Selectivity Estimation using Probabilistic Models

Lise Getoor
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
getoor@cs.stanford.edu

Ben Taskar
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
btaskar@cs.stanford.edu

Daphne Koller
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
koller@cs.stanford.edu

ABSTRACT

Estimating the result size of complex queries that involve selection on multiple attributes and the join of several relations is a difficult but fundamental task in database query processing. It arises in cost-based query optimization, query profiling, and approximate query answering. In this paper, we show how probabilistic graphical models can be effectively used for this task as an accurate and compact approximation of the joint frequency distribution of multiple attributes across multiple relations. *Probabilistic Relational Models (PRMs)* are a recent development that extends graphical statistical models such as Bayesian Networks to relational domains. They represent the statistical dependencies between attributes within a table, and between attributes across foreign-key joins. We provide an efficient algorithm for constructing a PRM from a database, and show how a PRM can be used to compute selectivity estimates for a broad class of queries. One of the major contributions of this work is a unified framework for the estimation of queries involving both select and foreign-key join operations. Furthermore, our approach is not limited to answering a small set of predetermined queries; a single model can be used to effectively estimate the sizes of a wide collection of potential queries across multiple tables. We present results for our approach on several real-world databases. For both single-table multi-attribute queries and a general class of select-join queries, our approach produces more accurate estimates than standard approaches to selectivity estimation, using comparable space and time.

1. INTRODUCTION

Accurate estimates of the result size of queries are crucial to several query processing components of a database management system (DBMS). Cost-based query optimizers use intermediate result size estimates to choose the optimal query execution plan. Query profilers provide feedback to a DBMS user during the query design phase by predicting resource consumption and distribution of query results. Precise selectivity estimates also allow efficient load balancing for parallel join on multiprocessor systems. Selectivity estimates can also be used to approximately answer counting (aggregation) queries.

The result size of a selection query over multiple attributes is determined by the joint frequency distribution of the values of these attributes. The joint distribution encodes the frequencies of all combinations of attribute values, so representing it exactly becomes infeasible as the number of attributes and values increases. Most commercial systems approximate the joint distribution by adopting several key assumptions; these assumptions allow fast computation of selectivity estimates, but, as many have noted, the estimates can be very inaccurate.

The first common assumption is the *attribute value independence assumption*, under which the distributions of individual attributes are independent of each other and the joint distribution is the product of single-attribute distributions. However, real data often contain strong correlations between attributes that violate this assumption, leading to very inaccurate approximations. For example a census database might contain highly correlated attributes such as *Income* and *Home-Owner*. The attribute value independence assumption would lead to an overestimate of the result size of a query that asks for low-income home-owners.

A second common assumption is the *join uniformity* assumption, which states that a tuple from one relation is equally likely to join with any tuple from the second relation. Again, there are many situations in which this assumption is violated. For example, assume that our census database has a second table for online purchases. High-income individuals typically make more online purchases than average. Therefore, a tuple in the purchases table is more likely to join with a tuple of a high-income individual, thereby violating the join uniformity assumption. If we consider a query for purchases by high-income individuals, an estimation procedure that makes the join uniformity assumption is likely to substantially underestimate the query size.

To relax these assumptions, we need a more refined approach, that takes into consideration the *joint distribution* over multiple attributes, rather than the distributions over the attributes in isolation. Several approaches to joint distribution approximation, also referred to as *data reduction*, have been proposed recently; see [2] for an excellent summary of this area. Most of this work has focused on the task of estimating the selectivity of a select operation — the *select selectivity* — within a single table. One simple approach for approximating the query size is via random sampling. Here, a set of samples is generated, and then the query result size is estimated by computing the actual query result size relative to the sampled data. However, the amount of data required for accurate estimation can be quite large. More recently, several approaches have been proposed that attempt to capture the joint distribution over attributes more directly. The earliest of these is the multidimensional histogram approach of Poosala and Ioannidis [23, 25]. They provide an extensive exploration of the taxonomy of methods for constructing multidimensional histograms and study the effectiveness of different techniques. They also propose an approach based on *singular value decomposition*, applicable only in the two-

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

dimensional case. A newer approach is the use of wavelets to approximate the underlying joint distribution [21, 27, 6].

