ERACER: A Database Approach for Statistical Inference and Data Cleaning

Chris Mayfield
Purdue University
West Lafayette, Indiana, USA
cmayfiel@cs.purdue.edu

Jennifer Neville
Purdue University
West Lafayette, Indiana, USA
neville@cs.purdue.edu

Sunil Prabhakar
Purdue University
West Lafayette, Indiana, USA
sunil@cs.purdue.edu

ABSTRACT

Real-world databases often contain syntactic and semantic errors, in spite of integrity constraints and other safety measures incorporated into modern DBMSs. We present ERACER, an iterative statistical framework for inferring missing information and correcting such errors automatically. Our approach is based on belief propagation and relational dependency networks, and includes an efficient approximate inference algorithm that is easily implemented in standard DBMSs using SQL and user defined functions. The system performs the inference and cleansing tasks in an integrated manner, using shrinkage techniques to infer correct values accurately even in the presence of dirty data. We evaluate the proposed methods empirically on multiple synthetic and real data sets. The results show that our framework achieves accuracy comparable to a baseline statistical method using Bayesian networks with exact inference. However, our framework has wider applicability than the Bayesian network baseline, due to its ability to reason with complex, cyclic relational dependencies.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Statistical Databases; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Experimentation, Performance

Keywords

Relational dependency network, approximate inference, discrete convolution, linear regression, outlier detection

1. INTRODUCTION

Although the database community has produced a large amount of research on integrity constraints and other safety measures to ensure the quality of information stored in relational database management systems (DBMSs), in practice databases often contain a significant amount of non-trivial errors. These errors, both syntactic and semantic, are generally subtle mistakes which are difficult or even impossible to express using the general types of constraints available in modern DBMSs. In addition, quality control on data input is decreasing as collaborative efforts increase, with the Internet facilitating widespread data exchange, collection, and integration activities. Clearly, there is an increasing need for new approaches to data cleaning for the purpose of maintaining quality in relational databases.

Data cleaning (or cleansing, scrubbing, etc.) is the process of identifying and repairing incorrect or corrupt records in a database. The goal is not only to bring the database into a consistent state (i.e., with respect to domain or integrity constraints), but also to ensure an accurate and complete representation of the real-world constructs to which the data refer. Two surveys of common techniques and general challenges in this research area include [16] and [22]. Removing impurities from data is traditionally an engineering problem, where ad-hoc tools made up of low-level rules and manually-tuned algorithms are designed for specific tasks. However, recent work has shown the the effectiveness of applying techniques from machine learning and data mining for the purpose of data cleaning [7]. In particular, statistical methods make it possible to automate the cleansing process for a variety of domains.

For this work we develop statistical methods for cleaning relational databases with the following characteristics:

- **Incomplete and erroneous:** There are both (1) missing values to be filled in, and (2) corrupted values to be identified. This goes beyond traditional statistical methods which make assumptions about the reliability of the non-missing values.
- **Correlated attributes:** The values of different attributes are correlated, both *within* and *across* tuples (involving perhaps multiple relations). Much of the prior work in data cleaning concentrates on values within a single tuple or relation.
- **High-level dependencies:** The attributes with large domains (i.e., many possible values), exhibit higher-level dependencies among sets of similar values (for categorical variables) or a numerical functional dependency (for continuous variables).

As an example of this type of domain, consider the task of inferring missing birth and death years of individuals in genealogical databases. The individuals are related through parent-child relationships and the birth and death years of an individual are correlated due to life expectancies. In addition, the birth years of parents and children are correlated due to expected parenting ages. Furthermore, since life expectancies and parenthood ages are likely to be similar over time, the dependencies do not need to be modeled for specific birth years. Instead they can be modeled as a higher-level functional dependency such as *birth year = parent's birth year +* ϵ. A statistical method can learn these dependencies from the available data and then use them to infer missing values automatically.

As another example, consider the task of inferring missing data in sensor networks. There are often relationships among the different types of measurements in the same sensor (e.g., temperature and humidity), as well as relationships among the measurements of neighboring sensors due to spatial locality. Again, a statistical method could learn these dependencies from observed data and then use them to infer missing values (e.g., due to battery loss) and/or clean corrupt values (e.g., due to sensor malfunction). Such a method can also be used for anomaly detection and intrusion detection systems.

This paper introduces ERACER, a database-centric statistical framework for *integrated* data cleaning and imputation. The core techniques are based on belief propagation [20] and relational dependency networks [18]. We show how to implement the inference and cleaning processes efficiently *at the database level.* This eliminates the expensive practice of migrating the data to and from statistical software such as R or Matlab, which is particularly useful when the amount of data—or limited processing time and resources—prevents a more extensive analysis. In contrast to prior work that cleans tuples in isolation, our approach exploits the graphical structure of the data to *propagate* inferences throughout the database. As a result the imputation and cleaning tasks are synergetic: additional information in the database helps identify errors more accurately, and corrected data values improve the quality of inference for the missing values.

2. RUNNING EXAMPLES

We consider tasks from two example domains throughout this paper: (1) inferring missing birth and death years of individuals in genealogical data, and (2) inferring missing temperature and humidity measurements in sensor data. Both of these domains have correlated attributes and graphical structures, which we exploit to make inferences and corrections for the underlying data values.

2.1 Genealogy Databases

Our first example data set is a sample of five million individuals from the Pedigree Resource File[1] (PRF), a lineage-linked database of records submitted to FamilySearch.org. A simplified version of the schema is:

```
person (ind_id, birth, death)
relative (ind_id, rel_id, role)
```

We assume for now that each birth and death year is a single value, and that either or both attributes may be unknown (i.e., NULL). We also assume the majority of the known values are correct, but expect there to be many non-trivial errors which cannot be detected by standard integrity constraints such as "birth year is less than death year." Our approach supports any pedigree database with a table of parent-child relationships, which in this example are designated by the `role` attribute. For simplicity we assume that all relationships are both present and correct; detecting and cleaning structural errors is an orthogonal problem.

The PRF data set has several properties which make it particularly interesting for testing our framework. For one, it is real data compiled from thousands of independent submissions over the past few decades. As a result it imposes many of the standard challenges of data integration, including erroneous attribute values and structural inconsistencies. Overall, about 68% of individuals in this sample have a birth year, 33% have a death year, and only 27% have both. The unsurprising trend is that older ancestors are more likely to have missing information. The data also contains a number of duplicate individuals, as it was originally collected for record linkage research.

2.2 Sensor Networks

We also used the Intel Lab Data[2] (ILD) which includes a number of measurements taken from 54 sensors once every 31 seconds for about 38 days. This amounts to over two million rows with the following schema:

```
measure (epoch, mote_id, temp, humid, light, volt)
neighbor (id1, id2, distance)
```

An individual measurement is identified by the mote_id and epoch, or the number of seconds since the deployment[3] of the sensors. Less than one percent of the original tuples contain NULL values. However, about 18% of the measurements are obviously incorrect (e.g., temperatures of over 100°C, and relative humidity values of -4%). In our experiments we set those values to NULL, thus treating them like missing data.

Sensor networks are becoming increasingly popular in manufacturing and other monitoring applications, to ensure the safety or stability of a particular environment. In this work we consider the battery-powered, wireless sensors motivated by TinyDB and others [15]. Although this data set is a network of stationary sensors, our framework is also applicable to other configurations including mobile ad-hoc networks.

