2011

Deriving Probabilistic Databases with Inference Ensembles

Julia Stoyanovich  
*University of Pennsylvania*

Susan B. Davidson  
*University of Pennsylvania, susan@cis.upenn.edu*

Tova Milo  
*Tel Aviv University*

Val Tannen  
*University of Pennsylvania, val@cis.upenn.edu*

Follow this and additional works at: [http://repository.upenn.edu/cis_papers](http://repository.upenn.edu/cis_papers)

Part of the [Computer Sciences Commons](http://repository.upenn.edu/cis_papers)

Recommended Citation


DOI: 10.1109/ICDE.2011.5767854

©2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This paper is posted at ScholarlyCommons, [http://repository.upenn.edu/cis_papers/648](http://repository.upenn.edu/cis_papers/648).

For more information, please contact repository@pobox.upenn.edu.
Deriving Probabilistic Databases with Inference Ensembles

Abstract
Many real-world applications deal with uncertain or missing data, prompting a surge of activity in the area of probabilistic databases. A shortcoming of prior work is the assumption that an appropriate probabilistic model, along with the necessary probability distributions, is given. We address this shortcoming by presenting a framework for learning a set of inference ensembles, termed meta-rule semi-lattices, or MRSL, from the complete portion of the data. We use the MRSL to infer probability distributions for missing data, and demonstrate experimentally that high accuracy is achieved when a single attribute value is missing per tuple. We next propose an inference algorithm based on Gibbs sampling that accurately predicts the probability distribution for multiple missing values. We also develop an optimization that greatly improves performance of multi-attribute inference for collections of tuples, while maintaining high accuracy. Finally, we develop an experimental framework to evaluate the efficiency and accuracy of our approach.

Disciplines
Computer Sciences

Comments
DOI: 10.1109/ICDE.2011.5767854

©2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/648
Deriving Probabilistic Databases with Inference Ensembles

Julia Stoyanovich 1, Susan Davidson 1, Tova Milo 2, Val Tannen 1

1University of Pennsylvania, Philadelphia, PA, USA
{jstoy, susan, val}@cis.upenn.edu
2Tel Aviv University, Tel Aviv, Israel
milo@cs.tau.ac.il

Abstract—Many real-world applications deal with uncertain or missing data, prompting a surge of activity in the area of probabilistic databases. A shortcoming of prior work is the assumption that an appropriate probabilistic model, along with the necessary probability distributions, is given. We address this shortcoming by presenting a framework for learning a set of inference ensembles, termed meta-rule semi-lattices, or MRSL, from the complete portion of the data. We use the MRSL to infer probability distributions for missing data, and demonstrate experimentally that high accuracy is achieved when a single attribute value is missing per tuple. We next propose an inference algorithm based on Gibbs sampling that accurately predicts the probability distribution for multiple missing values. We also develop an optimization that greatly improves performance of multi-attribute inference for collections of tuples, while maintaining high accuracy. Finally, we develop an experimental framework to evaluate the efficiency and accuracy of our approach.

I. INTRODUCTION

Many real-world applications deal with uncertain or missing data. For example, data integration often has to deal with inconsistencies across sources, while in scientific data management experimental results are often noisy or missing. Nevertheless, it is important for such applications to effectively manage the uncertain information being collected, motivating research on probabilistic databases.

Most research on probabilistic databases assumes that a probability distribution over missing or noisy values is known, presumably supplied by a domain expert. However, such quantitative input rarely exists in practice. The goal of this paper, therefore, is to provide a framework for deriving probability distributions for the missing data.

As an example, consider the incomplete relation $\mathcal{R}$ in Fig. 1 that describes a fictional dataset of personal profiles in a matchmaking site such as eHarmony.com. Profiles are described by four non-key attributes drawn from discrete domains: age $\in \{20, 30, 40\}$, income $\in \{\$50K, \$100K\}$, education $\in \{HS, BS, MS\}$, and netWorth $\in \{\$100K, \$500K\}$, with "?" indicating a missing value. Analyzing this relation, we notice that some trends hold among known values, e.g., that higher age often co-occurs with higher income (inc), and that higher income often co-occurs with higher net worth (nw). Using this type of information, we wish to produce a collection of predictions (estimations) of discrete probability distributions $\Delta_t$, one for each incomplete tuple $t \in \mathcal{R}$. $\Delta_t$ consists of all possible combinations of values of the attributes missing in $t$, each annotated with a probability, and with the probabilities adding to 1. Each combination of values corresponds to a possible complete version of $t$, hence these combinations are mutually exclusive. For example, $\Delta_{t_{12}}$ is given in a call-out in Fig. 1.

A. Background

Probabilistic databases The semantics of a probabilistic database is a probability distribution on a set of possible worlds, these being complete and fully determined (i.e., usual) database instances [26]. Since the number of such possible worlds may be large, this is too unwieldy to be represented and manipulated as such. Therefore, following some early work in [14], [22], [28], much research has focused on compact representations for probabilistic databases and on efficient query answering algorithms over such compact representations, e.g., [2], [3], [7], [16], [20], [24], [25], [27].

By making various independence assumptions, these approaches restrict the probability space from that of all possible worlds to combinations of much smaller probability spaces, leading to more compact representations. For example, in the independent-tuple model [7] each tuple is included in a possible world with some probability, independently of the others. In the disjoint-independent model [8] blocks of tuples are independent of each other, with each block consisting of a probability distribution on mutually exclusive tuples.
Probabilistic databases that we derive in this paper adhere to the disjoint-independent model. As illustrated in Fig. 1, each incomplete tuple gives rise to a distribution on a block of complete tuples. A possible world is obtained by choosing one complete tuple from each block, e.g., \( t_{12.2} \) is chosen from the block that corresponds to \( t_{12} \). The assumption that these choices are independent allows one to compute the probabilities of possible worlds or of query answers [8].

Learning a probabilistic model We now consider the question that is central to this paper. Given an incomplete database, which probabilistic model does it obey? A reasonable assumption is that each tuple in the observed relational dataset was generated independently and by the same statistical process. However, even if tuples are jointly independent there may exist correlations between the values of the different attributes in each tuple. A further reasonable and, in fact, common assumption is that attributes are random variables of a Bayesian network (BN) or of a Markov network (MN).

Graphical models such as BN and MN represent a joint distribution over a set of random variables. When available, these models may be used for exact statistical inference. An important drawback of these models, however, is that they are computationally very expensive to learn [21]. The structure of a BN is typically learned using one of three approaches. Constraint-based approaches identify independencies that hold over the data, and are sensitive to failures of individual independence tests. Score-based methods overcome this sensitivity, addressing learning as a model-selection problem, while Bayesian model averaging methods generate an ensemble of possible structures. The latter two methods select models from a hypothesis space of super-exponential size in the number of random variables — \( 2^{O(n^2)} \), and often resort to approximation.