Much less work has been done on estimating the selectivity of joins. Commercial DBMSs commonly make the uniform join assumption. One approach that has been suggested is based on random sampling: randomly sample the two tables, and compute their join. This approach is flawed in several ways [1], and some work has been devoted to alternative approaches that generate samples in a more targeted way [20]. An alternative recent approach is the work of Acharya *et al.* [1] on join synopses, which maintains statistics for a few distinguished joins. To our knowledge, no work has been done on approaches that support selectivity estimation for queries containing both select and join operations in real-world domains.

In this paper, we propose an alternative approach for the selectivity estimation problem, based on techniques from the area of *probabilistic graphical models* [17, 24]. As we will show, our approach has several important advantages. First, it provides a uniform framework for select selectivity estimation and foreign-key join selectivity estimation, introducing a systematic method for estimating the size of queries involving both operators. Second, our approach is not limited to answering a small set of predetermined queries; a single statistical model can be used to effectively estimate the sizes of any (select foreign-key join) query, over any set of tables and attributes in the database.

Probabilistic graphical models are a language for compactly representing complex joint distributions over high-dimensional spaces. They are based on a graphical notation that encodes *conditional independence* between attributes in the distribution. Conditional independence arises when two attributes are correlated, but the interaction is mediated via one or more other variables. For example, in a census database, education is correlated with income, and home-owner status is correlated with income. Hence, education is correlated with home-owner status, but only indirectly via the income level. Interactions of this type are extremely common in real domains. Probabilistic graphical models exploit the conditional independencies that exist in a domain, and thereby allow us to specify joint distributions over high dimensional spaces compactly.

In this paper, we provide a framework for using probabilistic graphical models to estimate selectivity of queries in a relational database. As we show, *Bayesian networks (BNs)* [24] can be used to represent the interactions between attributes in a single table, providing high-quality estimates of the joint distribution over the attributes in that table. *Probabilistic relational models (PRMs)* [18][1] extend Bayesian networks to the relational setting. As we show, PRMs allow us to represent skew in the join probabilities between tables, as well as correlations between attributes of tuples joined via a foreign-key. They thereby allow us to estimate selectivity of queries involving both selects and joins over multiple tables.

Like most selectivity estimation algorithms, our algorithm consists of two phases. The *offline* phase, in which the PRM is constructed from the database. This process is automatic, based solely on the data and the space allocated to the statistical model. The second, *online* phase, is the selectivity estimation for a particular query. The selectivity estimator receives as input a query and a PRM, and outputs an estimate for the result size of the query. Note that the same PRM is used to estimate the size of a query over any subset of the attributes in the database; we are not required to have prior information about the query workload.

[1]The term "probabilistic relational models" has two very distinct meanings in two different communities: the probabilistic modeling community [11] and the database community [14, 10]. We use the term in its former sense.

Throughout this paper, we make two important assumptions. The first is that foreign keys obey referential integrity. This assumption about the database is required in the construction of the PRM. The second is that all joins are equality joins between a foreign key and a primary key. This assumption is made purely for ease of presentation. While queries with foreign-key joins stand to benefit most from the probabilistic models that we propose, our methods are not limited to dealing with these queries and we describe how to extend our approach to a broader class of joins in Section 6.

The remainder of the paper is structured as follows. In Section 2, we consider the case of selectivity estimation for select operations over a single table. We define Bayesian networks, and show how they can be used to approximate the joint distribution over the entire set of attributes in the table. In Section 3, we move to the more complex case of queries over multiple tables. We present the PRM framework, which generalizes Bayesian networks to the relational case, and show how we can use PRMs to accomplish both select and join selectivity estimation in a single framework. In Section 4, we present an algorithm for automatically constructing a PRM from a relational database. In Section 5, we provide empirical validation of our approach, and compare it to some of the most common existing approaches. We present experiments over several real-world domains, showing that our approach provides much higher accuracy (in a given amount of space) than previous approaches, at a very reasonable computational cost, both offline and online.

2. ESTIMATION FOR SINGLE TABLES

We first consider estimating the result size for select queries over a single relation. For most of this section, we restrict attention to domains where the number of values for each attribute is relatively small (up to about 50), and to queries with equality predicates of the form *attribute* = *value*. Neither of these restrictions is a fundamental limitation of our approach; at the end of this section, we discuss how our approach can be applied to domains with larger attribute value spaces and to range queries.