In contrast to the PRF data, the ILD database does not have explicit relationships between individual measurements. Instead, we construct implicit links between sensor readings using domain knowledge like spatial and temporal locality (i.e., within m meters and $\pm s$ seconds). As a result, the correlations between these linked values are much more complex than the simple parent-child relationships described before, as they span multiple attributes with varying data types.

3. FRAMEWORK

A natural approach for inferring missing values in relational databases is to capture attribute dependencies with graphical models. *Graphical models* leverage techniques from graph theory and probability theory to represent and reason with complex joint distributions of many random variables

[1]We wish to thank Randy Wilson for providing us with this data, and for several insightful discussions about genealogical data management.

[2]See http://db.csail.mit.edu/labdata/labdata.html.

[3]We found the original ILD epoch values to be inconsistent, and recomputed them from the record timestamps.

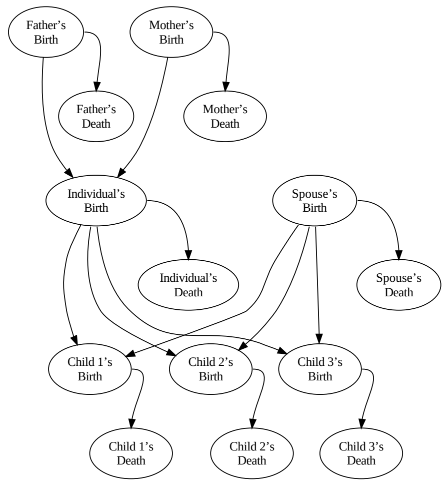

Figure 1: Bayesian network for the genealogy data.

compactly and efficiently. We refer the reader to Chapter 8 of [2] for an overview of graphical models and standard inference algorithms. In this section we assume the reader has a basic understanding of Bayesian networks.

Returning to the genealogy example, let the random variables $I.b$ and $I.d$ denote the birth and death years of an individual I in the database. The goal is to infer posterior distributions for each $I.b$ and $I.d$, based on the observed birth and death values from the individual's relatives R_I. We use parent-child relationships inherent in the data as a *template* for our graphical model. This approach is similar to learning a *probabilistic relational model* (PRM), a relational extension to Bayesian networks [9].

Figure 1 illustrates an example Bayesian network which results from *unrolling* the model template over a small set of individuals in the database. In this example we have an individual with two parents, one spouse, and three children. In terms of the rolled-out network, each child's birth year is influenced by the birth years of both parents. Likewise, each person's death year depends only on his or her own birth year. The father, mother, and spouse nodes have no parents, so they depend only on a prior distribution. In reality the size of the model could grow to include the entire database, as additional ancestors and descendants are added to the network. In addition to the model structure, the quantitative dependencies are specified with a *conditional probability distribution* (CPD) for each node in the network, i.e., conditioned on its parents. Our model template consists of two CPDs: $P(I.d|I.b)$ and $P(I.b|M.b, F.b)$.

A Bayesian network formulation like this is attractive due to its principled mathematical foundation. However, it has a number of limitations that prevent its applicability for large-scale data cleaning tasks:

- Inefficiency and potential inaccuracy due to a comparatively large number of parameters in the CPDs.
- Inflexibility for heterogeneous relational data sets, due to fixed CPD structure and the acyclicity constraint.
- Inability to identify and correct invalid data values, due to a lack of appropriate cleansing algorithms.

In the following sections, we propose a novel framework that uses locally-learned models and approximate inference

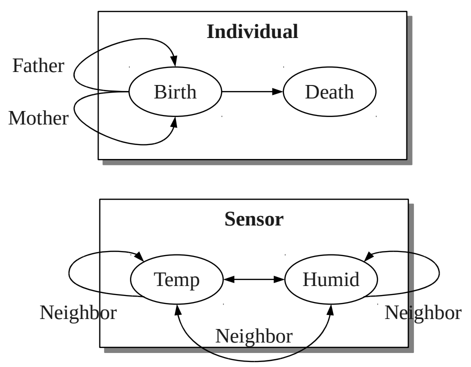

Figure 2: RDN model templates for the genealogy data (top) and the sesnor network data (bottom).

to offset these issues. We model heterogeneous relational dependencies using aggregation to relax the restrictions on the CPD structures, which approximates the full joint distribution and circumvents the acyclicity constraint. In addition, our algorithms are flexible enough to infer a posterior distribution for *every* variable, which we use to identify outliers and errors for data cleaning. Finally our method provides a natural implementation within database systems via SQL and user defined functions, resulting in significant improvements in runtime efficiency.

3.1 Relational Dependency Networks

Our framework models the database as a *relational dependency network* (RDN) [18]. RDNs extend dependency networks [11] to work with relational data in much the same way that relational Bayesian networks extend Bayesian networks and relational Markov networks extend Markov networks [10]. All of these models extend traditional graphical models to relational domains, removing the assumption of independent and identically distributed instances that underlies conventional machine learning techniques.

Figure 2 illustrates the RDN model templates for our example databases using plate notation. Continuing the genealogy example, we have a random variable for the individual birth and death year values—these correspond to the $I.b$ and $I.d$ nodes discussed previously. The edges represent statistical dependencies between the random variables. Edges within a plate represent dependencies among the random variables of a single tuple (e.g., a person's death year depends on the birth year). Edges that extend outside a plate represent dependencies among the random variables of *related* tuples (e.g., a person's birth year depends on the birth years of the mother and father).

The primary distinction between RDNs and other probabilistic relational models is that RDNs *approximate* the full joint distribution with a set of locally-learned CPDs. This approximation facilitates the representation of arbitrary relational (i.e., cross-tuple) dependencies needed for data cleaning. In particular, it allows cyclic dependencies (e.g., many-to-many relationships) that are difficult to model with relational Bayesian networks. RDNs also offer a relatively simple method for parameter estimation based on *pseudolikelihood*, which results in models that are easier to implement. If the local models are learned from a large data set, the overall approximation of the full joint distribution will be quite accurate due to the coordinating effects of the underlying data.

As in conventional Bayesian networks, each random variable is associated with a probability distribution conditioned on the other variables. Parents of a variable are either: (1) other attributes in the same tuple (e.g., $I.b$ and $I.d$), or (2) attributes in related tuples (e.g., $P.b$ where P is the parent of I). For the latter type of dependency, if the database relation is one-to-many, the parent consists of a set of attribute values (e.g., all the children's birth years). In this situation, the model uses *aggregation* functions to generalize across heterogeneous attributes sets (e.g., one person may have two children while another may have five). Aggregation functions are used either to map sets of values into single values, or to combine a set of probability distributions into a single distribution.

Note that this model does not attempt to capture all constraints present in the underlying database. Instead, for this work, we focus on a reasonable subset of dependencies which are likely to be most useful for inferring missing values. For example, a child's birth year is generally less than both of the parent's death years, but we do not include edges from $I.d$ back to $I.b$ in our graphical model template for two reasons. For one, representing too many constraints could lead to model overfitting and result in less accurate inferences. And second, additional dependencies will increase the complexity of the network and significantly hinder the runtime performance of our baseline used for evaluation.