Even if we know the probability model, the question remains how to effectively derive the probability values. One approach proposed in the literature is to obtain subjective weights and scores from domain experts and to then work them into a statistical model [10]. However, such quantitative input from domain experts is frequently unavailable.

Dependency networks The approach we take in this paper is to use dependency networks (DN) to learn the probability model [17]. Unlike exact representations such as BN and MN, a DN is an approximate representation; it approximates the joint distribution with a set of conditional probability distributions (CPDs) that are learned independently. This learning approach is significantly more efficient than the learning of exact models, because of its locality. However, because CPDs are learned independently, a DN is not guaranteed to specify a consistent probability distribution, in the sense that there is no joint distribution from which each of the local CPDs may be obtained via the rules of probability. For example, given a set of random variables \( x \), it is possible that the estimator of \( p(x_1 | x - \{ x_1 \} ) \) discards \( x_2 \) as an input, whereas the estimator \( p(x_2 | x - \{ x_2 \} ) \) retains \( x_1 \) as an input. Nonetheless, if the dataset is sufficiently large, strong inconsistencies will be rare because each CPD is learned from the same data, which is, in turn, generated from a single joint distribution. Importantly, Gibbs sampling inference techniques can be used to recover an approximation of the full joint probability distribution, regardless of the consistency of the local CPDs. This is justified both formally and experimentally in [17].

To improve consistency, ultimately improving the accuracy of inference, we use an ensemble based on Bayesian voting [11] to represent each local CPD. In particular, we mine the dataset for association rules, and use combinations of association rules, termed meta-rules, as CPD estimates. One or several meta-rules will be applicable to any inference task. We arrange meta-rules in a hierarchy, termed MRSL (for meta-rule semi-lattice). This hierarchy acts as an ensemble, and is used for inferring probability distributions over the missing values. Our approach turns out to scale well and to produce accurate estimates in a manner that is largely independent of network topology. We now give a detailed overview of our approach.

B. Our Approach

In this paper we will assume that the observed incomplete database consists of a single relation. If the database contains multiple incomplete relations, we may apply our techniques separately to each one. In addition, we may exploit correlations that hold across relations, by computing a primary-foreign key join when appropriate.

Input We start with a relation \( R \) as input. We assume that \( R \)’s tuples have been generated separately and independently by the same process. Importantly, we do not need to assume that this process follows a specific probabilistic model. Some of the tuples in \( R \) are complete, and some have one or more missing attribute values, and are hence incomplete. We do not assume that “how many” and “which” attribute values are missing follows a specific probabilistic model.

Output Given the relation \( R \), our approach ultimately produces a collection of predictions (estimations) of discrete probability distributions \( \Delta_t \), one for each incomplete tuple \( t \in R \). \( \Delta_t \) consists of the set of all possible combinations of values of the attributes missing in \( t \), each annotated with a probability, and with probabilities adding to 1. Each combination of values corresponds to a possible complete version of \( t \), and these combinations are mutually exclusive. Hence, as we discussed earlier, the output of our approach is a disjoint-independent probabilistic model associated with \( R \).

In order to estimate probability distributions over the missing values in each incomplete tuple in \( R \), we must carry out two steps. During the learning phase, an inference ensemble is built based on the complete portion of the data. During the inference phase, the ensemble is used to estimate probability distributions for incomplete tuples.

Learning phase Using the portion of \( R \) that consists of complete tuples, we build a model that is later used for inferring probabilities over the missing values in incomplete tuples. A separate model is built for each attribute \( a \) in \( R \), and captures an estimate of the conditional probability distribution of the values of \( a \), given values of some, or all, other attributes.

We begin the learning phase by mining frequent itemsets and association rules of attribute-value pairs [1] from the com-
complete tuples of \( R \). We then combine the mined association rules into meta-rules. (These are similar to sets of rule CPDs [21] and could be used to learn the parameters of an underlying Bayesian network if its topology were known.) Finally, we organize all the meta-rules corresponding to one attribute into a Meta-Rule Semi-Lattice (MRSL), which is ordered by subsumption. Fig. 2 presents an MRSL for the attribute \( age \). The top-level meta-rule, \( P(age) \), lists the frequencies of the values of \( age \) in the known portion of the dataset. Meta-rules at lower levels of the semi-lattice refine the estimates by progressively considering more evidence, e.g., the meta-rule \( P(age | edu = HS) \) refines the estimate of the CPD for the tuples in which \( edu = HS \). Meta-rules are annotated with weights that quantify the support of the meta-rule, i.e., the portion of the dataset from which the meta-rule was mined. Intuitively, meta-rules with a higher weight are supported by a larger portion of the dataset. Only rules that pass a specified support threshold are included in the MRSL. MRSLs are the result of the learning phase of our approach.

Inference phase Next, for each incomplete tuple \( t \in \mathcal{R} \) with a missing value for a single attribute \( a \), we use the MRSL corresponding to \( a \) to estimate the conditional probability distribution \( \Delta_{t,a} \) for the possible values of \( a \) in \( t \). Consider for example \( t_1 : \langle age = ?, edu = HS, inc = 50K, nw = 500K \rangle \). We identify five meta-rules in the MRSL in Fig. 2 that match \( t_1 : P(age), P(age | edu = HS), P(age | inc = 50K), P(age | nw = 500K), P(age | inc = 50K ∧ inc = 50K) \). When multiple meta-rules match, we use voting techniques to generate the estimate \( \Delta_{t,age} \). We may use all the meta-rules that match, or we may use the most specific applicable meta-rules, i.e., meta-rules that do not subsume any other meta-rules among the matches. In each case, each selected meta-rule contributes a vote to the final estimate, and we may combine the votes in different ways, e.g., with weighted voting or with averaging. Given its use of voting techniques for inference, the MRSL is an example of an inference ensemble [11]. The choice of which meta-rules to use, and which voting technique to employ, will have an impact on the accuracy of the estimated CPD. For example, for tuple \( t_1 \), averaging the predictions of all applicable meta-rules produces the CPD \((0.25, 0.51, 0.24)\), while weighted voting of the most specific meta-rules produces the CPD \((0.26, 0.48, 0.26)\). We will measure the impact of voter choice and voting method on accuracy in the evaluation.