Let R be some table; we use $R.*$ to denote the value (non-key) attributes $A_1, \ldots, A_n$ of R. We denote the joint frequency distribution over $A_1, \ldots, A_n$ as $F_{\mathcal{D}}(A_1, \ldots, A_n)$. It is convenient to deal with the normalized frequency distribution $P_{\mathcal{D}}(A_1, \ldots, A_n)$ where:

$$P_{\mathcal{D}}(A_1, \ldots, A_n) = F_{\mathcal{D}}(A_1, \ldots, A_n)/|R|.$$

This transformation allows us to treat $P_{\mathcal{D}}(A_1, \ldots, A_k)$ as a probability distribution. We can also view this joint distribution as generated by an imaginary process, where we sample a tuple r from R, and then select as the values of $A_1, \ldots, A_n$ the values of $r.A_1, \ldots, r.A_n$. (Note that we are not suggesting that the sampling process used to define $P_{\mathcal{D}}$ be carried out in practice. We are merely using it as a way of defining $P_{\mathcal{D}}$.)

Now, consider a query Q over a set of attributes $A_1, \ldots, A_k \subseteq R.*$, which is a conjunction of selections of the form $A_i = v_i$. Let $\mathcal{I}_Q$ be the event that the equalities in Q hold for r. It is clear that the size of the result of the query Q is:

$$\mathit{size}_Q[\mathcal{D}] = F_{\mathcal{D}}(Q) = |R| \cdot P_{\mathcal{D}}(\mathcal{I}_Q), \tag{1}$$

where $F_{\mathcal{D}}(Q)$ is the number of tuples satisfying Q and $P_{\mathcal{D}}(\mathcal{I}_Q)$ is the probability, relative to $\mathcal{D}$, of the event $\mathcal{I}_Q$. To simplify notation, we will often simply use $P_{\mathcal{D}}(Q)$. As the size of the relation is known, the joint probability distribution contains all the necessary information for query size estimation. Hence, we focus attention on the joint probability distribution.

Unfortunately, the number of entries in this joint distribution grows exponentially in the number of attributes, so that explicitly representing the joint distribution $P_{\mathcal{D}}$ is almost always intractable. Several approaches have been proposed to circumvent this issue by approximating the joint distribution (or projections of it) using a more compact structure [25, 21]. We also propose the use of statistical models that approximate the full joint distribution. However in order to represent the distribution in a compact manner, we exploit the *conditional independence* that often holds in a joint distribution over real-world data. By decomposing the representation of a joint distribution into factors that capture the independencies that hold in the domain, we get a compact representation for the distribution.

2.1 Conditional Independence

Consider a simple relation R with the following three value attributes, each with its value domain shown in parentheses: *Education* (*high-school*, *college*, *advanced-degree*), *Income* (*low*, *medium*, *high*), and *Home-Owner* (*false*, *true*). As shorthand, we will use the first letter in each of these names to denote the attribute, using capital letters for the attributes and lower case letters for particular values of the attributes. We use $P(A)$ to denote a probability distribution over the possible values of attribute A, and $P(a)$ to denote the probability of the event $A = a$.

Assume that the joint distribution of attribute values in a database is as shown in Fig. 1(a). Using this joint distribution, we can compute the selectivity of any query over E, I, and H. As shorthand, we will use Q_{eih} to denote a select query of the form $E = e, I = i, H = h$. Then $size_{Q_{eih}}[\mathcal{D}] = |R| \cdot P_{\mathcal{D}}(e, i, h)$. However, to explicitly represent the joint distribution we need to store 18 numbers, one for each possible combination of values for the attributes. (In fact, we can get away with 17 numbers because we know that the entries in the joint distribution must sum to 1.)

In many cases, however, our data will exhibit a certain structure that allows us to (approximately) represent the distribution using a much more compact form. The intuition is that some of the correlations between attributes might be indirect ones, mediated by other attributes. For example, the effect of education on owning a home might be mediated by income: a high-school dropout who owns a successful Internet startup is more likely to own a home than a highly educated beach bum — the income is the dominant factor, not the education. This assertion is formalized by the statement that *Home-owner* is *conditionally independent* of *Education* given *Income*, i.e., for every combinations of values h, e, i, we have that:

$$P(H = h \mid E = e, I = i) = P(H = h \mid I = i).$$

This assumption holds for the distribution of Fig. 1.