3.2 Component Models

The graphical model structures specify the qualitative conditional dependencies in the data (i.e., which random variables depend on each other). In addition to the network structure, the quantitative dependencies are specified with a conditional probability distribution (CPD) for each node in the network, conditioned on its parents. For example, our genealogy model template consists of two CPDs: $P(I.d|I.b)$ and $P(I.b|A.b)$, where we now model the mother and father dependencies with a generic "ancestor" dependency—in contrast to the Bayesian network formulation that modeled each parent individually.

Since RDNs consist of a set of local CPDs learned independently, we may use a different *component CPD* for each domain. Previous work has focused on using either relational Bayesian classifiers or relational probability trees as the component models [18]. In this work, we will use convolution models to capture the functional dependencies in the genealogy data, and regression models to represent the correlations in the sensor network data.

3.2.1 Convolution Models

One method for constructing CPDs is simply to aggregate the known instances in the database (see experiments). There are two main problems with this approach: (1) the CPDs consist of a very large number of parameters (in our experiments, the CPD $P(I.d|I.b)$ contained over 75,000 rows), and (2) many reasonable pairs of birth and death years will be missing from the CPD if unobserved in the database.

To address these issues, we observe that the dependencies between parent and child birth values are likely to be similar across different years. The functional dependency can be based on the offset or change in values rather than the specific values of the random variables. To exploit this, we propose a convolution-based approach to modeling death age and parent age at birth, instead of using conventional, value-specific CPDs.

Consider a pair of attributes that are correlated through this kind of function (e.g., the birth and death year of an individual). Instead of modeling the explicit dependence of death year on birth year $P(I.d|I.b)$, we can model the distribution of the *difference* of the two variables as an individual's *death age*: $M_{DA} = P(I.d - I.b)$. Similarly, we can model the difference in child and parent birth years as an individual's *parent age*: $M_{PA} = P(I.b - P.b)$. These distributions correspond to the two edge types in Figure 2.

Since we construct these distributions using subtraction, we use convolution to "add" information and make inferences between two attributes. (See the "apply" step in Section 4.2 for more details about the inference procedure.) The notable property of this approach is that it allows the use of information across the entire database for estimation, instead of matching on exact combinations of birth and death years. This technique is known as *shrinkage* in statistics literature.

When using convolution models as the RDN components, we infer each parent's influence independently and aggregate the resulting predictions (i.e., probability estimates) during inference. This is one approach to reasoning across heterogeneous structures (e.g., individuals with varying numbers of parents and/or children) that is often used in statistical relational learning [10].

3.2.2 Regression Models

The sensor network domain also involves a number of functional dependencies. For example as temperature rises, it is likely that relative humidity decreases. In addition, sensor readings are likely to be correlated both across time and space, since neighboring sensors usually experience similar states in the environment.

Because the sensor variables are continuous values, it is natural to consider linear regression for the component models in this domain. Specifically, we model the following:

$$S.t \sim \beta_0^t + \beta_1^t S.h + \beta_2^t avg(N.t) + \beta_3^t avg(N.h)$$
$$S.h \sim \beta_0^h + \beta_1^h S.t + \beta_2^h avg(N.t) + \beta_3^h avg(N.h)$$

where $avg(N.t)$ and $avg(N.h)$ refer to the average temperature and humidity values at neighboring sensors.

In this case, to deal with heterogeneous relationships (e.g., sensors with different numbers of neighbors), we first aggregate the values of the neighboring sensors rather than applying multiple models and aggregating the predicted probabilities. This is another approach typically used in statistical relational learning to deal with heterogeneous structure [10]. In the genealogy data, model aggregation is possible due to similarity of the individual model predictions (i.e., all parent models use birth year). However, in the sensor domain, individual models making predictions based on different types of information (e.g., temperature, humidity) are unlikely to have similar accuracy. In this case, a model that incorporates all the different sources of information into a single prediction is usually more accurate.

3.3 Learning CPDs

Our goal is to learn the parameters of the CPDs automatically based on the observed (i.e., non-NULL) instances in the database. Such a data-driven approach is what makes our method applicable to other domains with similarly struc-

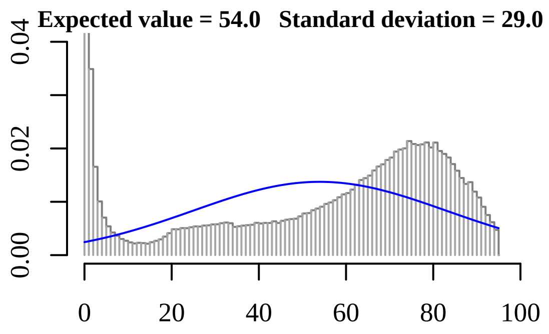

Figure 3: Convolution model for "Death Ages."

tured dependencies. In the case of genealogy, we may be able to obtain accurate conditional distributions (e.g., life expectancy models) from social science domain experts. However, in other domains it is likely that such domain experts and/or background knowledge do not exist. For this reason, we learn the parameters of the component models directly from the data.

3.3.1 Convolution Learning

To estimate the convolution models, we first compute the distribution of the difference of each pair of known instances. We have implemented a user defined aggregate named `hist` that returns a (normalized) histogram of the values it scans:

```
SELECT hist(death - birth)
FROM person;
```

In this example, the resulting model represents the distribution of death ages for all persons with known birth and death years. Figure 3 illustrates one of the histograms for the PRF data. The spike on the left side suggests a high child mortality rate for this data set. We have also computed the mean and standard deviation of this histogram, and plotted the normal distribution curve using these parameters. Clearly, a normal distribution does not provide a good approximation for this model.

We likewise compute the distribution of parenthood ages, i.e., individual birth minus parent birth, by self-joining the person table via its relatives:

```
SELECT hist(i.birth - p.birth)
FROM person AS i
  INNER JOIN relative AS r
    ON i.ind_id = r.ind_id AND r.role IN ('M', 'F')
  INNER JOIN person AS p
    ON r.rel_id = p.ind_id;
```

These queries require very little domain knowledge, namely that birth years and death years for an individual are correlated, as are the birth years of parents and children. In general, convolution models capture the dependencies between any pair of relational attributes that lie within an expected range of each other.

3.3.2 Regression Learning

We estimate the regression model coefficients offline[4] using linear model fitting (lm) in R, an open source statistical software package. For efficiency, we query a random sample of training data $\langle S.t, S.h, avg(N.t), avg(N.h)\rangle$ by joining each sensor with its neighbors and aggregating the neighboring values. After learning the appropriate linear regression models in R, we record the estimated coefficients $\beta_{\mathbf{T}} = [\beta_0^t, \beta_1^t, \beta_2^t, \beta_3^t]$ and $\beta_{\mathbf{H}} = [\beta_0^h, \beta_1^h, \beta_2^h, \beta_3^h]$ in the database for subsequent use. Note that coefficients 1–3 correspond to the three edge types in Figure 2, and β_0 is the model intercept.

3.4 Inference and Data Cleaning

Using the component models learned from the known instances, we now present an iterative, database-centric, approximate inference procedure to estimate posteriors for each missing value and identify/clean potentially corrupt values. The algorithm uses a message-passing approach to infer the posterior distribution locally for each node in the entire database, rather than explicitly constructing specific Markov blankets over which to do (potentially exponential) joint inference (see experiments for more details).

