

# **A Statistical Method for Integrated Data Cleaning and Imputation**

Chris Mayfield  
Jennifer Neville  
Sunil Prabhakar

CSD TR #09-008  
September 2009

# A Statistical Method for Integrated Data Cleaning and Imputation

Chris Mayfield, Jennifer Neville, Sunil Prabhakar

*Department of Computer Science, Purdue University  
West Lafayette, Indiana, USA*

{cmayfiel, neville, sunil}@cs.purdue.edu

**Abstract**—Real-world databases often contain both syntactic and semantic errors, in spite of integrity constraints and other safety measures incorporated into standard DBMSs. This is primarily due to the broad scope of incorrect data values that are difficult to fully express using the general types of constraints available. As a result many errors are subtle, and laborious to detect with manually-specified rules. However, combining statistical methods with extensions to conventional integrity constraints makes it possible to develop automated data cleaning methods for a variety of relational dependencies. In this work, we focus on exploiting the statistical dependencies among tuples in relational domains such as sensor networks, supply chain systems, and fraud detection. We identify potential statistical dependencies among the data values of related tuples and develop algorithms to automatically estimate these dependencies, utilizing them to jointly fill in missing values at the same time as identifying and correcting errors. The key features of our method are that (1) it uses an efficient approximate inference algorithm that is easily implemented in standard DBMSs and scales well to large databases sizes, and (2) it uses shrinkage and joint inference to accurately infer correct values even in the presence of both missing and corrupt values. We evaluate the method empirically on both synthetic and real-world genealogy data and compare to a baseline statistical method that uses Bayesian networks with exact inference. The results show that our algorithm achieves accuracy comparable to the baseline with respect to inferring missing values. However, our algorithm scales linearly rather than exponentially and can also simultaneously identify and correct corrupted values with high accuracy.

## I. INTRODUCTION

Although the database community has produced a large amount of research on integrity constraints and other safety measures to maintain and ensure the the quality of information stored in relational databases, real-world databases often still contain a non-trivial number of errors. These errors, both syntactic and semantic, are generally subtle mistakes, which are difficult or even impossible to express (and detect) using the general types of constraints available in modern database management systems. 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 automate data cleaning methods and ensure information quality in databases.

Researchers have recently begun to exploit new types of integrity constraints for data cleaning [1]. Removing data impurities is traditionally an engineering problem, where ad-hoc tools made up of low-level rules and manually-tuned algo-

rithms are designed for specific tasks. Extensions to conventional integrity constraints, together with statistical methods for data cleaning [2], make it possible to automate some of the cleansing process for a variety of domains.

For this work we consider the problem of cleaning relational databases where there are both (1) missing values to be filled in, and (2) corrupted or erroneous values to be identified and corrected. We focus on developing automated, statistical methods for domains with two important characteristics:

- The values of different attributes are correlated, both *within* and *across* tuples.
- 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 dates of parents and children are correlated due to parenting-age expectancies. Furthermore, since life expectancies and parenthood age are likely to be similar over time, the dependencies do not need to be modeled for specific birth dates, but rather can be modeled as a higher-level functional dependency (e.g.,  $birth\ year = parent's\ birth\ year + 25 + \epsilon$ ). A statistical method can *learn* these dependencies from the available data and then use them to *infer* missing values.

In this paper, we present an iterative statistical framework for inferring missing information and correcting errors based on belief propagation (see e.g., [3]), relational Bayesian networks [4], and convolution (see e.g., [5]). The imputation and cleaning tasks go hand in hand: additional information in the database helps identify errors more accurately, and corrected data values improve the quality of inference for the missing values. A salient feature of our approach is that we integrate the data cleaning into the actual inference process, rather than perform these two tasks separately. Although we focus on a genealogical domain for illustration and evaluation, the method is applicable to other relational databases with similar properties, including sensor networks, fraud detection domains, and supply-chain management.

Specifically, we propose a novel statistical approach for data cleaning and imputation. The model is a form of relational

Bayesian network [6], which allows us to specify a small model template to be *rolled out* based on the structure of the database rather than explicitly modeling the full database. Our approach uses convolution to efficiently and accurately estimate the dependencies at a higher-level than standard Bayes net conditional probability distributions. We develop an approximate inference technique based on belief propagation, which notably results in dramatic increases in efficiency. It is also possible to implement in SQL with user defined functions, and facilitates the integration of imputation and data cleaning. We evaluate the method empirically on both synthetic and real-world genealogy data, and compare to a baseline statistical method that uses Bayesian networks with exact inference. The results show that our algorithm achieves accuracy comparable to the baseline with respect to inferring missing values. However, our algorithm scales linearly rather than exponentially and can also simultaneously identify and correct corrupted values with high accuracy.

### A. Running Example

We consider the task of inferring missing birth and death years of individuals in genealogical databases. Common sources of genealogical information include vital records (e.g. birth, marriage, and death certificates), religious archives (e.g. baptisms and funerals), and public records (e.g. census data, immigration lists, and voter registrations). Integrating these sources on a large scale for public consumption is a challenging task. Ancestral File (AF) and Pedigree Resource File (PRF), available online at FamilySearch.org, are examples of such an effort. These systems include millions of individuals and their pedigrees, and are compiled from thousands of independent submissions over the past thirty years. Effective techniques for data cleaning, record linkage, and attribute standardization are crucial both during and after the integration process in order to maintain quality in the resulting data.