The conditional independence assumption allows us to represent the joint distribution more compactly in a factored form. Rather than representing $P(E, I, H)$, we will represent: the marginal distribution over *Education* — $P(E)$; a conditional distribution of *Income* given *Education* —$P(I \mid E)$; and a conditional distribution of *Home-owner* given *Income* — $P(H \mid I)$. It is easy to verify that this representation contains all of the information in the original joint distribution, *if the conditional independence assumption holds*:

$$\begin{aligned} P(H, E, I) &= P(E)P(I \mid E)P(H \mid I, E) \\ &= P(E)P(I \mid E)P(H \mid I) \end{aligned} \quad (2)$$

where the last equality follows from the conditional independence of E and H given I. In our example, the joint distribution can be represented using the three tables shown in Fig. 1(b). It is easy to verify that they do encode precisely the same joint distribution as in Fig. 1(a).

The storage requirement for the factored representation seems to be $3 + 9 + 6 = 18$, as before. In fact, if we account for the fact that some of the parameters are redundant because the numbers must add up to 1, we get $2 + 6 + 3 = 11$, as compared to the 17 we had in the full joint. While the savings in this case may not seem particularly impressive, the savings grow exponentially as the number of attributes increases, as long as the number of direct dependencies remains bounded.

Note that the conditional independence assumption is very different from the standard attribute independence assumption. In this case, for example, the one-dimensional histograms (i.e., marginal distributions) for the three attributes are shown in Fig. 1(c). It is easy to see that the joint distribution that we would obtain from the attribute independence assumption in this case is very different from the true underlying joint distribution. It is also important to note that our conditional independence assumption is compatible with the strong correlation that exists between *Home-owner* and *Education* in this distribution. Thus, conditional independence is a much weaker and more flexible assumption than standard (marginal) independence.

2.2 Bayesian Networks

Bayesian networks [24] are compact graphical representations for high-dimensional joint distributions. They exploit the the underlying structure of the domain — the fact that only a few aspects of the domain affect each other directly. We define our probability space as the set of possible assignments to the set of attributes $A_1, \ldots, A_n$ of a relation R. BNs can compactly represent a joint distribution over $A_1, \ldots, A_n$ by utilizing a structure that captures conditional independences among attributes, thereby taking advantage of the "locality" of probabilistic influences.

A Bayesian network $\mathcal{B}$ consists of two components. The first component, G, is a directed acyclic graph whose nodes correspond to the attributes $A_1, \ldots, A_n$. The edges in the graph denote a direct dependence of an attribute A_i on its parents $\text{Parents}(A_i)$. The graphical structure encodes a set of conditional independence assumptions: each node A_i is conditionally independent of its nondescendants given its parents.

Fig. 2(a) shows a Bayesian network constructed (automatically) from data obtained from the 1993 Current Population Survey of U.S. Census Bureau using their Data Extraction System [5]. In this case, the table contains 12 attributes: *Age*, *Worker-Class*, *Education*, *Marital-Status*, *Industry*, *Race*, *Sex*, *Child-Support*, *Earner*, *Children*, *Income*, and *Employment-Type*. The domain sizes for the attributes are, respectively: 18, 9, 17, 7, 24, 5, 2, 3, 3, 42, and 4. We see, for example, that the *Children* attribute (representing whether or not there are children in the household) depends on other attributes only via the attributes *Income*, *Age*, and *Marital-Status*. Thus, *Children* is conditionally independent of all other attributes given *Income*, *Age*, and *Marital-Status*.

The second component of a BN describes the statistical relationship between each node and its parents. It consists of a *conditional probability distribution (CPD)* $P_{\mathcal{B}}(A_i \mid \text{Parents}(A_i))$ for each attribute, which specifies the distribution over the values of A_i given any possible assignment of values to its parents. This CPD may be represented in a number of ways. It may be represented as a table, as in our earlier example. Alternatively, it can be represented as a tree, where the interior vertices represent splits on the value of some parent of A_i, and the leaves contain distributions over the values of A_i. In this representation, we find the conditional distribution over A_i given a particular choice of values $A_{k_1} = a_1, \ldots, A_{k_\ell} = a_\ell$ for its parents by following the appropriate path in the tree down to a leaf: When we encounter a split on some variable A_{k_j}, we go down the branch corresponding to the value of a_j; we then use the distribution stored at that