The name of our resulting framework, ERACER[5], is an acronym for the major phases of our framework:

1. **Extract**: use domain knowledge (i.e., model templates) to construct the graphical model structure.
2. **RDNs**: learn the parameters of each applicable RDN component model.
3. **Apply**: for each variable modified in the previous round of inference, apply the relevant models to construct (or update) the output distribution "message" to send to each related variable (i.e., neighbors).
4. **Combine**: aggregate the resulting predictions (or input values) to deal with heterogeneity.
5. **Evaluate**: for each variable, accept or reject the inferred posterior distribution after comparing it with the previous version.
6. **Repeat**: until probability distributions converge, i.e., when the resulting update count is close to zero.

Note that the first two steps (ER) describe the initial process of model construction and learning, whereas the remaining four (ACER) define the iterative inference and cleaning process in the database. We will explain the algorithms for each phase in Section 4.2.

4. IMPLEMENTATION

4.1 SQL Interface

Our framework requires the user to define three relations as input to the approximate inference and cleaning process. The first is named `model` and contains the learned parameters for the desired inference functions (i.e., histograms for convolution and/or coefficients for regression). The format of this table and the corresponding predictor functions are both user-defined, and are automatically referenced by the framework as needed.

The other two relations, `node` and `link`, correspond to the rolled-out relational dependency network structure (i.e., vertices and edges of the graphical model). For example, using the ILD database we populate the node table as follows:

[4]We are currently exploring ways to incorporate this one-time learning process into the database, such as using PL/R.

[5]The name is also a deliberate misspelling of *eraser*, which connotes its use for correcting erroneous data values.

```
INSERT INTO node
  SELECT make_nid(epoch, mote_id),
    new_basis(temp), new_basis(humid),
    new_basis(light), new_basis(volt)
  FROM measure;
```

The `make_nid` function constructs a unique integer identifier to serve as a simple key, given a set of (composite key) attributes. We currently achieve this with bitwise manipulations, although a more general mapping could be used if necessary. The `new_basis` function initializes an internal data structure for each random variable, which includes the following state:

`initial`	original value, if any (e.g., humid)
`pdf`	current prediction (or distribution)
`suspect`	data cleaning flag (true = outlier)
`round`	when pdf/suspect was last updated

Our implementation currently represents predictions as Gaussians (by storing the mean and standard deviation) or histograms (by storing the bin ranges). The initial prediction for a known value is either the value itself (with a probability of one attached to it) or a user-supplied Gaussian.

We next construct the link table based on the spatial and temporal locality of individual sensor readings:

```
INSERT INTO link
  SELECT make_nid(a.epoch, a.mote_id),
         make_nid(b.epoch, b.mote_id)
  FROM neighbor AS c
    INNER JOIN measure AS a ON c.id1 = a.mote_id
    INNER JOIN measure AS b ON c.id2 = b.mote_id
  WHERE a.epoch - 30 <= b.epoch
    AND a.epoch + 30 >= b.epoch;
```

Presently, the queries to construct these tables are the only domain knowledge required for our framework. We are currently looking into ways of automating this setup process by applying *structure learning* techniques. In the case of the PRF genealogy data, the node and link tables are essentially a copy of the original person and relative tables.

Once the model, node, and link relations are in place, the user executes the following aggregation[6] query to perform a single iteration of the approximate inference process:

```
SELECT erace(i, n)
FROM node AS i
  LEFT JOIN link AS l ON i.nid = l.id1
  LEFT JOIN node AS n ON l.id2 = n.nid
GROUP BY i;
```

Note the generality of the above query: we simply aggregate the neighboring nodes (if any) of each individual node. Depending on the nature of the underlying data and availability of indexes, a typical query engine will efficiently perform this aggregation using sorting and/or hashing. In addition, all attributes of the node tuple are inferred and cleansed during a single function call. The output of the `erace` function is the updated version of each node i, with inferred values filled in and/or corrected values identified. The user may then create a new table to store these results or update the previous node table in place.

[6]Note that we group by an entire tuple. We implement this in PostgreSQL by defining an operator class over the `node` type, which in turn sorts or hashes the nodes by their ids.

4.2 ERACER Algorithms

We now present the approximate inference and data cleaning algorithms in the context of our genealogy example. The following techniques also work for the sensor network domain, except that we use linear regression instead of convolution and consequently reverse the order of the combine and apply steps (see Section 3.1). The inference algorithm will operate in a manner similar to belief propagation [20], iterating over the apply, combine, evaluate, and repeat steps, and then propagating inferences about an individual to its parents and children in the next round.

As with other user-defined aggregates, the `erace` function consists of two phases: *transition*, in which each input tuple is processed, and *finalize*, in which the actual result is computed. In terms of our framework introduced in Section 3.4, the transition phase executes the apply and combine steps, and the finalize phase handles the evaluate step.

4.2.1 Apply

Recall in our example that we have learned models which correspond to the death age and parenthood age of individuals (M_{DA}, M_{PA}). Since we constructed these models using subtraction, we use addition to make inferences between related attributes. For example, we can infer a missing death year from a known birth year by shifting the x-axis by the birth year. In other words: $I.d = I.b + M_{DA}$ where M_{DA} is the death age model. Similarly, we can infer a missing birth year from a known death year by negating the histogram (i.e., mirroring it across the y-axis) and then shifting the result by the known death year: $I.b = I.d + [M_{DA} * -1]$.

Our framework will propagate these inferred posterior distributions as new evidence, which makes it possible to infer the birth years of an individual's children. In this case, since the evidence is now a distribution over possible values, we cannot simply add the evidence to the model by shifting the x-axis. Instead we use *discrete convolution* to add the random variables:

$$C.b = I.b * M_{PA} = \sum_{b' \in I.b} P_{I.b}(b') M_{PA}(b - b')$$

In other words, we compute the probability distribution of the sum of the random variables $I.b$ and M_{PA}.

In the apply step, we use convolution to infer a separate posterior distribution for each individual's birth and death years from each of his or her relatives (i.e., parents and children) that were updated in the previous round. We make a conditional independence assumption here for the sake of efficiency and flexibility. In other words, for each value $I.b$ we will infer a posterior from each parent and child birth year independently and then aggregate the predictions in the combine step below. We show empirically that the resulting accuracy of this approximation is quite good in practice.

4.2.2 Combine

The next task is to aggregate the individual predictions sent from each relative during the apply step. We interpret each bin of the histograms $P_1..P_j$ as a weight for their corresponding values, and simply "tally the votes" using:

$$P_m = \frac{1}{Z} \sum_j \sum_i P_j(i)$$

where Z is a standard normalizing function to rescale the probability distribution to sum to 1. Note that we currently

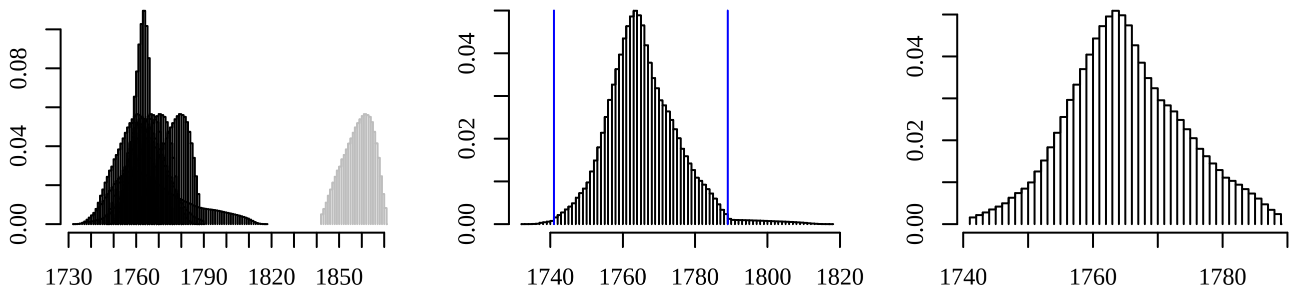

Figure 4: Visual example of combining inferences (left), computing the confidence interval (middle), and normalizing the resulting distribution (right).

interpret each piece of evidence P_j with equal value. As future work we plan to extend this method to compute a weighted sum, which can be used to factor in the reliability of each information source.