Without loss of generality, we assume each individual has a single birth date and death date, either of which may be unknown (i.e. NULL). We also assume that the majority of the known values are correct, but expect there to be many nontrivial errors (i.e. which cannot be detected by standard integrity constraints such as “birth year is less than death year”). Our study is designed for lineage-linked databases with a table of parent-child relationships between individuals. For simplicity we assume that all relationships are both present and correct; detecting and cleaning referential errors is a complementary problem, which we leave as future work.

### B. Contributions

The remainder of this paper follows this brief outline:

- We first present a baseline method for learning Bayesian networks, which uses exact inference to infer missing values jointly. This method provides a realistic objective for empirical comparison with our approach.
- We then outline our framework for integrated imputation and data cleaning, which uses a novel form of shrinkage based on discrete convolution and an approximate

inference technique based on belief propagation. We outline how our method performs data cleaning during the inference process to improve the resulting estimates and validate the underlying evidence.

- We discuss how our approach yields a natural implementation inside standard DBMSs (we have used PostgreSQL), and is significantly more efficient than the traditional inference techniques using Bayesian networks.
- We empirically study the accuracy and quality of the two methods on synthetic datasets with varying levels of missing and corrupt information. We show that our method achieves similar accuracy to the Bayes net method but with significant gains in runtime performance and added benefit of simultaneous data cleaning. We then demonstrate the effectiveness of our technique on a real dataset of five million individuals.

In the long term, we envision a platform for DBAs to specify or discover groups of attributes that are correlated, and then install statistical constraints that validate and cleanse the database over time. We have constructed the initial prototype of such a system for this paper, and present our findings when applied to genealogical databases.

## II. AN INITIAL APPROACH

One natural approach for inferring missing values in genealogical data is to model family tree dependencies with directed graphical models [6]. *Graphical models* leverage techniques from graph theory and probability theory to represent and reason with complex joint distributions of many random variables compactly and efficiently. We refer the reader to chapter eight of [7] for an overview of graphical models and standard inference algorithms. In this section we assume the reader has a basic understanding of Bayesian networks.

As a baseline, we outline an approach for constructing a Bayesian network from a genealogical database and estimating the model parameters from the non-null values in the database. We then apply the learned model to perform statistical inference, computing a posterior distribution for each missing value in the data conditioned on the observed evidence. We discuss the limitations of this method in practice and outline several practical implementation challenges as motivation for our approximation framework described in section III.

### A. Bayesian Network Model

Let the random variables  $I.b$  and  $I.d$  denote the birth and death years of the 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 the template for our graphical model. This approach is similar to learning a probabilistic relational model [6], a relational extension to Bayesian networks.

Figure 1 shows our directed model template using plate notation. For each individual we have a random variable representing their birth and death year values—these correspond

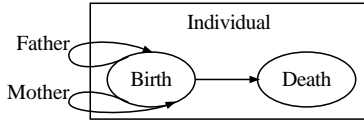


Fig. 1. Bayesian network template

to the nodes  $I.b$  and  $I.d$ . The edges represent statistical dependencies between the random variables. Edges within a plate represent dependencies among the random variables of a single individual (e.g., a person’s death year depends on their birth year). Edges that extend outside a plate represent dependencies among the random variables of *related* individuals (e.g., a person’s birth year depends on the birth year of their mother). Directed graphical models capture causal dependencies, i.e. we say that an individual’s birth date influences his or her death date. Furthermore, a father’s and mother’s birth dates jointly influence the birth dates of all of their children.

The Bayesian network structure specifies the 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* in the network. Our model template consists of two CPDs:  $P(I.d|I.b)$  and  $P(I.b|M.b, F.b)$ .

Note that this approach 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 several reasons. For one, this correlation is not a causal relationship but a domain constraint. Secondly, capturing too many constraints can lead to model overfitting and result in less accurate inferences. And finally, additional dependencies will increase the complexity of the network and significantly hinder the runtime performance.

Figure 2 illustrates an example Bayesian network which results from unrolling our model template (figure 1) 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 Bayesian network, each child’s birth date is influenced by the birth dates of both parents. Likewise, each person’s death date depends only on his or her own birth date. In this example, 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 parents of the ancestors and children of the descendants are added to the network.

### B. Learning CPDs

Our goal is to automatically *learn* the parameters of the CPDs 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 applications with similarly-

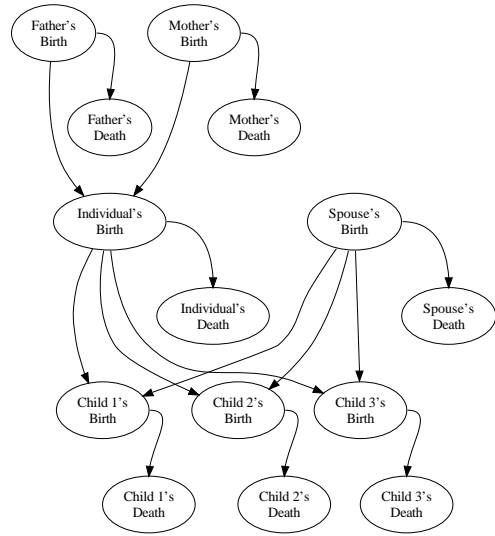


Fig. 2. Example instance of model

structured 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 dependencies directly from the genealogical data.

To encode the CPDs ( $P(I.d|I.b)$  and  $P(I.b|M.b, F.b)$ ) we must represent a posterior probability distribution for each variable (i.e.  $I.d$  and  $I.b$ ) for each set of conditioning values (i.e.  $\{I.b\}$  and  $\{M.b, F.b\}$  respectively). We represent these distributions using histograms, with one bin for each year in the distribution.