E	I	H	$P(E,I,H)$
h	l	f	0.27
h	l	t	0.03
h	m	f	0.105
h	m	t	0.045
h	h	f	0.005
h	h	t	0.045
c	l	f	0.135
c	l	t	0.015
c	m	f	0.063
c	m	t	0.027
c	h	f	0.006
c	h	t	0.054
a	l	f	0.018
a	l	t	0.002
a	m	f	0.042
a	m	t	0.018
a	h	f	0.012
a	h	t	0.108

(a)

E	$P(E)$
h	0.5
c	0.3
a	0.2

I	E	$P(I \mid E)$
l	h	0.6
m	h	0.3
h	h	0.1
l	c	0.5
m	c	0.3
h	c	0.2
l	a	0.1
m	a	0.3
h	a	0.6

H	I	$P(H \mid I)$
t	l	0.1
f	l	0.9
t	m	0.3
f	m	0.7
t	h	0.9
f	h	0.1

(b)

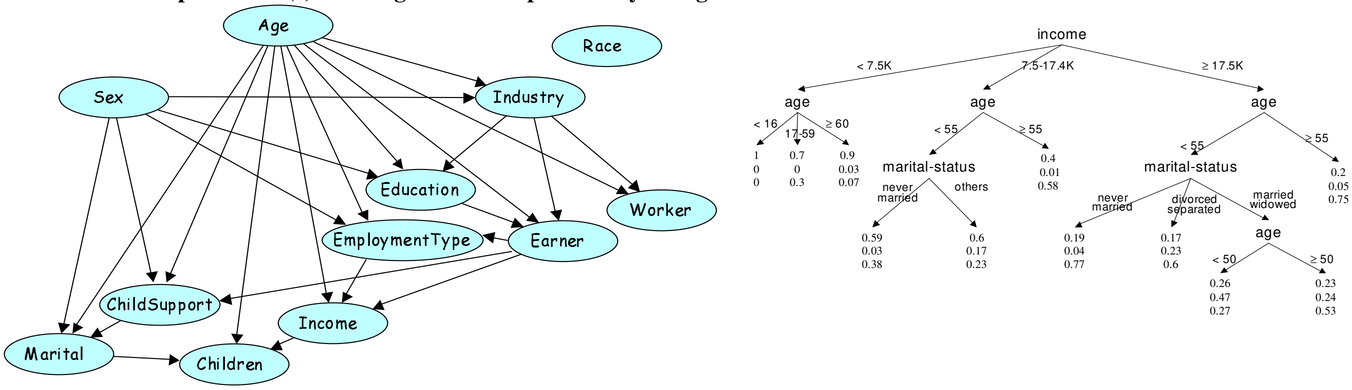
E	$P(E)$
h	0.5
c	0.3
a	0.2

I	$P(I)$
l	0.47
m	0.30
h	0.23

H	$P(H)$
t	0.344
f	0.656

(c)

Figure 1: (a) The joint probability distribution for a simple example. (b) A representation of the joint distribution that exploits conditional independence. (c) The single-attribute probability histograms.



(a) (b)

Figure 2: (a) A Bayesian network for the census domain. (b) A tree-structured CPD for the *Children* node given its parents *Income*, *Age* and *Marital-Status*.

leaf. The CPD tree for the *Children* attribute in the network of Fig. 2(a) is shown in Fig. 2(b). The possible values for this attribute are *N/A*, *Yes* and *No*. We can see, for example, that the distribution over *Children* given *Income* $\geq 17.5K$, *Age* < 55, and *Marital-Status* $=$ *never-married* is $(0.19, 0.04, 0.77)$; the distribution given *Income* $\geq 17.5K$, *Age* < 50, and *Marital-Status* $=$ *married* is $(0.26, 0.47, 0.27)$, as is the distribution given *Income* $\geq 17.5K$, *Age* < 50, and *Marital-Status* $=$ *widowed*: the two instantiations lead to the same induced path down the tree.