Since the predictions may have been based on erroneous information, we select a representative range from the combined distribution based on an *election* process (see Figure 4 for an illustration). First, if the posterior distribution contains multiple discontinuous regions (i.e., separated by zero probabilities), we choose the sub-region of the distribution with the highest probability, drop the other regions and renormalize the distribution. In case of ties (i.e., multiple disjoint regions with equal mass), we select the region with lowest variance. Once we have chosen a single continuous region, we compute a 99% confidence interval and renormalize the resulting distribution. This process saves space and prevents the inference procedure from carrying forward trivial amounts of probability mass due to error propagation.

Returning to our running example, the left plot in Figure 4 shows multiple inferences for a missing birth year, using evidence from the individual's parents and children. We assume the gray distribution on the right resulted from an erroneous piece of evidence, and will discard it since it is a disjoint region of relatively low probability mass. After computing the 99% interval (center), the system returns the combined distribution (right). Finally, we store the contributing *sub-pdfs* (i.e., the individual distributions which passed the election) for use in feedback control during subsequent rounds (see Section 4.2.4).

4.2.3 Evaluate

In the final step of each iteration, we compare the combined distribution with the previous version. Figure 5 summarizes our algorithm for evaluating updated inferences at the end of each round. Recall that in addition to filling in missing values, the other objective of our framework is to identify potential errors in the original data. For this reason, the system computes expected distributions for *all* data items, not just the missing values. We then check to see how close the inferred distributions are to the initial values (or distributions) of the non-missing data.

Lines 1–12 analyze the updated distribution to validate existing data. In the first case (Lines 3–6), when the current pdf is not marked as a suspect, we call the `Outlier` function to see whether the observed value lies within the expected pdf. Figure 6 gives our current method for outlier detection. Specifically, an original distribution is questionable if the majority of it is not contained within the majority of the inferred distribution resulting from the combine step.

```
**Evaluate**: reviews changes after propagating evidence
    Input: initial, current, and updated versions of pdf
    Output: resulting pdf (at the end of the round)
1.  missing := initial is null;
2.  if missing = false
3.    if suspect = false
4.      if Outlier(initial, updated) = true
5.        suspect := true;
6.        return updated;
7.    else
8.      if Outlier(initial, updated) = false
9.        suspect := false;
10.       return initial;
11.     else
12.       missing := true;
13. if missing = true
14.   if Diverge(current, updated) = true
15.     return updated
16. return current
```

Figure 5: Algorithm for evaluating updates.

```
**Outlier**: is the original range outside of the inferred?
    Input: initial pdf P and inferred pdf V
    Output: true if P lies outside of V, false otherwise
1.  $[a, b] := ConfInt(P)$
2.  $[c, d] := ConfInt(V)$
3.  return $a < c$ or $b > d$
```

Figure 6: Algorithm for outlier detection.

If the distribution is determined to be an outlier, we set its `suspect` flag and return the expected pdf as the updated value. Consider for example an individual with the birth year 840 and the death year 1914. Clearly, one (or both) of these values must be incorrect. After propagating known evidence from several relatives, the birth year is estimated to be between 1819 and 1871. Since 840 lies outside this interval, we mark it as an error and treat it as missing in subsequent iterations.

In practice, we cannot tell which of the two conflicting pieces of information is incorrect in a single round. Instead, we aggressively identify errors and rely on Lines 8–10 to correct any false assumptions. Continuing our previous example, 1914 is also marked as an error because it lies outside of the distribution inferred from the birth year 840. However at the end of the second iteration, `Evaluate` runs with $initial = 1914$, $current \approx 908$, and $updated \approx 1908$. Since the initial value no longer lies outside of the inferred range, we clear the suspect flag and restore the original value.

Line 12 essentially means "treat suspicious base data as if it were missing," allowing inferences to be refined during subsequent rounds. Lines 13–15 address the case of inferring missing data. For efficiency, we calculate the Jensen-Shannon divergence [13] to measure the difference between the inferred distribution and its previous version (if any). The `Diverge` function returns true when the JS divergence between the current and updated pdfs exceeds a user-defined threshold. Put differently, the distribution converges when the JS divergence becomes relatively small. We used the threshold of $JS > 0.05$ in our experiments.

If no changes were deemed necessary throughout the algorithm, Line 16 simply returns the previous version of the probability distribution.

4.2.4 Repeat

In order to prevent feedback and amplification of erroneous information, we must propagate from X to Y only the information that did not originate at Y. For example, before convoluting a child's birth year with the parent age model to predict an individual's birth year, we first *distill* any portion of the child's evidence which came from the individual. To accomplish this, we associate a unique identifier with every distribution as a lightweight form of lineage. Recall that we store the contributing sub-pdfs (or messages) for every inferred distribution at the end of the combine step. During subsequent rounds, we check during the apply step if a source variable X contains a sub-pdf from the target variable Y. If found, we compute a distilled distribution X' by removing the sub-pdf from Y and running the combine and evaluate steps as before.

As discussed in Section 4.1, the user executes the `erace` aggregate query multiple times (i.e., once per round). In our experiments, most of the inferred distributions converged in as few as 4–6 iterations.

4.3 Architecture

To evaluate the efficiency of our approach, we formulated an exact Bayesian network model for the genealogy data (see Section 5.1). At a high level, this approach took several hours on average to run exact inference in Matlab, compared to only several minutes for our database-centric approach in PostgreSQL. In addition, our framework used only a minimal amount of RAM on the order of several megabytes. The baseline approach in Matlab required anywhere from 30 to 3000 MB, often exceeding the memory of our test server. For the Markov blankets that didn't crash Matlab, average memory utilization was about 550 MB.