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

```
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 table, we convert the raw counts into probability densities by normalization (i.e. divide each count by the sum of all counts).

Learning the remaining distribution  $P(I.b|M.b, F.b)$  involves self-joining the person table to group each individual along with both parents:

```
SELECT
  m.birth AS mb, f.birth AS fb,
  i.birth AS ib, count(*)
FROM person AS i
  INNER JOIN relative AS r1
    ON i.ind_id = r1.ind_id AND r1.role = 'M'
  INNER JOIN person AS m ON r1.rel_id = m.ind_id
  INNER JOIN relative AS r2
    ON i.ind_id = r2.ind_id AND r2.role = 'F'
  INNER JOIN person AS f ON r2.rel_id = f.ind_id
GROUP BY mb, fb, ib
```

This process is completely data driven; we do not rely on any domain specific knowledge to learn the model parameters.

The appeal of this approach is its simplicity and general applicability. However, this will only work if the underlying database has enough information to construct the complete range of possible values. For example, if no person in the database has an observed death year of 1900, then that year will not be present in the posterior distribution of missing death years.

To adjust for unobserved combinations, we applied Laplace smoothing to the CPDs (i.e. add one to each frequency and renormalize), since many  $P(I.b|M.b,F.b)$  combinations were unobserved in the data. Although this results in a larger space of possible values, it prevents the inference engine from zeroing out intermediate potentials for the more likely answers.

Note also that the above queries have no WHERE clause. This results in a single CPD over the entire database, which can then be shared at each applicable node in the Bayesian network. However, much of the CPD will be irrelevant at any given node. For example, consider a person that is born in 1970—her parents and children will probably be born after 1900 so the CPD information for 1500-1800 will have little influence on the posterior inference.

In our experiments using real data, the above query constructs a full CPD of about 150,000 rows. Smaller CPDs are more desirable in practice because they reduce the runtime and improve the quality of the inferences. Thus we refine each CPD *locally* by dropping highly improbable entries and renormalizing. Ideally, we would estimate a lower and upper bound for each node (to minimize the size of each CPD), but determining these bounds ultimately requires running inference—the process we are currently setting up.

A compromise between these two extremes is to estimate the lower and upper bound of birth and death years for each inference *subgraph* or *Markov blanket* (described in more detail in the next section). We do this by identifying the minimum and maximum generations of observed and unobserved nodes in each Markov blanket, and apply a heuristic based on maximum parenthood age in the database to estimate the overall range of the entire blanket.

### C. Exact Inference

To infer values for the missing birth and death years in our data, we can use any 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 [8]. 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. In our case, since two parents can have multiple children, the network will be multiply-connected and inference is worst-case exponential in the size of the network.

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 exactly one blanket.

Once we find the Markov blankets, we can then perform exact inference on each blanket individually. For each Markov blanket in the database, we construct the corresponding Bayesian network and local CPDs. We then use the junction tree inference algorithm in the Bayes Net Toolbox to compute the posterior distribution of each missing value, given the observed evidence. 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 our approximation framework in section III.

One main challenge with running inference at this level of granularity is dealing with floating point underflows. The total number of probability assignments for a Markov blanket is proportional to the size of the CPDs and the number of unobserved nodes. For example, a typical Markov blanket ranges over 200 years, and has twenty birth and death nodes (forty total), of which fifteen are unobserved. The likelihood value for an assignment to these variables may become zero in hardware is relatively high, given each of the fifteen nodes can take up to 200 possible discrete values.

In addition to the computational issues of applying exact inference methods, there are a number of limiting assumptions of the Bayesian network approach which do not always hold in practice. Up to this point we have assumed there are no errors in the observed data. One common type of error is a simple typo, for example a birth year of 840 instead of 1840, and the Markov blanket is estimated to range from 1700 to 1900. In our implementation we treat such nodes as unobserved, since there is no mapping from their values into the discrete domain of the CPD. Outliers are even more problematic when learning the CPDs, as the presence of incorrect data values makes estimating the lower and upper bounds for the Markov blanket highly unreliable.

The graphical model template also imposes structural constraints which may not hold in the case of dirty databases. For instance, we cannot directly apply this model to an individual with more than two parents. We omit Markov blankets with structural anomalies from our experiments, for the sake of comparing the two algorithms.

## III. OUR FRAMEWORK

The Bayesian network formulation is attractive due to its principled mathematical formulation. However as discussed in the previous section, it has a number of limitations:

- Inefficiency and (potential) inaccuracy due to a large number of parameters in the CPDs
- Inflexibility due to fixed CPD structure
- Exponential exact inference algorithm
- Inability to identify and clean *corrupt* data values

In this section we propose a novel framework that uses a shrinkage technique and approximate inference to offset these problems and significantly improve runtime efficiency. The algorithm makes an assumption of conditional independence among parents and children that allows us to relax the restrictions on the CPD structures. In addition, the algorithm is flexible enough to infer a posterior distribution for *every* variable, which we use to identify outliers as errors for data cleaning. Finally our method provides a natural implementation within database systems via SQL and user defined functions.

The main ideas and contributions of this alternative approach are: 1) a shrinkage technique using discrete convolution, and 2) an iterative approximation algorithm based on belief propagation. *Shrinkage* is a statistical approach to improve the accuracy of a *local* estimator by incorporating additional (e.g., *global*) information to reduce the effects of local sampling variation. For example in our application, we can construct more accurate CPDs by considering not only the counts for specific combinations of birth and death years, but also incorporating the counts for *similar* combinations. Belief propagation is an iterative algorithm used for approximate inference in graphical models. We apply a similar message passing technique in our framework.

#### A. Convolution Models

Our proposed model is similar in structure to the Bayesian network model in figure 1. We model the missing data values with random variables  $I.b$  and  $I.d$  with the same dependencies. However, instead of choosing a fixed structure for the CPDs, we model each parent’s influence independently and aggregate the information during inference. This approach is often used in relational learning to tie model parameters across heterogeneous structures, for instance related individuals with varying numbers of parents (see e.g., [4]).

In addition, we observe that the dependencies between parent and child birth year are likely to be similar across different years. The functional dependency is more likely 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* and use this instead of the value specific CPDs of the Bayesian network.

Consider a pair of attributes that correlated through this kind of function, for example 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 year as an individual’s *parent age*:  $M_{PA} = P(P.b - I.b)$ . These two distributions correspond to the two types of edges in figure 1.

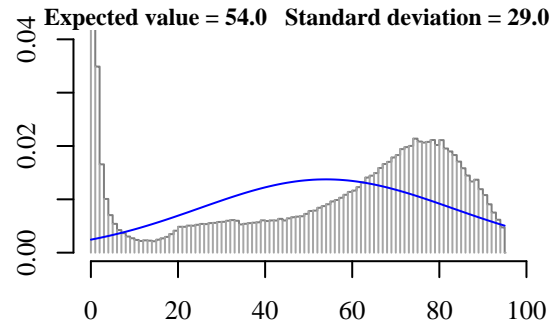


Fig. 3. Convolution model for “Death Ages”

Since we constructed the distributions using subtraction, we use convolution to “add” information and make inferences between two attributes. See the *apply* step in section III-C 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 is the shrinkage aspect of the model.

#### B. Learning Dependencies

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 *model* that returns a (normalized) histogram of the values it scans:

```
SELECT model(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 an example histogram using our real dataset. The spike on the left side suggests a high child mortality rate for this dataset. We have also computed the mean and standard deviation of this histogram, and plotted the normal distribution for these parameters. Clearly, the normal distribution does not provide a good approximation.

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

```
SELECT model(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
```

Notice that in contrast to the queries in section II-B, we no longer need to group by the individual instance values, resulting in a more efficient query plan. We have also eliminated two joins by shrinking across both parents.

These queries also require very little domain knowledge, namely that birth dates and death dates for an individual are correlated, as are the birth dates 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. For example, we can learn such models for neighboring nodes in a sensor network.

### C. Approximate Inference

Using the difference distributions constructed in the previous steps, we develop an iterative, approximate inference procedure to estimate posteriors for each missing value and identify/clean 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 Markov blankets over which to do (exponential) joint inference. At a high level, the inference process iterates over three major phases:

- 1) **Apply** models: for each value (or distribution) modified in the previous round, construct (or update) an output distribution for each applicable model and relative.
- 2) **Combine** inferences: aggregate and normalize the resulting predictions for each individual, and detect conflicting evidence values using elections.
- 3) **Evaluate** changes: for each individual, accept or reject the resulting distribution after comparing it with the previous version.

We now explain the details of each step below in the context of our genealogy example. Each step corresponds to a separate UPDATE statement. The iteration repeats until all probability distributions converge, i.e. when the update count in the final step is zero. In our experiments, most datasets converged after about five iterations.

1) *Apply*: Since we constructed the difference distributions ( $M_{DA}, M_{PA}$ ) using subtraction, we can use addition to make inferences between two 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 algorithm will propagate these inferred posterior distributions as new evidence, for example to infer birth dates 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 two random variables:

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

This gives us the probability distribution of the sum of these two random variables.

For simplicity, we first construct initial point distributions<sup>1</sup> over all the non-missing data. As a result we do not need to differentiate between original evidence (scalar values) and intermediate inferences (histograms) when applying inference—we can just use convolution throughout the entire algorithm.

<sup>1</sup>The point distribution over a known value is simply a histogram with one bin, i.e. the value itself with a probability of one attached to it.

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 will show empirically that the resulting accuracy of this approximation is quite good in practice.

The inference algorithm will operate in a manner similar to belief propagation [3], iterating over the apply, combine, evaluate steps, and then propagating inferences about an individual to its parent and children in the next step. To prevent feedback and amplification of erroneous information, we need to make sure to propagate from  $X$  to  $Y$  only the information that did not originate at  $Y$ . To do this, we use a lightweight form of lineage and only propagate those portions of a predicted distribution that did not originate with the target. For example, before convoluting the child’s birth year with the parent age model to predict the 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 history identifier with each inferred distribution, which represents the origin or lineage of that data.

2) *Combine*: The next step is to aggregate the predictions from the *apply* step. We interpret each bin of the histograms  $P_1..P_j$  as a weight for their corresponding values, and simply add up the “votes” bin by bin 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 interpret each piece of evidence  $P_j$  with equal value. As future work we plan to extend this method to compute a weighted sum, depending for example on 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 a type of *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. Otherwise, if there is a single continuous probability distribution, we compute a 95% confidence interval, drop the tails, and renormalize the resulting distribution. This 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. Note the gray distribution on the right likely resulted from an erroneous