The conditional independence assumptions associated with the BN $\mathcal{B}$, together with the CPDs associated with the nodes, uniquely determine a joint probability distribution over the attributes via the *chain rule*:

$$P_{\mathcal{B}}(A_1, \ldots, A_n) = \prod_{i=1}^{n} P_{\mathcal{B}}(A_i \mid \text{Parents}(A_i)).$$

(This formula is precisely analogous to the one we used in Eq. (2) for our simple example from Section 2.1.) Thus, from our compact model, we can recover the joint distribution; we do not need to represent it explicitly. In our example above, the number of entries in the full joint distribution is approximately 7 billion, while the number of parameters in our BN is 951—a significant reduction!

2.3 BNs for Query Estimation

Our conditional independence assertions correspond to equality constraints on the joint distribution in the database table. In general, of course, these equalities will rarely hold exactly. In fact, even if the data was generated by independently generating random samples from a distribution that satisfies the conditional independence (or even unconditional independence) assumptions, the distribution derived from the frequencies in our data will not satisfy these assumptions. However, in many cases we can *approximate* the distribution very well using a Bayesian network with the appropriate structure. We defer a longer discussion of this issue to Section 4.

A Bayesian network is a compact representation of a full joint distribution. Hence, it implicitly contains the answer to any query about the probability of any assignment of values to a set of attributes. Thus, if we construct a BN $\mathcal{B}$ that approximates $P_{\mathcal{D}}$, we can easily use it to estimate $P_{\mathcal{D}}(Q)$ for any query Q over R. Assume that our query Q has the form $r.\mathbf{A} = \mathbf{a}$ (here we abbreviate a multidimensional select using vector notation). Then we can compute

$$F_{\mathcal{D}}(\mathbf{A} = \mathbf{a}) = |R| \cdot P_{\mathcal{D}}(Q) \approx |R| \cdot P_{\mathcal{B}}(\mathbf{A} = \mathbf{a})$$

Of course, generating the full joint distribution $P_{\mathcal{B}}$ can be computationally very expensive, and is almost always infeasible in the runtime setting in which query size is typically estimated. Thus, we need a more efficient algorithm for computing $P_{\mathcal{B}}(\mathbf{A} = \mathbf{a})$. Although the problem of computing this probability is NP-hard in the worst case, BN inference is typically very efficient for network structures encountered in practice. The standard BN inference algorithms [19] use special-purpose graph-based algorithms that exploit the graphical structure of the network. The complexity of these algorithms depends on certain natural parameters relating to the connectivity of the graph. These parameters are typically small for most real-world models, allowing very effective inference for many networks with hundreds of nodes or more [16, 26].

At the start of this section, we made several assumptions. We now describe how to relax these assumptions. First, we made the assumption that our select operations used only equality predicates. It is straightforward to extend the techniques that we have just described for computing the probability of an assignment of values to a set of attributes to handle range queries by computing the probability that an assignment of values to the attributes falls in that range. We simply sum over all potential value assignments satisfying the range constraints. While at first glance, this may sound quite expensive, the BN inference algorithms described above can be easily adapted to compute these values without any increase in computational complexity. The second important assumption that we have been making is that the domains for the attributes are small to moderately sized. We can lift this restriction on our BNs by using techniques that have been developed for discretization of domain values [12, 22]. In cases where domain values are not ordinal, we can use feature hierarchies if they are available [9] or we may use any of a number of clustering algorithms. Once we have built a BN over the discretized or abstracted attribute value space, we must now modify our query estimation techniques to provide estimates for queries over the base level values. One method for doing this is to simply compute the selectivity estimate for an abstract query, which maps the base level values to their appropriate discretized or abstracted value, and to then compute an estimate for the base level query by assuming a uniform distribution within the result.

3. JOIN SELECTIVITY ESTIMATION

In the previous section, we restricted attention to queries over a single table. In this section, we extend our approach to deal with queries over multiple tables. We restrict attention to databases satisfying *referential integrity*: Let R be a table, and let F be a foreign key in R that refers to some table S with primary key K; then for every tuple $r \in R$ there must be some tuple $s \in S$ such that $r.F = s.K$. Throughout this paper, we restrict attention to *foreign key joins* — joins of the form $r.F = s.K$. We will use the term keyjoin as a shorthand for this type of join.