We do not extensively compare the actual running times between the two approaches because of their fundamental differences in implementation. Figure 7 highlights the key components and flow of each system. The main difference between the two is the role of the database. Our framework is implemented with user defined functions (UDFs) in PostgreSQL, with most of our key algorithms written in C. This greatly simplifies the query processing in SQL and allows us

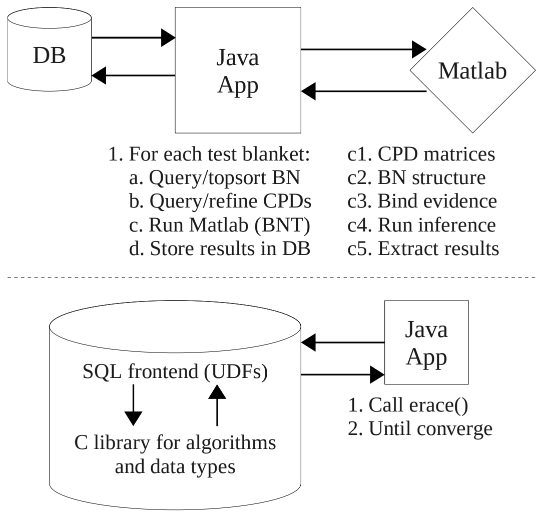

Figure 7: Comparison of system architectures: exact inference with Bayesian networks (top) versus approximate inference with data cleaning (bottom).

to piggyback the inference and cleaning operations on the actual table scans and aggregation for each phase. In contrast, the Bayesian network implementation requires us first to discover the Markov blanket for each missing node, and then move the data out of the database (blanket by blanket) into Matlab for inference.

Another fundamental difference is that our framework computes the posterior distribution of every node, not just those with missing values. This enables us to perform outlier detection and data cleaning on the fly, with little additional overhead. Adding data cleaning to the Bayesian network approach would be much more difficult because joint inference would have to be performed over the entire database, and we would lose the ability to decompose the inference process into a set of small Markov blankets.

5. EXPERIMENTS

We thoroughly evaluated the accuracy and quality of our inference framework over a variety of synthetic and real data sets. Because of the acyclic nature of the genealogy data, we were able to construct an exact Bayesian network model for comparison. Note that such a formulation is not possible for the sensor domain because the model graph contains cycles. Consequently, most of our results presented here focus on the genealogy example.

5.1 Baseline Model

To evaluate the accuracy of our approach, we first develop an exact Bayesian network model for comparison. In this case, our model consists of two CPDs (see Section 3.3): $P(I.d|I.b)$ and $P(I.b|M.b, F.b)$. To encode the CPDs, we must represent a posterior probability distribution for each variable (i.e., $I.d$ and $I.b$) for each set of conditioning values in the database (i.e., $\{I.b\}$ and $\{M.b, F.b\}$ respectively). As before, we represent these distributions using histograms, with one bin for each year in the distribution.

We construct the CPDs by aggregating the known instances in the database. For example, the CPD $P(I.d|I.b)$ is given by the following query:

```
SELECT birth, death, count(*)
FROM person
GROUP BY birth, death;
```

The prior distribution $P(I.b)$ is given by running the same query without the `death` attribute. For each resulting CPD, we convert the raw counts into probability densities by normalization (i.e., divide each count by the sum of all counts).

To infer values for the missing birth and death years in our data, we can use any standard inference algorithm to compute posterior distributions conditioned on the observed evidence in the database. For this work, we use the *junction tree* inference method included with the Bayes Net Toolbox for Matlab [17]. Exact inference in Bayesian networks is only linear (in the size of the network) if the model corresponds to a polytree, where there is at most one undirected path between any two nodes of the network. However in our domain, since two parents can have multiple children and the network can be multiply-connected, exact inference is worst-case exponential.

Rather than run exact inference over the entire database as one large graphical model, to reduce the runtime we automatically decompose the network into a set of Markov blankets, one for each group of related missing values. In a Bayesian network, the *Markov blanket* for a random variable X consists of the set of observed variables that render X conditionally independent of the rest of the network—it contains the node's parents, children, and other parents of the children (not necessarily "spouses" in terms of genealogy). If any of these nodes are also unobserved, the blanket is extended recursively along these same paths. The final subgraph contains all the information needed to jointly infer the posterior of the unobserved variables in the blanket. Note that a Bayesian network may contain multiple Markov blankets, but each unobserved node belongs to one blanket.

After we discover the Markov blankets, we construct the corresponding CPDs and perform exact inference (using the junction tree algorithm) on each one. We acknowledge that in practice one may prefer to use more sophisticated approximate inference techniques or even expectation maximization (a joint learning/inference technique). However, this approach gives a finer resolution of the resulting uncertainty and is more directly comparable with the output of our approximation framework described in Section 4.2.

5.2 Methodology

Note that the results from either inference process are marginal distributions for the missing birth and death years (or temperature and humidity values). The basic idea of our experiments is to clear-out a subset of known values (i.e., set them to NULL), run the queries from Section 4.1, and compare the results with the original values.

We are primarily interested in two measures for evaluating our framework. First, we measure how accurate the resulting marginals are with respect to the real (i.e., original) values by computing the *mean absolute error*:

$$\frac{1}{N}\sum_{i=1}^{N}|e_i - a_i|$$

where e_i is the expected value of the marginal and a_i is the actual value of the original data. This measure captures the average bias of the inference algorithm. Note we interpret the expected value as a predictor of the unknown data.

Second, we report the average standard deviation of the resulting posterior distributions. This measure essentially captures the quality of the results. Distributions with higher levels of uncertainty are less useful in practice, even if the overall average bias is relatively low. Of course the ideal result would be to maximize both the accuracy and quality of the inferences, but this is non-trivial because of the bias-variance trade-off.

In order to control the amount of missing and erroneous information in our experiments, we implemented a synthetic data generator based on Section 4.2 of [19]. The main approach is to simulate an isolated population, using a variety of parameters that determine its size and structure over time. Some of the parameters include the starting and ending year, the initial population size and age distribution, immigration rate and age distribution, divorce rate, and maximum pregnancy age. In addition we can specify the birth rate, life expectancy, and marriage age distribution for different time periods. For our experiments, we generated a diverse population of one million people born between the years 1500 and 2000.

5.3 Results for Genealogy Data

For each experiment we generate a new database by copying and altering the input data set (i.e., synthetic or PRF data). We then introduce missing and erroneous information randomly throughout the database, and proceed to discover the Markov blankets for all missing data values. Recall that related unobserved items end up in the same blanket. By nature of the underlying graphical model, many of the Markov blankets are trivial to solve, e.g., an individual with a known birth year but missing death year. We therefore constrain our experiments to blankets with five or more unobserved values. In addition, we filter out blankets with observed values outside of the interquartile range of the domain (i.e., 1650 to 1850), to minimize any bias introduced by boundary cases (i.e., windowing effects).

The next step is to learn the models for each inference algorithm. This involves the "count group by" queries for the baseline and the "histogram scan" queries for our framework. Because of this data-driven approach, we re-run this step for each new database rather than constructing them once from the original (i.e., uncorrupted) data. For fairness in our experiments over corrupted databases, we applied a coarse level of data cleaning when constructing the models for both algorithms. Specifically, we restricted our CPDs and histograms to include only those instances that fell within a lower and upper bound, e.g., $Min_{DA} \leq I.d - I.b \leq Max_{DA}$. For death ages we used 0 to 94, and for parent ages the range 18 to 50. Note that exact inference applies this domain knowledge indirectly by requiring the CPDs to span a limited range of domain values, whereas our approximate inference framework is not based on the actual values.

We then ran both implementations over one hundred nontrivial Markov blankets at each level of missing or corrupt data, and averaged the results. Most Markov blankets in this set ranged in size from 10 to 30 nodes, but some had as many as 150. Because of the high number of variables and relatively large CPD sizes, the exact inference often underflowed or ran out of memory. We therefore designed our test driver to keep running experiments until we gathered results successfully from one hundred Markov blankets at each level of missing/corrupt data.