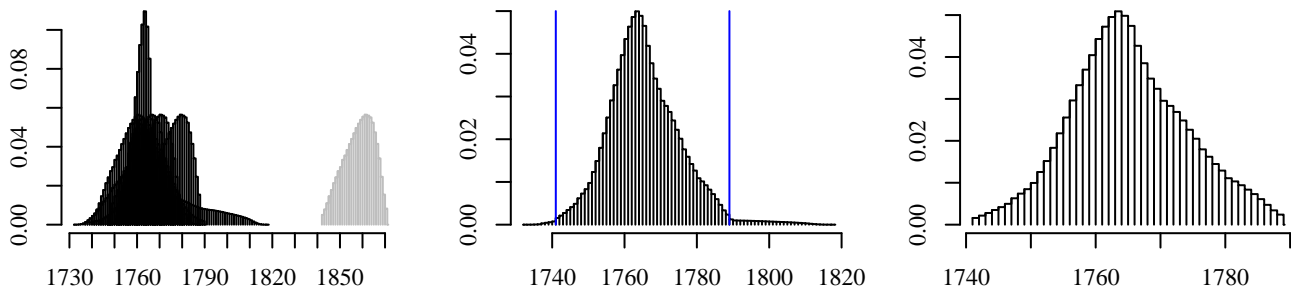


Fig. 4. Visual example of combining inferences (left), computing the confidence interval (middle), and normalizing the resulting distribution (right).

**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*

Fig. 5. Algorithm for evaluating updates

piece of evidence, and will be discarded since it is a disjoint region of (relatively) low probability mass.

Once we have aggregated the inferences, trimmed, and renormalized the posterior distributions, the algorithm *evaluates* the resulting distribution to identify outliers and/or terminate the inference process. To facilitate feedback prevention, the algorithm stores the contributing *sub-pdfs* along with each combined distribution.

3) *Evaluate*: In the final step of each iteration, we compare the combined distribution with its previous version. Figure 5 summarizes our algorithm for evaluating updated inferences at the end of each round. In addition to filling in missing values, the other objective of our framework is to identify potential errors in the underlying evidence 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 point distributions of the observed values.

Lines 1–12 analyze the updated distribution to validate existing data. In the first case (lines 3–6), when the current

**Outlier:** is the original range outside of the inferred?  
 Input: known pdf  $P$  and inferred pdf  $V$   
 Output: true if  $P$  lies outside of  $V$ , false otherwise

1.  $[a, b] := CI(P)$
2.  $[c, d] := CI(V)$
3. **return**  $a < c$  **or**  $b > d$

Fig. 6. Algorithm for outlier detection

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 an outlier if the majority of it is not contained within the majority of the inferred distribution resulting from the *combine* step.

If the distribution is determined to be an outlier, we set the *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 date is estimated to be between 1819 and 1871. Since 840 lies outside this interval, we mark it as an error and replace it with the inferred range.

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 return the original point distribution.

Line 12 essentially means “treat suspicious base data as if it were missing,” allowing the inference to be refined during subsequent rounds. If no changes were deemed necessary throughout the algorithm, line 16 simply returns the current version of the distribution.

Lines 13–15 address the case of inferring missing data. The `Diverge` function calculates the Jensen-Shannon divergence to measure the difference between the inferred distribution and

its previous version (if any):

$$JS(P, Q) = \frac{1}{2} [KL(P, A) + KL(Q, A)]$$

where  $KL$  is the Kullback-Leibler divergence:  $KL(P, Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$  and  $A$  is the average distribution:  $A(P, Q) = \frac{1}{2} \sum_i [P(i) + Q(i)]$ .

In other words, the JS divergence is the average KL divergence from each distribution to the average between them. All three of these measures can be computed efficiently in a single loop. We chose JS divergence for our framework based on the empirical results in [9]. Of course, other distributional similarity functions may be more desirable for other applications.

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.

The output distribution from the `evaluate` function is then propagated for inference in the next round. We attach the latest sub-pdfs to the resulting distribution (or the previous value, if not replaced) so that the distribution can be *distilled* appropriately. Before sending a posterior distribution to a particular neighbor  $Y$ , the distilling process re-computes the distribution as if all evidence from  $Y$  had been missing. This means we must repeat all previous steps, including election and outlier detection, before determining the appropriate distribution to propagate. In some rare cases, that evidence may have been the deciding factor in whether to flag an error or abort inference altogether. For the most part, however, this function simply ensures that all information flows in a single direction: either from ancestors to descendants, or vice versa.

#### IV. EXPERIMENTS

We thoroughly evaluated the accuracy and quality of our inference framework, using exact inference in Bayesian networks as a baseline. Our test server was a 2.4 GHz Pentium 4 with 2 GB of RAM, running Linux 2.6.27.10, PostgreSQL 8.3.5, and Matlab R2008a.

##### A. Data Sets

1) *Population Generator*: 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 [10]. The main approach is to simulate an isolated population, given 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.

The actual simulation process behaves as follows. We first generate an initial population of *founders* with birth years corresponding to the age distribution for the starting year. When constructing new individuals, the simulator determines their

birth and death years, their gender (uniformly distributed), and the year they will be eligible to marry and have children. The outer loop of the simulation updates the population each year by introducing new *immigrants*, removing deceased individuals, terminating and arranging marriages, and adding *newborns* to eligible couples. When finished, the simulator outputs two relations: the set of individuals and their corresponding parent-child relationships.

For our experiments we generated a population of one million people born between the years 1500 and 2000. We based most of our parameters on appendix B of [11], which models the relatively isolated Finnish Kainuu sub-population over the past several centuries. The main difference between our simulator and theirs (and the corresponding parameter values) is how we model population growth. Rather than control the target population size for each year, we model births with Poisson processes and maintain a constant rate of immigration. We also discard individuals with no parents or children, which are less interesting for our experiments.

2) *Pedigree Resource File*: We also obtained a sample of five million individuals from the Pedigree Resource File<sup>2</sup> (PRF), a lineage-linked database of records submitted to FamilySearch.org. This data set was originally designed for research on record linkage techniques for genealogical data, for which this work is complementary. Overall, about 68% of individuals 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 be missing information.