3.1 Joining Two Tables

Consider a medical database with two tables: Patient, containing tuberculosis (TB) patients, and Contact, containing people with whom a patient has had contact, and who may or may not be infected with the disease. We might be interested in answering queries involving a join between these two tables. For example, we might be interested in the following query: "patient.*Age* = *60+* and contact.*Patient* = patient.*Patient-ID* and contact.*Contype* = *roommate*", i.e., finding all patients whose age is over 60 who have had contact with a roommate.

A simple approach to this problem would proceed as follows: By referential integrity, each tuple in Contact must join with exactly one tuple in Patient. Therefore, the size of the joined relation, prior to the selects, is |Contact|. We then compute the probability p of Patient.*Age* = *60+* and the probability q of Contact.*Contype* = *roommate*, and estimate the size of the resulting query as $|\text{Contact}| \cdot p \cdot q$.

This naive approach is flawed in two ways. First, the attributes of the two different tables are often correlated. In general, foreign keys are often used to connect tuples in different tables that are semantically related, and hence the attributes of tuples related through foreign key joins are often correlated. For example, there is a clear correlation between the age of the patient and the type of contacts they have; in fact, elderly patients with roommates are quite rare, and this naive approach would overestimate their number. Second, the probability that two tuples join with each other can also be correlated with various attributes. For example, middle-aged patients typically have more contacts than older patients. Thus, while the join size of these two tables, prior to the selection on patient age, is |Contact|, the fraction of the joined tuples where the patient is over 60 is lower than the overall fraction of patients over 60 within the Patient table.

We address these issues by providing a more accurate model of the joint distribution of these two tables. Consider two tables R and S such that $R.F$ points to $S.K$. We define a joint probability space over R and S using an imaginary sampling process that randomly samples a tuple r from R and independently samples a tuple s from S. The two tuples may or may not join with each other. We introduce a new *join indicator variable* to model this event. This variable, J_F, is binary valued; it is true when $r.F = s.K$ and false otherwise.

This sampling process induces a distribution

$$P_{\mathcal{D}}(J_F, A_1, \ldots, A_n, B_1, \ldots, B_m)$$

over the values of the join indicator J_F, and the value attributes $R.* = \{A_1, \ldots, A_n\}$ and $S.* = B_1, \ldots, B_m$. Now, consider any select-keyjoin query Q over R and S: $r.\mathbf{A} = \mathbf{a}, s.\mathbf{B} = \mathbf{b}, r.F = s.K$ (where again we abbreviate a multidimensional select using vector notation). It is easy to to see that the size of the result of Q is:

$$\begin{aligned} \mathit{size}_Q[\mathcal{D}] &= F_{\mathcal{D}}(Q) \\ &= |R| \cdot |S| \cdot P_{\mathcal{D}}(\mathbf{A} = \mathbf{a}, \mathbf{B} = \mathbf{b}, J_F = \mathit{true}). \end{aligned}$$

In other words, we can estimate the size of any query of this form using the joint distribution $P_{\mathcal{D}}$ defined using our sampling process. As we now show, an extension of the techniques in Section 2 allow us to estimate this joint distribution using a probabilistic graphical model.

3.2 Probabilistic Relational Models

Probabilistic relational models (PRMs) [18] extend Bayesian networks to the relational setting. They allow us to model correlations not only between attributes of the same tuple, but also between attributes of related tuples in different tables. This extension is accomplished by allowing, as a parent of an attribute $R.A$, an attribute $S.B$ in another relation S such that R has a foreign key for S. We can also allow dependencies on attributes in relations that are related to R via a longer chain of joins; to simplify the notation, we omit the description. A PRM for our TB domain is shown in Fig. 3(a). Here, for example, we have that the type of the contact depends on the age and gender of the patient.

DEFINITION 3.1.: *A probabilistic relational model (PRM) Π for a relational database is a pair $(\mathcal{S}, \theta)$, which specifies a local probabilistic model for each of the following variables:*