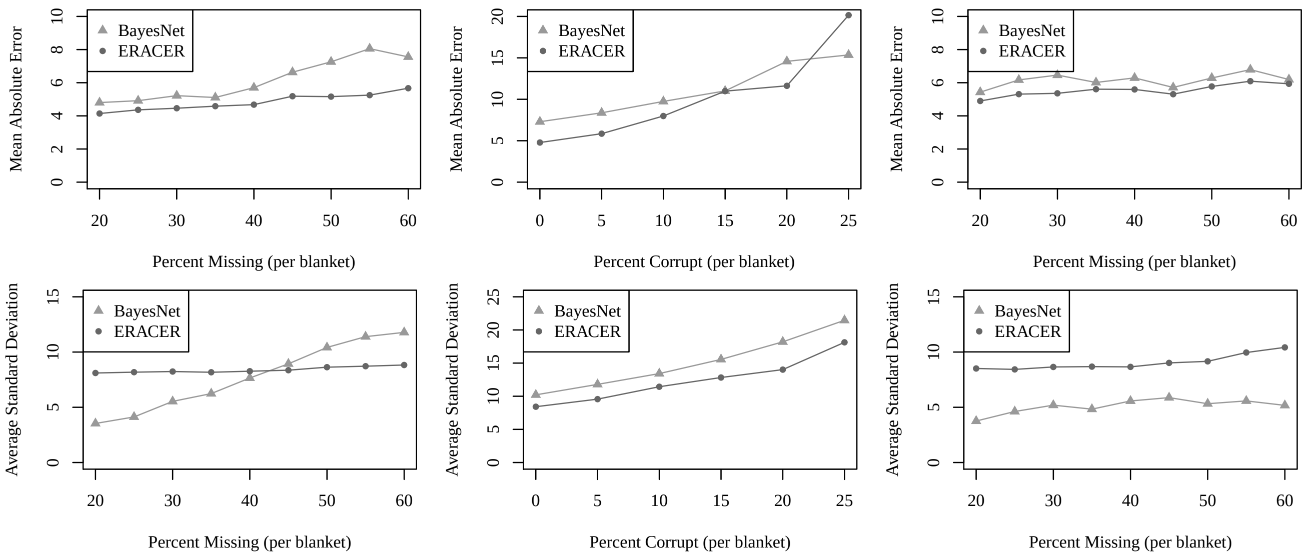

Figure 8: Accuracy (top) and quality (bottom) at varying amounts of missing and corrupt data. Our experiments with the PRF data (right) had results comparable to the synthetic data sets (left and center).

5.3.1 Missing Data

Our first experiment considers the effectiveness of the inference algorithms at varying levels of missing data. We begin by deleting the birth and death years (i.e., setting them to NULL) for a random one third of individuals in the synthetic data set. This results in a variety of Markov blankets, for which we ran both inference algorithms (i.e., junction tree and ERACER).

To our surprise, the baseline Bayesian network model initially experienced more than twice the error of our approach. This was mainly due to the sparsity of the $(I.b, M.b, F.b)$ values for estimating the CPDs. To fairly evaluate the accuracy of our approximate inference algorithm compared to exact inference, we implemented a shrinkage technique for the Bayesian network CPDs that attempts to emulate the age-based modeling approach in our framework. For each cell in the baseline CPDs, instead of estimating probabilities from counts of specific birth/death years, we use counts from all records with the same age difference, for example:

$$\begin{aligned} P(I.d = 1975 | I.b = 1942) &= \frac{|I.d = 1975 \wedge I.b = 1942|}{|I.b = 1942|} \\ &\Rightarrow \frac{|I.d - I.b = 33|}{N} \end{aligned}$$

Figure 8 (left) shows the accuracy (top) and quality (bottom) for the two inference algorithms, using this shrinkage extension for the Bayesian network approach. Both methods are able to infer the missing data within four to eight years of the actual values. Our framework is slightly more accurate, which we attribute to the additional flexibility that comes with modeling the data as a relational dependency network. The Bayesian network approach is more consistent in terms of quality: as the amount of missing data increases, the uncertainty of the marginals also increases. In contrast, the variance of our inference method depends on the component models (which in this experiment were based on convolution).

5.3.2 Corrupt Data

Our next experiment builds on the previous by introducing random errors into the synthetic database. We first select a random 15% of individual records to corrupt, in addition to the one third deleted previously. In practice, mistakes can be anything from swapped or missing digits (e.g., 1480 or 840 instead of 1840) to completely random values (e.g., -74). Although our convolution-based framework can handle any numerical value, the exact inference algorithm requires values within the discrete domain. We therefore corrupt data by replacing it with random values in the range [1500–2000).

Figure 8 (center) shows the accuracy (top) and quality (bottom) for the two inference algorithms, respectively. We found that the exact inference algorithm is somewhat resistant to corrupted data values, because of the way it makes inferences throughout each Markov blanket jointly. However, the resulting inferences have slightly higher variance. In other words, the uncertainty of the results is higher because of the contradictory evidence.

Our framework performed as well as exact inference at lower levels of corrupt data, but was more sensitive to errors overall. This is to be expected because each marginal is inferred independently, and relies on majority vote to identify errors. When the majority of related data is erroneous, errors will propagate throughout the Markov blanket. However, this performance is sufficient for many applications where the majority of information is correct.

The reason why our approximation framework continues to outperform joint inference in Bayesian networks is the built-in data cleaning. When running this same experiment without suspect identification (i.e., Lines 2–12 in Figure 5), the accuracy of approximate inference quickly deteriorated, often resulting in errors more than twice those for exact inference in Bayesian networks. We have omitted this curve from Figure 8 for clarity.

Due to space limitations, we cannot present the full results on the data cleaning aspect of our framework. In short, our implementation achieved a high accuracy of 95% in our

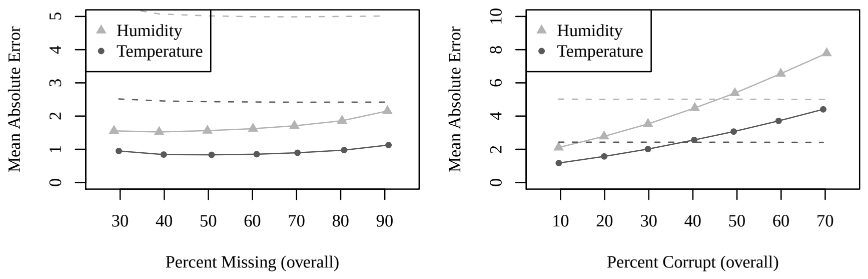

Figure 9: Accuracy of predicted sensor measurements, at varying levels of missing and corrupt data.

identification of erroneous values, compared to a baseline of 77% if we had simply identified no errors.

5.3.3 PRF Data

The experimental setup for the PRF data was a bit more involved because we did not know for certain where the mistakes in the data set were (i.e., we didn't inject them ourselves). Instead, we ran our data cleaning algorithm to completion over the entire data set, and identified the individuals having both a birth and death year that were not flagged as errors after five rounds. We then generated the test database as before, but introduced the applicable alterations within a random subset of these seemingly reliable data values.