Suppose now that we are given tuple \( t_2 : \langle age = ?, edu = ?, inc = 50K, nw = 500K \rangle \), and that we need to estimate the joint probability distribution over two attributes, \( age \) and \( edu \). One approach would be to estimate the CPDs for \( age \) and for \( edu \) separately, and then to compute \( P(age, edu | inc = 50K ∧ nw = 500K) = P(age | inc = 50K) \times P(edu | inc = 50K ∧ nw = 500K) \), but that would rely on independence assumptions that are not warranted. Instead, we use ordered Gibbs sampling [17] using both the MRSL for \( age \) and that for \( edu \) to produce an estimate of \( \Delta_t \).

We optimize the performance of inference based on the observation that many similar computations are performed by Gibbs sampling while computing \( \Delta_u \) and \( \Delta_v \) when \( u \) and \( v \) are related by subsumption. This suggests implementing multi-attribute inference holistically for the entire \( \mathcal{R} \), and caching the results of partial computations for re-use. The resulting optimization turns out to be very effective. This completes the inference phase of our approach and results in the desired disjoint-independent probabilistic model.

**Experimental evaluation** Although our approach does not need to make statistical assumptions about how each tuple is generated, and how missing values arose, such assumptions are needed for experimental evaluation. In order to judiciously evaluate the scalability and accuracy of our approach, we define an experimental benchmark that generates data according to Bayesian networks of various sizes and topologies. We treat the occurrences of missing values uniformly.

Our experimental framework takes as input the description of the topology of a Bayesian network, and generates an instance of the network by randomly selecting probability distributions for each random variable in accordance with the topology. Given a BN instance, we sample it to generate a set of complete tuples of specified size. The sample is then split into training and test. MRSL is learned from the training set. The test set is further processed, and one or more attribute values are replaced by a “?” in each tuple. Inference is then run over the test set, using the MRSL built on the training set. The accuracy of inferred probability distributions is evaluated by comparing them to the corresponding true probability distributions of the Bayesian network that generated the dataset.

**C. Overview of Contributions and Roadmap**

This paper makes the following contributions. We develop the Meta-Rule Semi-Lattice (MRSL), an inference ensemble used for deriving probabilistic databases starting from incomplete data (Section II). We demonstrate how MRSL can be learned from the known (complete) portion of the data (Section III), and how it can be used to estimate probability distributions for a single unknown attribute value (Section IV).

We develop a sampling-based inference approach used when multiple attribute values are unknown, and develop an optimization that greatly improves the performance of inference over entire databases (Section V). We provide an extensive experimental evaluation of the efficiency and effectiveness of our approach (Section VI), and show that it is both scalable
and that it makes accurate probability estimates. We describe related work in Section VII and conclude in Section VIII.

II. MODEL

This section defines the concepts used in the remainder of the paper. First, we define the notion of (in)complete tuples, then introduce meta-rules and meta-rule semi-lattices that will be used in the sequel to derive probability distributions for incomplete tuples.

**Database** We assume that a database consists of a single relation \( R \) with a set \( A \) of attributes, each with a corresponding domain of values. We limit our discussion to discrete finite-valued attributes, and propose to break up the domains of continuous attributes into sub-ranges, treating each sub-range as a discrete value. We distinguish two types of tuples in \( R \).

**Definition 2.1 (Incomplete Tuple):** An incomplete tuple \( t \) is an assignment of values to a subset of attributes \( A \subset A \). Missing attribute values are denoted with "?". We refer to \( A \) as the complete portion of \( t \).

**Definition 2.2 (Complete Tuple or Point):** A complete tuple (also called a point) is an assignment of values to all attributes in \( A \).

For example, in Fig. 1 \( t_1 \) : \( \langle \text{age} = 20, \text{edu} = \text{HS}, \text{inc} = \text{?}, \text{nw} = \text{?} \rangle \) is an incomplete tuple, and \( t_2 \) : \( \langle \text{age} = 20, \text{edu} = \text{BS}, \text{inc} = 50K, \text{nw} = 100K \rangle \) is a point (complete tuple).

**Meta-Rules** We start by introducing the notions of support, subsumption and association rules.

We view \( R \) as consisting of two disjoint subsets of tuples – the complete part \( R_c \), consisting of the points in \( R \), and the incomplete part \( R_i \), consisting of the incomplete tuples.

**Definition 2.3 (Support):** We say that point \( p \in R_c \) matches incomplete tuple \( t \in R_i \), if \( p \) and \( t \) agree on the values of attributes in the complete portion of \( t \). The support of \( t \) is the fraction of points in \( R_c \) that match \( t \). When \( R_c \) is known from context we denote this value by \( supp(t) \).

For example, in Fig. 1 point \( t_4 \) supports tuple \( t_1 \), while point \( t_2 \) does not. In fact, 3 out of 8 points in \( R_c \) \( \langle t_4, t_6, \) and \( t_7 \rangle \) support \( t_1 \), and so \( supp(t_1) = \frac{3}{8} \).

**Definition 2.4 (Subsumption):** Given incomplete tuples \( t_1 \) and \( t_2 \), we say that \( t_1 \) subsumes \( t_2 \) if the complete portion of \( t_1 \) is a proper subset of the complete portion of \( t_2 \), i.e., if \( t_1 \) assigns the same values to the attributes to which \( t_1 \) also assigns values, and also makes some additional value assignments. We denote this by \( t_2 \prec t_1 \).

To continue with our example, \( t_1 \prec t_5 \) and \( t_3 \prec t_5 \). No subsumption holds between \( t_1 \) and \( t_3 \).

We are now ready to define our notion of an association rule, from which our meta-rules are constructed.¹

**Definition 2.5 (Association Rule):** An association rule \( r \) is a pair of tuples \( \langle t_1, t_2 \rangle \), where \( t_1 \prec t_2 \). We refer to the set of attribute value assignments that are common to \( t_1 \) and \( t_2 \) as the body of the rule, denoted \( body(r) \). (In fact, \( body(r) \) is precisely the complete part of tuple \( t_2 \).) The set of attribute value assignments made by \( t_1 \) but not by \( t_2 \) is the head of the rule, denoted \( head(r) \). We also define confidence of an association rule \( conf(r) = \frac{supp(t_1)}{supp(t_2)} \).

Returning to our example, given \( t_3 \) and \( t_5 \), we may define an association rule \( r : \langle t_3, t_5 \rangle \) with \( body(r) = \{\text{age} = 20\} \) and \( head(r) = \{\text{inc} = 50K\} \). Now, we are interested in association rules with a single attribute value assignment in the head. Note that confidence estimates the conditional probability that the assignment in the head takes place, given the evidence in the body.

We next combine association rules that have the same body and assign different values to the same attribute in the head to form a meta-rule.

