34	journal→ issn, issn→ eissn, title→ journal, journal→ subject, title→ authors
course	35	class→ course, class→ days, course→ units, title→ course, instructor→ institution

Table 1: Number of generated pFDs (MergeFD-PerTuple) and Example generated pFDs.

columns (e.g., course section and title are mixed together in one column). In addition, violations can also be caused by incorrect schema mappings.

To generate the set of high probability pFDs $\bar{F}$, we take the pFDs generated from the MergeFD-PerTuple algorithm and threshold the probability at $\tau = 0.8$[2]. For each pFD $X \rightarrow A$ in $\bar{F}$, exists a set of data sources $\bar{S}$ where for every data source $R \in \bar{S}$, every attribute in X and A distinctly maps to an attribute in R. We compute the probability p of $X \rightarrow A$ over R; if $p < \tau$ then we report the data source R as it violates the pFD $X \rightarrow A$ in $\bar{F}$.

Results:

The results are promising for data cleaning using pFDs. Table 3 shows the number of data sources violating the pFDs we generated using the MergeFD-PerTuple algorithm over three domains. We looked at all the violating data sources and report the number of dirty ones among those, including the breakdown among the four types of dirtiness, either in data or in schema mapping. A data source R is considered dirty if it (1) contains at least one column with only dummy values; (2) contains at least one column with half or more values that has entity ambiguity; (3) contains at least one column with values from more than one attributes; and (4) contains at least one column which maps to a wrong attribute in the mediated schema.

For the people, course and bib domains, we discovered 5, 80 and 7 dirty sources respectively. Some of the clean data sources are mistakenly reported because the violated pFDs do not generally hold, for example 'class→ instructor' in the course domain and 'authors→ title' in the bib domain. The fraction of dirty sources in the reported data sources (i.e. precision) is 100%, 38% and 43% respectively.

In addition, we randomly sampled 45–50 data sources each from the three data sets, and checked if they are dirty according to the above criteria. The fraction of dirty data sources in the sample, is used to estimate the number of dirty sources over each entire data set. For example, in course domain, 11 sources are found dirty in 50-source sample, thus, we estimate that the whole data set have 66 dirty sources among 545 sources. As we can see from Table 3, the estimated number of dirty sources are 5, 66 and 0 respectively. We compute the percentage of dirty sources discovered using pFDs (i.e. recall) as the number of dirty sources found violating pFDs over the estimated numbers of dirty data sources, which is 100% (5/5), 47% (31/66) and 100% (3/0) respectively.

3.2 Schema Normalization

In a pay-as-you-go integration system, an automatically generated mediated schema can be large and hard to digest. The resulting schema often is too big to be presented as a single schema [16]. Another use of pFDs is to analyze the mediated schema and improve it to represent the objects and relationships over data.

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 for two reasons. First, the set of generated pFDs has imperfect precision and recall as shown in results in Section 2.3. Second, our goal of normalization is to identify objects and relationships. If not done carefully, BCNF normalization can separate attributes of one entity into several relations.

This section discusses how we solve these problems and achieve the goal of semantic normalization. Throughout this section, we use the following example to illustrate our approach.

Example 3.1. *Consider the following mediated schema that is automatically generated from a set of source schemas.*

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

Figure 3 shows pFDs over this mediated schema. If we consider the pFDs with a probability of no less than .9 as deterministic and apply BCNF normalization, we can obtain several valid normalization results, one of which is:

Business-contact(<u>*name*</u>, email, organization)
Org(<u>*organization*</u>, address, zip, country, title)
Addr(<u>*address*</u>, city)
City(<u>*city*</u>, 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,*

[2]As described in Section2.3, the setting of the threshold is rather stable between 0.5 and 0.95.