Our experiments with the PRF data were particularly insightful because the data set already contained a significant amount of missing and erroneous information, in addition to what we introduced synthetically. Figure 8 (right) shows the accuracy (top) and quality (bottom) for the two inference algorithms, respectively. Overall, our approximate framework outperformed the exact inference algorithm, but resulted in a slightly higher level of uncertainty.

5.4 Results for Sensor Data

We evaluated the accuracy and quality of our inference framework over variants of the Intel Lab Data, which contains over 2.25 million measurements. In contrast to the genealogy experiments, we do not compare our framework with a baseline Bayesian network model. Such a formulation is not possible for the sensor domain because the model graph contains cycles. Instead, we compare with a baseline estimator that simply imputes the average value over the entire database. We also compare the accuracy of predicting temperature and humidity values, which differ in the amount of correlation present in the underlying data.

The original version of the Intel Lab Data has many data values that are clearly incorrect measurements. For example, about 18% of records have temperatures of over 100°C and/or a relative humidity of -4%. We therefore begin our experiments with a pre-processing step that sets these values to NULL. This "pre-cleaning" utility also removes multiple reports from the same time epoch. We then run our framework on the resulting data set in order to find the remaining non-trivial errors.

Figure 9 presents the results[7] for predicting humidity and temperature values. We report the mean absolute error for values that were originally known (i.e., not counting the 18% that were previously set to NULL). The trends are similar to those of the genealogy data: as the amount of missing and corrupt data increases, so does the average prediction error. However, on average our framework was able to predict temperatures within one degree Celsius, and humidity values within two percent. The dashed lines show the average error of the baseline estimator that replaces each NULL value with the mean temperature and humidity values.

These results validate the generality of our framework, as the two domains represent significantly different types of correlations. In addition, we identified a much higher amount of errors in the ILD database than in the genealogy experiments. However, the baseline estimator outperforms our inference framework when more than half of the non-NULL values are corrupt. This is to be expected, given the majority of known data is incorrect. One area of future work is to detect this situation by evaluating the amount of uncertainty in intermediate predictions and tracking the behavior of suspects identified.

6. RELATED WORK

Several recent works have shown the benefit of performing statistical inference at the database level. In particular, [5] presents a case study of implementing statistical techniques with SQL and MapReduce interfaces for deep analysis of data warehouses. [21] developed a DBMS with first-class support for continuous functions using piecewise linear segments, with regression analysis as a target application. We have taken a similar approach in our work by pushing the approximate inference computation into the database engine.

One project closely related to ours is [4], which extends belief propagation (the sum-product algorithm) for inference in graphical models to perform data cleaning. This method models dependencies in sensor networks using (undirected) Markov random fields, whereas our approach models correlations using relational dependency networks. Both methods use approximate inference methods to simultaneously fill in missing values and clean corrupted data. However, the model in [4] requires multiple observations on each node for estimation. In contrast, our method uses relational modeling techniques to tie parameters across attributes of related tuples, and thus only requires a single observation for each tuple. In addition, we apply shrinkage techniques to further improve estimation by exploiting higher-level dependencies in the data.

A more common approach to data cleaning is solving the *constraint repair problem*, which is to find a (minimal) set of data modifications that cause the database constraints to be satisfied. [3] proposes a heuristic for repair-construction based on equivalence classes of attribute values. While this is effective in data integration environments where there is a

[7]We thank the Cyber Center at Purdue University Discovery Park for the use of their servers during these experiments.

high level of redundancy, it fails to exploit other correlations in the data. In contrast, we propose a statistical method for modifying erroneous data values based on higher-level dependencies between attributes. One major benefit to using statistics instead of heuristics is that we are able to quantify the resulting uncertainty in a principled manner.

Recently, there has been a surge of interest in leveraging integrity constraints not only for controlling data quality but automatically improving it. The tutorial in [8] gives an overview of several extensions to integrity constrains for data cleaning, including *conditional functional dependencies* (CFDs) and *conditional inclusion dependencies* (CIDs). By definition, these dependencies are satisfied over given subsets of the relation (i.e., as specified by the conditions). Discovering these dependencies gives insight about which areas of the database require cleansing. [6] presents two heuristic algorithms which use CFDs to suggest repairs. More closely related to our work is [14], which considers the problem of repairing numeric values. The main idea is to utilize *denial integrity constraints*, which prevent multiple attributes from taking certain combinations of values.

Another project that reasons about integrity constraints probabilistically is [1], which focuses primarily on duplicate detection and key repairs in relational databases. Our task is not to repair primary keys (e.g., by deleting tuples), but focuses instead on inferring other attributes using a different class of relational dependencies. This follows the spirit of [12], which presents a technique for detecting corrupt values by learning generative models of clean vs dirty records and a probabilistic model of the corruption process itself. In comparison, we identify outliers by comparing them with their predicted distributions, i.e., as if they were missing.

7. CONCLUSION

We have presented two statistical, data-driven methods for inferring missing data values in relational databases: a novel approximation framework using convolution or regression, and a baseline exact method using Bayesian networks. Our system not only achieves results comparable to the baseline, it also performs data cleaning on the non-missing values, is significantly more efficient and scalable, requires a minimal amount of domain knowledge, and provides additional flexibility for exploiting the underlying dependencies. Our long term vision is to provide an SQL-based, declarative framework which enables data owners to (1) specify or discover groups of attributes that are correlated, and (2) apply statistical methods that validate and cleanse the database over time using these dependencies.

8. REFERENCES

[1] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *IEEE International Conference on Data Engineering (ICDE)*, 2006.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[4] F. Chu, Y. Wang, D. S. Parker, and C. Zaniolo. Data cleaning using belief propagation. In *2nd International Workshop on Information Quality in Information Systems*, 2005.

[5] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. Mad skills: New analysis practices for big data. In *International Conference on Very Large Data Bases (VLDB)*, 2009.

[6] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *International Conference on Very Large Data Bases (VLDB)*, pages 315–326, 2007.

[7] T. Dasu and T. Johnson. *Exploratory data mining and data cleaning*. Wiley-IEEE, 2003.

[8] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. In *International Conference on Very Large Data Bases (VLDB)*, 2008.

[9] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar. Probabilistic relational models. In *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[10] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[11] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *JMLR*, 1:49–75, 2001.

[12] J. Kubica and A. Moore. Probabilistic noise identification and data cleaning. In *IEEE International Conference on Data Mining (ICDM)*, 2003.

[13] L. Lee. Measures of distributional similarity. In *37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, 1999.

[14] A. Lopatenko and L. Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *IEEE International Conference on Data Engineering (ICDE)*, 2007.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30:122–173, 2005.

[16] H. Müller and J.-C. Freytag. Problems, methods, and challenges in comprehensive data cleansing. Technical report, Humboldt-Universität zu Berlin, Institut für Informatik, 2003.

[17] K. P. Murphy. The bayes net toolbox for matlab. In *33rd Symposium on the Interface of Computing Science and Statistics*, 2001.

[18] J. Neville and D. Jensen. Relational dependency networks. *Journal of Machine Learning Research (JMLR)*, 8:653–692, 2007.

[19] V. Ollikainen. *Simulation Techniques for Disease Gene Localization in Isolated Populations*. PhD thesis, University of Helsinki, Finland, 2002.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[21] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *ACM International Conference on Management of Data (SIGMOD)*, 2008.

[22] W. Winkler. Data cleaning methods. In *ACM Intl Conf on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.