**Definition 2.6 (Meta-Rule):** A meta-rule \( m \) is a set of association rules \( \{r_1, \ldots, r_n\} \), where all \( r_i \) have the same attribute value assignments in the body, but assign different values to the same attribute in the head. We overload notation and use \( body(m) \) to denote the (common) attribute value assignments in the body of \( m \), and \( head(m) \) to denote the name of the attribute in the head (rather than an attribute value assignment, since the values are different).

An estimated CPD of \( m \), denoted \( \Delta(m) \), is an estimate of the conditional probability distribution over the entire domain of values for \( head(m) \), with \( body(m) \) given as evidence.

Consider tuples \( t_1, t_8, t_{11} \), and \( t_{14} \) in Fig. 1, and suppose that \( supp(t_1) = 0.06 \), \( supp(t_8) = 0.41 \), \( supp(t_{11}) = 0.29 \), and \( supp(t_{14}) = 0.06 \). The following subsumption relationships hold between the tuples: \( t_1 \prec t_8, t_1 \prec t_8, \) and \( t_{14} \prec t_8 \). Note that \( supp(t_8) = supp(t_1) + supp(t_{11}) + supp(t_{14}) \), because \( t_1, t_{11} \) and \( t_{14} \) agree on \( edu = \text{HS} \), and together enumerate all possible assignments of \( age \). We may use the four tuples to generate three association rules: \( r_1 : \langle t_1, t_8 \rangle, r_2 : \langle t_{11}, t_8 \rangle, \) and \( r_3 : \langle t_{14}, t_8 \rangle \). Noticing that the rules agree on the body and assign different values to the same attribute in the head, we combine them into the meta-rule \( m = \{r_1, r_2, r_3\} \), with \( head(m) = \text{age} \) and \( body(m) = \{\text{edu} = \text{HS}\} \). Finally, we compute \( \Delta(m) \) that estimates \( P(\text{age}|\text{edu} = \text{HS}) = \left[ \begin{array}{cccc} 0.06 & 0.29 & 0.06 & 0.41 \\ 0.41 & 0.70 & 0.15 & 0.15 \end{array} \right] \) (See Fig. 2). We also record the support of \( t_8 \) as the support of \( m \).

**Meta-Rule Semi-Lattices** Meta-rules are grouped together to form semi-lattices. These will be used to derive probability distributions for incomplete tuples. The notion of meta-rule subsumption is used to determine the shape of the semi-lattice.

**Definition 2.7 (Meta-Rule Subsumption):** Given meta-rules \( m_1 \) and \( m_2 \), we say that \( m_1 \) subsumes \( m_2 \) if \( m_1 \prec m_2 \), i.e., if \( m_1 \) assigns the same values to the attributes to which \( m_1 \) also assigns values, and also makes some additional value assignments. We denote this by \( m_2 \prec m_1 \).

The meta-rule semi-lattice is now defined as follows.

**Definition 2.8 (Meta-Rule Semi-Lattice):** For each attribute \( a \), its meta-rule semi-lattice, or \( MRSL_a \), is a partial order \( \langle M, \prec \rangle \), where \( M \) is the set of meta-rules with head attribute \( a \), and \( \prec \) is the meta-rule subsumption relationship.

An example of an MRSL that includes the meta-rules derived above is given in Fig. 2. Finally, we define the MRSL model.

**Definition 2.9 (MRSL Model):** Given a relation \( R \), an MRSL model is a set of MRSLs, one for each attribute in \( R \).
Algorithm 1 MRSL learning algorithm.

Require: Complete relation $R^c$, support threshold $\theta$, maxItemsets.

1: $MRSL = \emptyset$
2: $freqItemsets = ComputeFreqItemsets(\theta, maxItemsets)$
3: for $a \in Attributes(R^c)$ do
4:    $assocRules = ComputeAssocRules(a, freqItemsets)$
5:    $metaRules = ComputeMetaRules(assocRules)$
6:    $MRSL_a = ComputeSubsumption(metaRules)$
7: end for
8: return $MRSL$

III. LEARNING THE MODEL

The algorithmic approach of this paper uses frequent itemsets and association rules as basic building blocks, and we assume for ease of exposition that these are discovered from the complete portion of the data, $R^c$. In practice, the complete portion of incomplete tuples in $R^A$ may also be used to discover association rules. As is customary in frequent itemset mining (see, e.g., [1]), an itemset is recorded if its support passes a support threshold.

Algorithm 1 presents the MRSL learning algorithm. The algorithm takes a complete relation $R^c$ as input and outputs the $MRSL$ model, a set containing a meta-rule semi-lattice for each attribute in $R^c$ (Def. 2.9). Support threshold $\theta$ and an integer $maxItemsets$ are also supplied as arguments.

As the first step, in a call to $ComputeFreqItemsets$, frequent itemsets of attribute-value pairs are identified in the dataset. This procedure implements Apriori [1], a standard frequent itemset mining algorithm. However, the essence of our method is not dependent on which frequent itemset mining algorithm is used. Apriori is a bottom-up algorithm that starts with frequent 1-itemsets, and iteratively builds frequent itemsets of size $k$ by joining together appropriate pairs of itemsets of size $k-1$, and verifying that their frequency passes the support threshold. The algorithm terminates when no frequent itemsets are identified at a particular round $k$. Run-time of Apriori at round $k$ is quadratic in the number of frequent itemsets found at round $k - 1$. In order to control execution time, we modify the algorithm slightly by introducing another termination condition, namely, we stop after round $k$ if either no new frequent itemsets are found, or if more than $maxItemsets$ frequent itemsets are found at that round. We set $maxItemsets = 1000$ in our implementation. We empirically determined that this setting effectively controls model-building time, without a significant effect on accuracy.

Having identified frequent itemsets, the algorithm proceeds to build a meta-rule semi-lattice for each attribute $a$. First, in a call to $ComputeAssocRules$, association rules with $a$ as the head attribute are identified. We compute association rules irrespective of their confidence, i.e., there is no confidence threshold in our algorithm (see Def. 2.5).

Next, $ComputeMetaRules$ is invoked and identifies groups of association rules with the same attribute-value assignments in the body, and with different values assigned to the head attribute $a$. These association rules are grouped into a single meta-rule. Recall that each association rule is a pair of frequent itemsets (or incomplete tuples), and association rules are grouped into a meta-rule if they agree on the tuple that represents the body (see Def. 2.6 and the example that illustrates it). We record the support of the frequent itemset that corresponds to the body of the meta-rule as that meta-rule’s support. Fig. 2 lists these as weights above each meta-rule. For example, the weight of the meta-rule that computes $P(\text{age} | \text{edu}) = HS$ is $w = 0.41$, which is precisely the support of the frequent itemset $\text{edu} = HS$ in the dataset.