Domain	# Sources Found (Violating pFDs)	Dirty Sources In Found Sources # (Breakdown)	Frac. Dirty Sources In Found Sources (Precision)	# Sources Sampled /Total	Dirty Sources In Sampled Sources Est. # (Breakdown)	Percentage Discovered (Recall)
People	5	5 (5, 0, 0, 0)	100%	45/45	5 (5, 0, 0, 0)	100%
Course	80	31 (0, 15, 13, 3)	38%	50/545	66 (0, 22, 33, 11)	47%
Bib	7	3 (0, 3, 0, 0)	43%	50/618	0 (0, 0, 0, 0)	100%

Table 3: Results for data cleaning application using pFDs with type-of-dirtiness breakdown.

Domain	Mediated Schema	Normalized Schemas
People	(organization, name, country, work phone, zip, city address, fax, title, email, state)	Organization(<u>organization</u>, country, zip, city, address, fax) People(<u>name</u>, work phone, title, organization, email, state)
Course	(catalog number, class number, section, units, location, subject, fee, time, title, instructor days, institution, catalog number, term)	Course(<u>catalog number</u>, location, subject, units, class number, fee, institution) Class(<u>class number</u>, section, time, title, instructor, days, catalog number, term)
Bib	(journal title, title, id, subject, source, eissn authors, volumne, years, issue, issn)	Journal(<u>journal title</u>, issn, subject, source, eissn) Article(<u>title</u>, id, authors, volumne, years, issue)

Table 4: Results for mediated schema normalization application using pFDs.

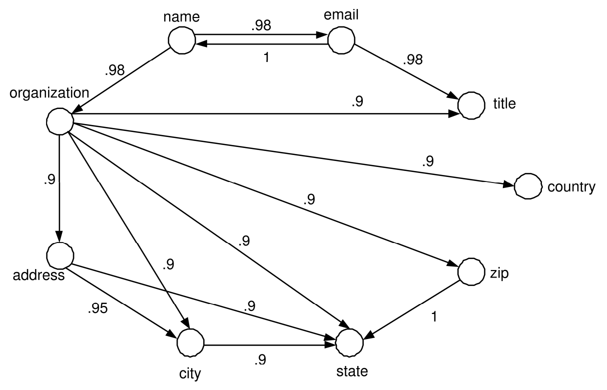

Figure 3: Example 3.1: pFDs for a given mediated schema of the people domain. To avoid cluttering the graph, we omit pFDs whose probabilities are below .9; we also omit most of the pFDs from name or email to other attributes, whose probabilities range from .95 to 1.

address, city, state, zip, country *are all attributes that describe addresses; splitting them into several relations is not desired.* □

Selecting pFDs As shown in the above example, 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 set of generated pFDs may have missing or wrong pFDs as shown in results in Section 2.3. We thus do a further pruning according to the following heuristic.

HEURISTIC 3.2. *Each non-key attribute must belong to one and only one entity or relationship.* □

Following this heuristic, 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. Let η be the threshold for pFDs that are considered likely to be true and δ be the maximum difference between two selected pFDs with A on the right-hand side. We prune a pFD $X \rightarrow^p A$ if one of the following three conditions holds.

- $p < \eta$;
- Let $p_{max} = \max_Y(Pr(Y \rightarrow A)), p_{max} - p > \delta$;
- $\exists Y$ such that $Y \rightarrow^{p'} A, p' > p_{max} - p$, and $\exists\{Z_1, \ldots, Z_l\}, l \geqslant 1$, such that each of $X \rightarrow Z_1, \ldots, Z_l \rightarrow Y$ has a probability above η.

Consider Example 3.1. If we set $\eta = 0.9$ and $\delta = 0.05$, 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 each attribute will remain in the same relation as the attribute that represents the key of its object [15].

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 $K \rightarrow K'$.

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.

Continuing Example 3.1: After we apply pruning, a dependency-preserving 3NF normalization obtains the following relations.