The PRF data set has several additional properties which make it particularly interesting for testing our algorithms. 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 in data integration, namely cleaning erroneous attribute values and repairing structural inconsistencies. It also contains duplicate records for some individuals, which further complicates the integrity of Markov blankets in the case of the baseline approach.

##### B. Methodology

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

1) *Evaluation Measures*: We are primarily interested in two measures for evaluating the approaches we have presented in this paper. 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, if we interpret the expected value as a predictor of the unknown data.

<sup>2</sup>We wish to thank Randy Wilson for providing us with this data set, and for several insightful discussions about genealogical data management.

Secondly, 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 accuracy and quality, but this is non-trivial because of the bias-variance trade-off.

A common alternative to finding representative values for probability distributions is to compute the *most probable explanation* (MPE) for each Markov blanket. The MPE is a joint assignment of missing birth and death years that results in maximum likelihood globally, in contrast to our approach of computing the expected value of each marginal locally. However, because the underlying inference models are highly skewed (e.g. relatively high infant mortality rates in life expectancy), we found the results for exact inference using MPE are generally worse than simply using the expected value. In addition, we are more interested in storing the entire range of possible birth years for future use, rather than the single most likely value.

2) *Experimental Setup*: For each experiment we generate a new database by copying and altering one of the aforementioned data sets. 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 generate the models for each inference algorithm, based on the modified data set. This involves the “count group by” queries for exact inference 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 construct them once from the original (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 non-trivial 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

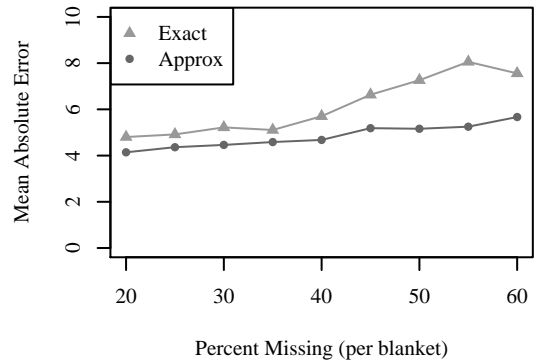


Fig. 7. Accuracy at varying amounts of missing data

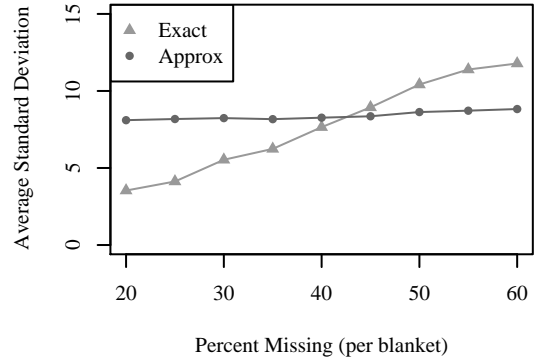


Fig. 8. Quality at varying amounts of missing data

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 for each level of missing/corrupt data.

### C. Results

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.

To our surprise, the Bayes net model (as described in section II) experienced more than twice the error of our approach. This is mainly due to the effects of shrinkage and the sparsity of the  $(I.b, M.b, F.b)$  values for estimating the Bayes net CPDs. To fairly evaluate the accuracy of our approximate inference algorithm compared to exact inference, we implemented a shrinkage technique for the Bayes net model that attempts to emulate the age-based convolution in our approach. For each cell in the Bayes nets CPDs (see section II-B), instead of estimating probabilities from counts of specific birth/death years (e.g.,  $P(I.d = 1975|I.b = 1942) = |I.d = 1975 \wedge I.b = 1942|/|I.b = 1942|$ ), we use the counts from all records with the same age difference (e.g.,  $P(I.d = 1975|I.b = 1942) = |I.d - I.b = 33|/N$ ).