Importantly, for a particular assignment of attribute values in the body, not all values of the head attribute may be accounted for, because some frequent itemsets do not pass the support threshold $\theta$. As a result, supports of the association rules may not sum to 1. Our inference algorithms in Sections IV and V require that all probability distributions be positive, i.e., that each value in the domain of $a$ have a non-zero probability. To produce a valid positive CPD estimate for each meta-rule, we smooth and re-normalize each CPD, assigning a probability of at least 0.00001 to each value, and distributing any remaining probability mass equally among all values of $a$.

Finally, $ComputeSubsumption$ is invoked, and outputs an $MRSL$ according to meta-rule subsumption (Def. 2.7). An example of the $MRSL$ for attribute $\text{age}$ is given in Fig. 2.

We will experimentally evaluate the performance of Algorithm 1, and the size of the model in Section VI-B.

IV. INFERENCE FOR A SINGLE ATTRIBUTE

Algorithm 2 presents the inference procedure for tuples with a single missing value. The algorithm takes as input an incomplete tuple $t$ with a missing value for attribute $a$, $MRSL_a$, a voter selection mechanism $vChoice$ (with possible values all and best), and a voting scheme $vScheme$ (with possible values weighted and averaged). Other voter selection mechanisms and voting schemes exist, and we implement two reasonable options for each. The output $cpd$ is an estimate of the probability distribution over the values of $a$ in $t$.

First, $GetMatchingMetaRules$ is invoked, and returns the set of meta-rules from $MRSL_a$ that match tuple $t$, i.e., meta-rules in which the complete portion of the body makes the same attribute-value assignments as does $t$. If $vChoice = all$, all matching meta-rules are returned; if $vChoice = best$, only meta-rules that do not subsume any other matches are returned. Next, depending on whether the voting scheme is averaged or weighted, probability distributions of the voters are either averaged, position by position, or a weighted average is computed, with the support of the meta-rule serving as the
weight, and the computed CPD estimate is returned. We will experimentally evaluate the impact of different voter choices and voting schemes on run-time performance and accuracy of inference in Section VI-C.

V. INFERENCE FOR MULTIPLE ATTRIBUTES

In this section we describe a method for estimating the joint probability distribution for incomplete tuples with multiple missing values. We first give some background on Gibbs sampling and present the basic version of multi-attribute inference in our setting (Section V-A) that takes a single incomplete tuple as input. We then argue that multi-attribute inference may be optimized for a workload of incomplete tuples, and present the tuple-DAG optimization (Section V-B).

Suppose that we are interested in estimating the probability distribution over the missing values of \( \text{inc} \) and \( \text{nw} \) in incomplete tuple \( t_{12} \) (Fig. 1). One approach is to estimate the values of \( \text{inc} \) and \( \text{nw} \) independently, using the inference procedure of Section IV. However, since attributes are often non-independent in practice, this may result in an inaccurate estimate. Instead, we propose to use Gibbs sampling over MRSL models to estimate the joint probability distribution over the missing values simultaneously.

A. Gibbs Sampling

Gibbs sampling is commonly used to approximate the joint probability distribution over multiple variables when the conditional probability distribution over each variable is known. Our MRSL model described in earlier sections fits this application scenario exactly. A variant called ordered Gibbs sampling operates as follows. Start with a valid random assignment of attribute values. Then, repeatedly cycle through each attribute \( a \), resampling it in accordance with the MRSL \( a \) (i.e., sampling from the estimated CPD for \( a \), with all other attributes given as evidence). This procedure defines a Markov chain with a unique stationary joint distribution that can be reached from any initial state of the chain [17]. The first few tuples in the sampling are discarded, which is referred to as the burn-in period. The joint probability distribution can be estimated reliably after a sufficient number of iterations following the burn-in. The length of burn-in \( (B) \), and the subsequent number of iterations \( (N) \), may be estimated using standard techniques.

A Gibbs sampler converges to the true joint probability distribution if all local distributions are positive, i.e., they define transitions between all pairs of states, and if the local models are consistent, i.e., together they define a valid probabilistic space. As we will demonstrate in the experimental evaluation in Section VI-C, MRSL models make accurate predictions for any attribute — a good indication of consistency.

Sampling over the entire probabilistic space can be problematic. Suppose that we want to run inference for \( t_1 : \langle \text{age} = 20, \text{edu} = \text{HS}, \text{inc} = ?, \text{edu} = ? \rangle \). The support of the complete portion of \( t_1 \) in our dataset is 0.06, i.e., only 6% of the sampled points will be relevant to \( t_1 \). Assuming that \( N \) points that match \( t_1 \) are required to compute \( P(\text{inc}, \text{edu} | \text{age} = 20, \text{edu} = \text{HS}) \) reliably, we will have to draw \( \frac{N}{0.06} \) samples, wasting 94% of the samples. A standard way to address this is to only sample from the part of the space where \( \text{age} = 20 \) and \( \text{edu} = \text{HS} \). During sampling \( \text{age} \) and \( \text{edu} \) remain fixed, and the sampler cycles through the remaining attributes.

Sampling from the entire probabilistic space, which we call *all-at-a-time*, and sampling individually for each tuple, which we call *tuple-at-a-time*, are two possible approaches. There are also other options available. For example, suppose that we want to run inference for tuples \( t_1 \) and \( t_9 \) in Fig. 1. We leverage tuple subsumption (Def. 2.4) and observe that points generated when sampling *tuple-at-a-time* for \( t_8 \) may also be used to estimate the CPD for the missing values in \( t_1 \), because \( t_1 \prec t_8 \). In fact, *all-at-a-time* sampling may be viewed as sampling for the tuple \( t^* = \langle \text{age} = ?, \text{edu} = ?, \text{inc} = ?, \text{nw} = ? \rangle \), and samples generated in this way may be used to estimate the CPD for any tuple \( t \), since \( t \prec t^* \).

B. Workload-driven Sampling

Based on the observation that incomplete tuples may reuse samples from tuples that subsume them, we propose to build the tuple DAG, a data structure based on tuple subsumption (Def. 2.4). Fig. 3 shows the tuple DAG for a subset of the incomplete tuples in Fig. 1, and Algorithm 3 presents our workload-driven sampling algorithm.