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

```
NORMALIZATION ($Med$)
1   $rels \leftarrow \{Med\}$;
2   // Generate pFDs with single attribute on each side
3   $pFDs \leftarrow$ GENERATEPFDS($Med$);
4   // Find key of the relation
5   $keys \leftarrow$ FINDKEYS($Med, pFDs$);
6   // Apply Heuristic3.2 to prune pFDs
7   $pFDs \leftarrow$ PRUNEPFDS($pFDs$);
8   // 3NF Normalization
9   for each $D : \bar{X} \rightarrow A \in pFDs$ do
10      if $\bar{X} \notin keys$ then
11          // Normalize rels according to $D$;
12          $rels \leftarrow$ 3NFNORM($rels, D$);
13  endif endfor
14  // Avoid oversplitting
15  $rels \leftarrow$ MERGE($rels, k$);
16  return $rels$;
```

Figure 4: Algorithm NORMALIZATION: normalize a mediated schema using generated pFDs.

Figure 4 gives the complete algorithm for mediated schema normalization.

Results:

Table 4 shows results of NORMALIZATION over the mediated schema of each domain. The mediated schema of each domain is a single-table schema created as would be produced by an automated algorithm such as [16]. As shown in the table, the mediated schema of people, bib and course domain have 11, 11 and 14 attributes respectively. After applying the normalization algorithm, with parameters $\eta = 0.8$, $\delta = 0.05$ and $k = 2$, each mediated schema is normalized into two meaningful entities in the domain. In the people domain, the two entities are person and organization; in the bib domain, the two entities are journal and article(paper); and in the course domain, the two entities are course and class offerings. As we can see, although not perfect (for example, 'fax' attribute should be in the People instead of the Organization table), the schema normalization algorithm can do a reasonable job in normalizing a wide mediated schema to multiple schemas, each representing a real-world entity.

4. RELATED WORK

The idea of deriving functional dependencies from data has been proposed in TANE [11] and CORDS [12], and they call the functional dependencies they generate *approximate FDs* or *soft FDs*. The concept and the algorithms are very close to our probabilistic functional dependencies and the PERTUPLE model over single data source.

Recently, there has been work on discovering *conditional functional dependencies* from data [10, 7]. However, such dependencies most likely to exist in large, complex databases, but very rarely in the corpus with small data sources, which we are focusing on.

Some previous work [7, 8, 10] focused on using functional dependencies or conditional functional dependencies generated over a single large database to discover dirty data items. None of the past literatures explores dataset with many small data tables and apply functional dependencies over the domain to discover dirty data sources.

A large body of work [5, 3, 13, 14, 17] on creation of mediated schemas focused on semi-automatically merging schemas to create a mediated schema, where ambiguity needs to be resolved by users. Das Sarma et al. [16] proposed automatically generating a probabilistic mediated schema from source schemas and then consolidating them to provide one single-table mediated schema; however, the mediated schema is not normalized. Moreover, our normalization aims at normalizing the mediated schema into relations describing objects and relationships.

5. CONCLUSION

In this paper we show how to extend TANE algorithm over single table to generate probabilistic functional dependencies from data corpus with hundreds of small, dirty and incomplete data tables in one domain. We show how to use generated dependencies to measure and improve the quality of schema and data, by dirty data source discovery and mediated schema normalization. The experiments over real data sets show promising results. Future work includes how to automatically adjust the parameters in the algorithms.

6. REFERENCES

[1] Apache derby.

[2] E. Arichtein, L. Gravano, V. Sokolovna, and A.Voskoboynik. Snowball: A prototype system for extracting relations from large text collections. In *SIGMOD*, 2001.

[3] 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.

[4] M. J. Cafarella, A. Halevy, D. Zhe Wang, Y. Zhang, and E. Wu. Webtables: Exploring the power of tables on the web. In *Proc. of VLDB*, 2008.

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

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in knowitall (preliminary results). In *WWW*, 2004.

[7] W. Fan, F. Geerts, M. Xiong, and L. V.S. Lakshmanan. Discovering conditional functional dependencies. In *Proc. of ICDE*, 2009.

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

[9] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*

[10] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. In *Proc. of VLDB*, 2008.

[11] 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.

[12] 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.

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

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

[15] R. Ramakrishnan and J. Gehrke. Database management systems. McGraw-Hill, 2003.

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

[17] 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.