Figures 7 and 8 show the accuracy and quality for the

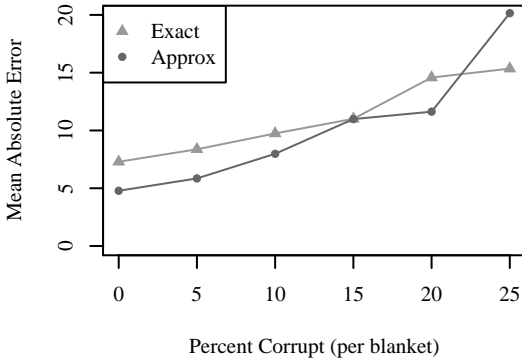


Fig. 9. Accuracy at varying amounts of corrupted data

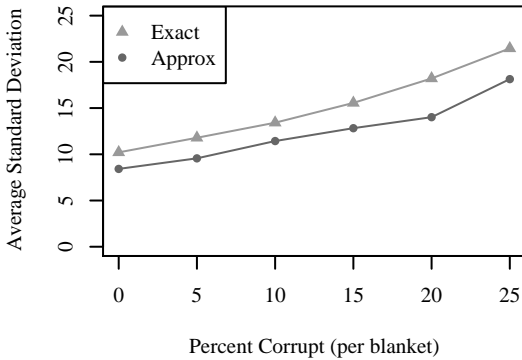


Fig. 10. Quality at varying amounts of corrupted data

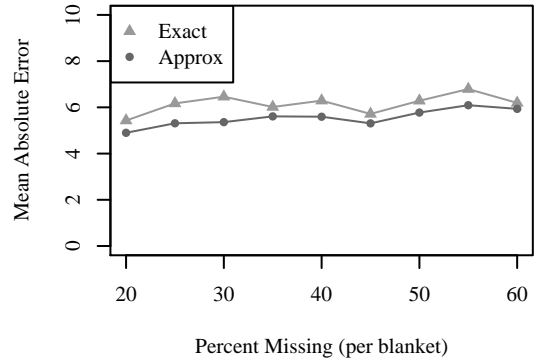


Fig. 11. Comparable accuracy using real data from PRF

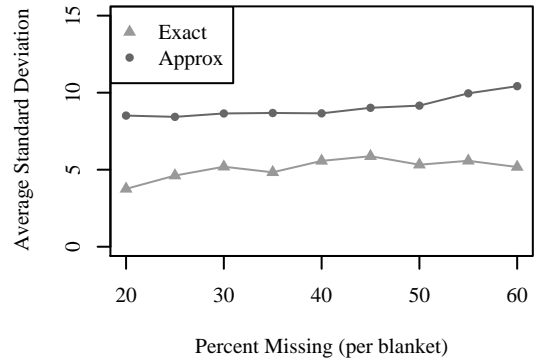


Fig. 12. Comparable quality using real data from PRF

two inference algorithms, using this shrinkage extension for the baseline 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, but the Bayesian network approach is more consistent in terms of quality. This is because it infers the posterior distributions jointly. As the amount of missing data increases, our framework must apply convolution additional times as the data propagates across generations. Consequently, the internal distributions increase in size which results in higher standard deviations.

2) *Corrupted 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 domain.

Figures 9 and 10 show the accuracy and quality 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, then 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 deteriorates, often resulting in errors more than twice as those for exact inference. We have omitted this curve from figure 9 for clarity.

3) *Real Data*: The experimental setup for the PRF data is a bit more involved because we do not know for certain where the mistakes in the dataset are (i.e. we didn't inject them ourselves). Instead, we ran our data cleaning algorithm to completion over the entire dataset, 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. Figures 11 and 12 show the accuracy and quality for the two inference algorithms, respec-

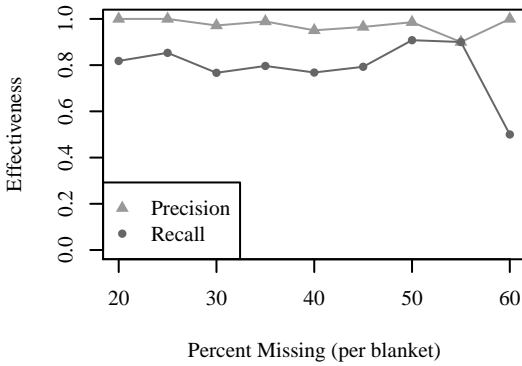


Fig. 13. Data cleaning scores, by amount missing

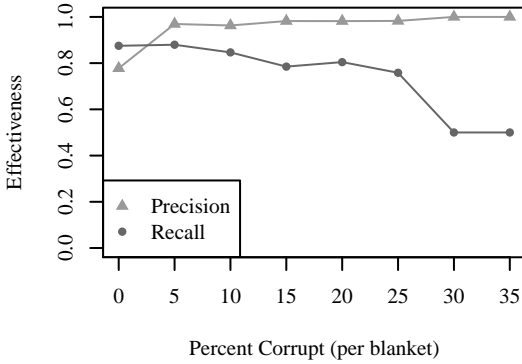


Fig. 14. Data cleaning scores, by amount corrupt

tively. Overall, our approximate framework outperformed the exact inference algorithm, but resulting in a slightly higher level of uncertainty.

4) *Data Cleaning*: Our last set of experiments study the effectiveness of the data cleaning in terms of how many errors we were able to identify in our synthetic dataset. The following table summarizes the results for the previously discussed experiments over corrupted data.

True Positives: 1171	Corrupt: 23.3 %
True Negatives: 4813	Accuracy: 94.8 %
False Positives: 28	Precision: 97.7 %
False Negatives: 298	Recall: 79.7 %

The 600 random Markov blankets selected for those experiments consisted of 6,310 individuals, of which 1,469 had corrupted (i.e. randomized) birth years. The true positives were the erroneous values that our framework successfully identified, and the false positives were correct values that we mistakenly cleansed as outliers. Overall however, we achieved a high accuracy of 95% in our identification of erroneous values (compared to the baseline of 77% if we had simply identified no errors).

Our framework was highly accurate in the assessment of incorrect data but somewhat less effective at identifying all the errors. Figures 13 and 14 show the precision and recall at varying levels of missing and corrupted data. The system achieved high precision for the errors it identified, but the recall began to drop at lower levels of quality in the underlying

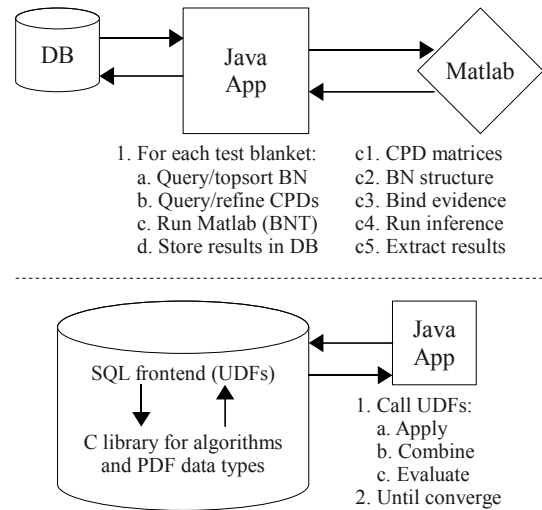


Fig. 15. Comparison of system architectures: Exact inference with Bayesian networks (top) versus approximate inference with data cleaning (bottom).

dataset. This is to be expected for two reasons. For one, the absence of good evidence diminishes the effect of the elections and outlier detection subroutines. Secondly, our experimental design which was necessary for the Bayesian network implementation produced some errors that are nearly impossible to detect. In fact, a significant number of the randomized values fell within 25–50 years of the actual values, which often satisfied the dependencies.