The algorithm takes as input the MRSL model, a workload of tuples with multiple missing attributes \( \mathcal{R} \), and integers \( B \) and \( N \), representing the length of burn-in period and the target number of samples per tuple, respectively. The output of the algorithm is a probability distribution over the missing values for each tuple \( t \in \mathcal{R} \). The algorithm starts by computing a tuple DAG based on subsumption, and returns the set of roots — incomplete tuples that are not subsumed by any other tuples. Then the algorithm visits roots of the DAG in a round-robin fashion, and, if it encounters a root that has not been sampled for yet, takes \( B \) samples as part of burn-in, and discards those samples (lines 6-8). Next, a sample is taken and recorded (line 9), and then the number of samples accumulated by \( r \) so far is checked. When sample size reaches \( N \), sampling for \( r \) completes, its samples are propagated to its subsumees (line 15), and subsumees are promoted to root status when appropriate (lines 18-20). Note that when \( r \)'s samples are shared with its subsumee \( s \) (in ShareSamples), only samples that match \( s \) are recorded. The algorithm terminates when all tuples have accumulated \( N \) samples.

Note that the DAG in Fig. 3 only has two levels, because at most three attributes are missing per tuple in our running example. The DAG can be deeper in relations in which more attributes are potentially missing. However, we will demon-
strate experimentally in Section VI-D that the tuple DAG optimization results in significant performance improvements even for relations with few attributes.

VI. EXPERIMENTAL EVALUATION

Our experimental evaluation was performed using a Java prototype, with all processing done in memory. Experiments were carried out on Xeon X3450 quad-core 2.66GHz machines, with 4GB of RAM, running Linux (Fedora 12, 64-bit).

We next describe our experimental framework (Section VI-A), then present three sets of experiments. The first, described in Section VI-B, studies the properties of the MRSL model and shows that the model can be learned in reasonable time, even for large databases. The second set of experiments, described in Section VI-C, studies the accuracy and run-time performance of inference for a single missing attribute, showing that high accuracy is indeed achievable in realistic settings. The last set of experiments in Section VI-D studies the accuracy of sampling-based inference for multiple missing values per tuple and demonstrates the performance improvements achieved by our tuple-DAG optimization.

A. Experimental Framework

Our experimental framework allows us to execute repeatable experiments, while varying the parameters of the probabilistic model and of the dataset. In the experiments we assume the Bayesian network that generated the incomplete relation is known, allowing us to compare the inferred probability distribution with the true probability distribution of the BN.

Architecture of the Experimental Framework Our framework takes as input a description of the topology of a Bayesian network, specifying the number and names of random variables, along with a domain of values, and with a set of parents. First, the BN Instance Generator is invoked and instantiates network parameters by randomly populating conditional probability distributions over each variable given its parents. In all experiments, we randomly generate three network instances for each network topology, and average all results over these network instances. This allows us to control for variation in performance that is due to the particulars of the probability distributions.

For each network instance, the BN Sampler is invoked and uses forward sampling ([21] Sec. 12.1) to generate a dataset according to the specification of the network instance. The dataset is then split into training (90%) and test (10%) subsets. For each network instance, we execute three random splits of the dataset into training and test, and average all results across splits, thus reducing the variance in reported results. The training set is used for learning the MRSL model. The test set is further processed and one or several attributes in each tuple are replaces with “?”. Which attributes are replaced in a given tuple is chosen uniformly at random.

Experimental Dataset Our experimental evaluation uses 20 different Bayesian network topologies, divided into two sets. Properties of the networks are summarized in Table I. BN1 through BN7 have varying attribute cardinalities and topologies. BN8 through BN20 have regular topologies, and all attributes in a particular network have the same cardinality. Adopting common parameter settings from the literature on experiments on probabilistic database, our networks have between 4 and 10 attributes. Attribute cardinality ranges from 2 to 10; recall that when attribute domains are larger we group the values into a smaller number of buckets.

Note that, although our benchmark uses attributes of bounded cardinality, our results are applicable to attributes of high cardinality. The decisive parameter is not the cardinality of an individual attribute, but rather the size of the Cartesian product of domains of all unknown attributes (dom. size in Table I). Our benchmark assumes that the values of all attributes are potentially unknown, and accommodates the size of the Cartesian product of domains as high as 500,000. In a real-life setting this would typically be smaller, as only a handful of attributes would be missing.

Depth is the length of the longest path in the network graph, and is 0 when all attributes are independent. We use different subsets of the 20 networks in different experiments, depending on the trends that we wish to illustrate.

Measuring Accuracy To quantify the accuracy of inference, we compare the probability distributions predicted by MRSL to the true probability distributions of the Bayesian network, using Kullback-Leibler (KL) divergence [6], a non-symmetric measure of the difference between two probability distributions. When the distributions are close the absolute value of KL divergence is close to zero.

To further examine accuracy, we also compare the most probable value derived by MRSL to the true most probable, reported as percentage of correct top-1 guesses. This measure is naturally sensitive to the particulars of the probability distributions, since correct top-1 predictions are difficult to make when the probability distributions are nearly uniform.

Algorithm 3 Workload-driven sampling.

Require: MRSL model, R^i, # samples N, # burn-in samples B
1: roots = a set of tuples that have no parents in the DAG
2: completed = ∅
3: roots = ComputeTupleDAG(R^i)
4: while roots ≠ ∅ do
5: r = GetNext(roots)
6: if NotInitialized(r) then
7: DoSampleDiscard(MRSR, r, B) // run burn-in for r
8: if SampleSize(r) = N then
9: Remove(roots, r) // finished sampling for r
10: ComputeCPD(r)
11: for s ∈ GetSubsuminges(r) do
12: ShareSamples(r, s)
13: if SampleSize(s) = N then
14: Add(completed, s) // finished sampling for s
15: else if IsRoot(s, roots) then
16: Add(roots, s)
17: end if
18: end for
19: end if
20: end while
21: return completed
TABLE I
CHARACTERISTICS OF 20 BAYESIAN NETWORKS IN OUR EXPERIMENTS.