#### D. Performance

At a high level, our experiments took several hours on average to run exact inference in Matlab, but 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 naive 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 15 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 and allows us to piggyback the inference and cleaning operations in the actual table scans for each phase. In contrast, the BNT implementation requires us to discover the Markov blankets, 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 all nodes, not just those for missing values. This enables us to perform 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 inference into a set of small Markov blankets.

## V. RELATED WORK

Data cleaning is a well studied problem, but is far from solved in many application domains and there is certainly no catch-all solution. Two surveys of common techniques and general challenges in this research area are [2] and [12]. More recently, there has been a surge of interest in leveraging integrity constraints not only for enforcing data quality but automatically improving it [1]. We have taken this approach in our work as well.

The work most closely related to ours is [13], which extends belief propagation (aka the sum-product algorithm) for inference in graphical models to perform data cleaning. Their method models dependencies in sensor networks using (undirected) Markov random fields, whereas our approach models correlations using (directed) Bayesian networks. Both methods use approximate inference methods to simultaneously fill in missing values and clean corrupted data. However, the model in [13] requires multiple observations on each node (e.g., sensor readings) 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 (e.g., individual). In addition, we apply shrinkage techniques to further improve estimation by exploiting higher-level dependencies in the data.

Another related project that reasons about integrity constraints probabilistically is [14], which focuses primarily on duplicate detection and key repairs in relational databases. Our task is not to repair keys (e.g., by deleting tuples), but focuses instead on inferring other attributes using a different class of functional dependencies.

In our framework, we manage intermediate evidence during the inference and cleansing process with probability distributions. There are many parallels with recent advancements in probabilistic database management systems such as Orion [15], MayBMS [16], Trio [17], BayesStore [18], and MystiQ [19]. This line of work has shown the great benefit of managing uncertainty of data inside database engines, enabling query optimizers and storage managers to exploit the uncertainty for increased performance. A particularly promising area of application development using these systems is statistical inference and data cleaning over extended periods of time, where the intermediate results persist and improve incrementally as new evidence arrives.

The uncertain data community has also demonstrated other approaches to data cleaning. [20] proposes new metrics for information quality based on entropy and possible worlds semantics, and shows how to reduce uncertainty by maximizing these values within a specified budget. [21] reduces the problem of conditioning probabilistic data (i.e. adding new evidence) to computing tuple confidence values.

## VI. CONCLUSION

We have presented two statistical, data-driven methods for inferring missing data values in relational databases: a baseline exact method using Bayesian networks, and a novel approximation framework using shrinkage and convolution. 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.

## REFERENCES

- [1] 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.
- [2] H. Müller and J.-C. Freytag, "Problems, methods, and challenges in comprehensive data cleansing," Humboldt-Universität zu Berlin, Institut für Informatik, Tech. Rep., 2003.
- [3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [4] L. Getoor and B. Taskar, Eds., *Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [5] G. Casella, R. L. Berger, and R. L. Berger, *Statistical inference*. Duxbury Pacific Grove, CA, 2002.
- [6] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar, "Probabilistic relational models," in *An Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [7] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [8] K. P. Murphy, "The bayes net toolbox for matlab," in *33rd Symposium on the Interface of Computing Science and Statistics*, 2001.
- [9] L. Lee, "Measures of distributional similarity," in *37th Annual Meeting of the Association for Computational Linguistics*, 1999, pp. 25–32.
- [10] V. Ollikainen, "Simulation techniques for disease gene localization in isolated populations," Ph.D. dissertation, University of Helsinki, Finland, 2002.
- [11] H. Toivonen, P. Onkamo, K. Vasko, V. Ollikainen, P. Sevon, H. Mannila, M. Herr, and J. Kere, "Data mining applied to linkage disequilibrium mapping," *The American Journal of Human Genetics*, vol. 67, no. 1, pp. 133–145, 2000.
- [12] W. Winkler, "Data cleaning methods," in *ACM Intl Conf on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.
- [13] 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.
- [14] P. Andritsos, A. Fuxman, and R. Miller, "Clean answers over dirty databases: A probabilistic approach," in *IEEE International Conference on Data Engineering (ICDE)*, 2006.
- [15] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng, "Database support for probabilistic attributes and tuples," in *IEEE International Conference on Data Engineering (ICDE)*, 2008, pp. 1053–1061.
- [16] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *IEEE International Conference on Data Engineering (ICDE)*, 2008, pp. 983–992.
- [17] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "Uldbs: Databases with uncertainty and lineage," in *International Conference on Very Large Data Bases (VLDB)*, 2006.
- [18] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein, "Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models," in *International Conference on Very Large Data Bases (VLDB)*, 2008, pp. 340–351.
- [19] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.
- [20] R. Cheng, J. Chen, and X. Xie, "Cleaning uncertain data with quality guarantees," in *International Conference on Very Large Data Bases (VLDB)*, 2008, pp. 722–735.
- [21] C. Koch and D. Olteanu, "Conditioning probabilistic databases," in *International Conference on Very Large Data Bases (VLDB)*, 2008, pp. 313–325.