<table>
<thead>
<tr>
<th>network</th>
<th>num. attrs</th>
<th>avg card</th>
<th>dom. size</th>
<th>depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN1</td>
<td>4</td>
<td>4</td>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>BN2</td>
<td>5</td>
<td>4.4</td>
<td>1400</td>
<td>3</td>
</tr>
<tr>
<td>BN3</td>
<td>5</td>
<td>5.2</td>
<td>2400</td>
<td>3</td>
</tr>
<tr>
<td>BN4</td>
<td>5</td>
<td>5.2</td>
<td>2400</td>
<td>0</td>
</tr>
<tr>
<td>BN5</td>
<td>5</td>
<td>5.2</td>
<td>2400</td>
<td>2</td>
</tr>
<tr>
<td>BN6</td>
<td>10</td>
<td>2</td>
<td>1024</td>
<td>4</td>
</tr>
<tr>
<td>BN7</td>
<td>10</td>
<td>4</td>
<td>518,400</td>
<td>4</td>
</tr>
<tr>
<td>BN8</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>BN9</td>
<td>6</td>
<td>2</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>BN10</td>
<td>6</td>
<td>4</td>
<td>4096</td>
<td>2</td>
</tr>
<tr>
<td>BN11</td>
<td>6</td>
<td>6</td>
<td>46,656</td>
<td>2</td>
</tr>
<tr>
<td>BN12</td>
<td>6</td>
<td>8</td>
<td>262,144</td>
<td>2</td>
</tr>
<tr>
<td>BN13</td>
<td>6</td>
<td>2</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>BN14</td>
<td>6</td>
<td>4</td>
<td>4096</td>
<td>6</td>
</tr>
<tr>
<td>BN15</td>
<td>6</td>
<td>6</td>
<td>46,656</td>
<td>6</td>
</tr>
<tr>
<td>BN16</td>
<td>6</td>
<td>8</td>
<td>262,144</td>
<td>6</td>
</tr>
<tr>
<td>BN17</td>
<td>8</td>
<td>2</td>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>BN18</td>
<td>10</td>
<td>2</td>
<td>1024</td>
<td>2</td>
</tr>
<tr>
<td>BN19</td>
<td>10</td>
<td>2</td>
<td>1024</td>
<td>3</td>
</tr>
<tr>
<td>BN20</td>
<td>10</td>
<td>2</td>
<td>1024</td>
<td>5</td>
</tr>
</tbody>
</table>

B. Learning the MRSL Model

In this section we present our first set of experiments, which focus on learning the MRSL from the known portion of the data, as described in Section III. We study the influence of training set size and support threshold on two parameters – the time it takes to learn the MRSL model, in seconds, and the size of the resulting model, quantified as the total number of meta-rules. As we will see, even for large data sets, the learning time is very reasonable (under 10 min), implying that learning the MRSL from the data as part of an off-line process is feasible.

We experimented with 7 training set sizes, ranging from 1000 to 100,000 tuples, and with 5 values of support between 0.001 and 0.1. We considered 10 networks in this experiment. The networks had between 4 and 6 attributes, attribute cardinality from 2 to 8, and domain size from 16 to 262,144.

Fig. 4(a) presents the time taken to build the model as a function of training set size. The observation points represent averages across all ten networks, with support fixed at the median value of 0.02. Actual model building times varied between 0.01 and 6 sec for training size 1000 (avg 1.3 sec), and between 0.6 and 570 sec for training size 100,000 (avg 86 sec). We observe that model building time increases linearly with increasing training set size. We also considered the relationship between model size and training set size, and noticed that model size stays approximately constant. (Graph omitted due to space considerations.)

Fig. 4(b) and 4(c) show the relationship between support and model building time and model size, respectively. Results are averaged over ten networks, with training set size fixed at the median 10,000 tuples. We observe that values of both parameters decrease super-linearly with increasing support, with model size dropping particularly sharply. This is as expected, since fewer associations pass the support threshold as the threshold increases. Actual model building times ranged between 0.06 and 1.3 sec (avg 0.36 sec) for support 0.1, and between 0.07 and 230 sec (avg 29 sec) for support 0.001.

The same trends hold for individual networks as for averages in Fig. 4, with variations largely due to network size. We also observed that larger domain size usually resulted in longer model building times, albeit with some exceptions. The depth and width of the network did not seem to influence either the model building time or the resulting model size. The lack of a clear relationship between model building times and network properties may in part be due to the optimization that we use to control model building time, where we stop exploring higher-dimensional itemsets when the number of frequent itemsets reaches 1000 (see Section III).

C. Inference for a single missing attribute

In our second set of experiments we study the performance of inference over a single missing variable. Our results show that the MRSL model may be used to make accurate predictions of probability distributions in this case. Specifically, the best-averaged and best-weighted methods achieve highest accuracy when enough training data is available. As expected, larger training sets and lower support threshold values allow for higher accuracy. Interestingly, accuracy correlates with attribute cardinality and with network size, but not with network topology. Finally, single-attribute inference correlates with model size, and achieves reasonable run-time performance, even when the size of the MRSL is large.

In this experiment we used 14 networks. In the first part of this experiment, we consider how accuracy of inference depends on the size of the training set, on the support threshold, on the voting method being used, and on network characteristics. We experiment with five training set sizes, ranging from 1000 to 100,000 points, and with five support thresholds, ranging from 0.001 to 0.1.

Effect of Support, Training Size, and Voting on Accuracy We found, as expected, that larger training sets and lower values of support result in the most accurate model. Table II presents average accuracy of inference for the networks used in this experiment, for support of 0.001 and training set of size 100,000 (these settings give highest accuracy), for a variety of voting methods. The top-1 accuracy (% of correct guesses of the most likely value) is, for the most part, in agreement with KL – lower values of KL correlate with higher top-1 accuracy, and KL values of 0.1 and below typically lead to top-1 accuracy of over 90%.

We next explore the dependency between accuracy and training set size, for different voting methods. The left side of Fig. 5 plots average KL divergence as a function of training set size, for support = 0.001, which gives highest potential accuracy. Recall that lower KL values correspond to higher accuracy. We observe that KL divergence decreases as training set size increases to 5000 points, and then plateaus. We also
We observe no difference in accuracy among these networks, and conclude that topology does not directly influence accuracy of inference, at least not for the networks we considered.

Fig. 8(b) considers the effect of network size on accuracy for crown-shaped networks $BN_8$, $BN_9$, $BN_{17}$, and $BN_{18}$. The networks have similar topology and the same attribute cardinality, and differ only in the number of attributes (size). We observe that, for crown-shaped networks, accuracy depends on size, with smaller networks achieving higher accuracy.

Fig. 8(c) plots the effect of attribute cardinality on accuracy for line-shaped networks $BN_{13}$, $BN_{14}$, $BN_{15}$, and $BN_{16}$, each containing 6 attributes with cardinality varying from 2 to 8. We observe that, for these networks, accuracy of inference depends on attribute cardinality, with lower cardinality corresponding to higher accuracy.
Run-time Performance  As a final experiment in this section, we consider the run-time performance of single-attribute inference. In our experiments we determined that the choice of a voting method has no measurable effect on inference time. We report average inference for \textit{support} = 0.001, for a variety of test set sizes. Fig. 9 plots inference time as a function of model size, for batches of 500, 1000, 5000, and 10,000 tuples. Each point on the charts represents inference time for an entire batch. To ease understanding of the results, we also plot regression lines. We notice that inference time scales linearly with the size of the model. When the MRSL model is of size 10,000 or less, as is the case for the majority of the networks in our experiments, it takes 0.153 ms to run inference for a single missing attribute per tuple. For larger models, which we typically see for BN7, BN12, and BN16, it takes 1.539 ms to run such inference, on average.

D. Sampling-Based Inference

In this last set of experiments we consider the accuracy and efficiency of inference for tuples with multiple missing attribute values. Recall that our approach, described in Section V, is based on (optimized) Gibbs sampling. Our experiments show that this achieves high prediction accuracy for multi-variable inference with MRSL, particularly when individual MRSL models are highly accurate. In those cases about 2000 sampling points per tuple are needed, and prediction accuracy increases with increasing number of samples. The experiments further confirm that our \textit{tuple-DAG} optimization technique is extremely effective, reducing the sampling runtime by close to an order of magnitude. We compared the accuracy of \textit{tuple-DAG} to \textit{tuple-at-a-time}, and, as expected, found no difference for any of the networks (plot omitted).

We consider 10 networks with 4 to 8 attributes, cardinality between 2 and 5.2, and domain size between 16 and 4096.

Accuracy  The accuracy of multi-attribute inference varied for different networks, and we observed multiple trends. For this reason, rather than presenting averages across networks as we did in previous sections, we present results on a case-by-case basis. For the majority of the networks (7 out of 10), very high prediction accuracy was achieved, with KL divergence values of 0.1 or below. The left side of Fig. 10 plots average KL divergence for BN8, for a varying number of missing attributes. As expected, prediction accuracy increases (KL divergence decreases) as the number of samples per tuple increases. Prediction accuracy is higher for tuples with fewer missing values, also as expected.
For some networks, KL divergence remained above 0.1 for the experimental parameters we considered. The middle of Fig. 10 plots average KL divergence for BN$_{17}$, for up to five missing attributes per tuple. We observe similar trends as for BN$_8$, but prediction accuracy is in general lower. This is to be expected, since BN$_{17}$ is a much larger network, and prediction accuracy of individual MRSL models for BN$_{17}$ is lower than for BN$_8$ (see Table II). Nonetheless, KL values of 0.5 (2000 samples/tuple, 5 missing attributes), translate into about 40% correct top-1 guesses, as compared to 3% for random guessing.

Finally, the right side of Fig. 10 plots average KL for BN$_{17}$. We do not observe the trend seen for BN$_8$ and BN$_{17}$, where accuracy improved with increasing number of points per tuple, and was lower for tuples with more missing values. We plan to investigate reasons for this in the future, but remark that KL values of about 0.26 (2000 samples per tuple, tuples with 3 and 4 missing values) translate to top-1 accuracy of 38%, compared to under 2% for random guessing.

Run-time Performance In our experiment we explore the run-time performance of Gibbs sampling for workloads of incomplete tuples, with and without the tuple-DAG optimization (see Section V). The experiment in this section considers a workload of incomplete tuples with a varying number of missing values. For each network, at most networkSize − 1 attributes were missing, where networkSize is the total number of attributes per network. We present performance for the case where 500 points are sampled per incomplete tuple from MRSL. We measured performance with other sample sizes, and observed the same trends.

Sampling cost is a function of workload size — the number of distinct incomplete tuples in the test set. To quantify sampling cost we measure wall-clock time of inference and sample size — the total number of sampled points. The left side of Fig. 11 plots sample size as a function of workload size. Each dot corresponds to a single observation in a single network. The choice of a network has no bearing on sampling cost, and we plot all observations on a single graph. We observe that sample size increases linearly with increasing workload size, and that tuple-DAG clearly outperforms the tuple-at-a-time baseline, and increases with a much lower slope. The left side of Fig. 11 plots wall-clock time of inference as a function of sample size. We observe that wall-clock time increases linearly with workload size, justifying our choice to minimize the number of sampled points per workload. Here again tuple-DAG significantly outperforms the baseline in all cases.

Fig. 9. Inference time as a function of model size, with support=0.001.

VII. RELATED WORK

In our work we use an ensemble method for inference. Ensemble methods are widely used for a variety of machine learning tasks [11] and in many application domains. To the best of our knowledge, this is the first work in which an ensemble is used for inference in the context of probabilistic databases. Our approach builds on dependency networks proposed in [17]. Our approach differs in that we propose an inference ensemble, and develop a performance optimization of sampling for workloads of tuples.

Our work is also related to mining approximate and conditional functional dependencies (e.g., [13], [15], [18], [19]) and to data cleaning (e.g., [4], [5], [23]). However, while the goal of that work is largely on detecting correlations among attributes, and on predicting the most probable set of attribute values, our goal is to estimate probability distributions and to do so efficiently. Our work is particularly related to ERACER [23], that builds on relational dependency networks and models correlations between attribute values within a tuple and across tuples. Similarly to our approach, the model is derived from a collection of local CPDs, and so is approximate. Another similarity is the averaging of votes at the level of local CPDs. In contrast to our approach, ERACER focuses on prediction accuracy, while we consider, and quantify, both accuracy and run-time performance of learning and inference, particularly with multiple missing values. A thorough comparison with their method is in our immediate plans.

There has been much work on probabilistic databases that assumes that a probabilistic model and the associated probability values are known. An approach for deriving probability values based on input from domain experts is proposed in [10]. Another approach, partially based on learning, is found in [9]. Here a probabilistic model—combinations of time-varying multivariate Gaussians—is postulated, and its parameters are learned from historical data. This seems to work well for sensor network data. Concerns about how the probability distributions are derived are also addressed in [20] and [27].

VIII. CONCLUSION AND FUTURE WORK

This paper proposes a novel framework for deriving probability distributions for incomplete databases. Our solution is based on inference ensembles, called meta-rule semi-lattices, that are learned from the complete portion of the data. We develop a mechanism for selecting a subset of the available meta-rules to infer probability distributions for a single missing
attribute. We propose an optimized inference algorithm based on Gibbs sampling for predicting the probability distribution over multiple missing values. A thorough experimental study demonstrates the accuracy and efficiency of our approach.

In the future we plan to evaluate our approach on real-world datasets. We will also compare the performance of our method with other methods that are based on local CPD estimates. Query optimization for the probabilistic databases generated by our framework is another intriguing problem that we intend to study in the future. In particular, our approach opens new possibilities for partial materialization of probability values, as well as for lazy, query-targeted learning and inference.

IX. ACKNOWLEDGMENTS

We would like to thank Ben Taskar for his valuable input.

This work was supported in part by the US National Science Foundation grant 0937060 to the Computing Research Association for the CIFellows Project, and by grants IIS-0803524 and IIS-0629846, by the Israel Science Foundation, and by the US-Israel Binational Science Foundation.

REFERENCES


In SIGMOD, 2008.