- *for each table R and each attribute $A \in R.*$ a variable $R.A$;*

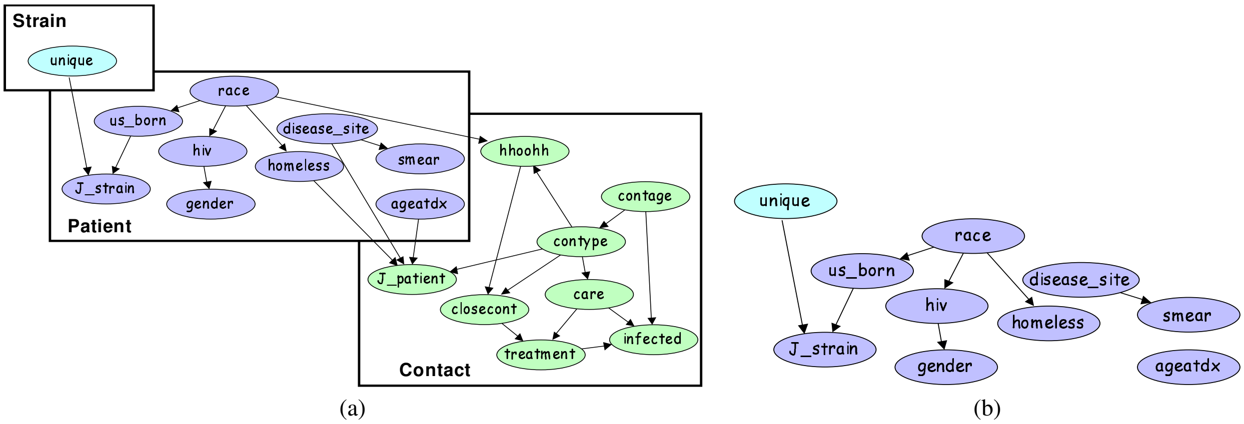


Figure 3: (a) A PRM for the Tuberculosis domain. (b) A query-evaluation BN for TB domain and the keyjoin query p.*Strain* $=$ s.*Strain-ID*.

- *for each foreign key F of R into S, a boolean join indicator variable $R.J_F$.*

For each variable of the form $R.X$:

- *$\mathcal{S}$ specifies a set of parents Parents$(R.X)$, where each parent has the form $R.B$ or $R.F.B$ where F is a foreign key of R into some table S and B is an attribute of S;*
- *θ specifies a CPD $P(R.X \mid$ Parents$(R.X))$.* ∎

This framework allows a probabilistic dependence of an attribute $r.A$ on an attribute $s.B$. This type of dependence only makes sense if s is related to r via some foreign key dependence: Our PRM models a distribution where r and s are chosen independently at random; there is no reason for their attributes to be correlated unless they are somehow related. Hence, we constrain the PRM model to allow $r.A$ to depend on $s.B$ only if $r.F = s.K$. More precisely, we require that if $R.F.B$ is a parent of $R.A$, then $R.J_F$ must also be a parent of R. We also require that the CPD of $R.A$ is only defined for cases where $R.J_F =$ *true*; in other words, in the CPD tree for $R.A$, $R.J_F$ is at the root of the tree, and only the fork in which $R.J_F =$ *true* is meaningful.

Note that the join indicator variable also has parents and a CPD. Consider the PRM for our TB domain. The join indicator variable Patient.J_{Strain} has the parents Patient.*USBorn* and Strain.*Unique*, which indicates whether the strain is unique in the population or has appeared in more than one patient. There are essentially three cases: for a non-unique strain and a patient that was born outside the U.S., the probability that they will join is around 0.001; for a non-unique strain and a patient born in the U.S. the probability is 0.0029, nearly three times as large; for a unique strain, the probability is 0.0004, regardless of the patient's place of birth. Thus, we are much more likely to have U.S.-born patients joining to non-unique strains than foreign-born ones. (The reason is that foreign-born patients often immigrate to the U.S. already infected with the disease; such patients typically have a unique strain indigenous to their region. U.S.-born patients, on the other hand, are much more likely to contract the disease by catching it from someone local, and therefore will appear in infection clusters.)

3.3 Selectivity Estimation using PRMs

We now wish to describe the relationship between a PRM model and the database. In the case of BNs, the connection was straightforward: the BN $\mathcal{B}_R$ is an approximation to the frequency distribution $P_{\mathcal{D}}$. In the case of PRMs, the issue is more subtle, as a PRM does not describe a distribution over a single table in isolation. The probability distribution of an attribute can depend on parents that are attributes in other, foreign-key related tuples. We therefore need to define a joint probability distribution over a tuple r together with all tuples on which it depends. To guarantee that the set of tuples we must consider is finite, we place a stratification restriction on our PRM models.

DEFINITION 3.2.**:** *Let $\prec$ be a partial ordering over the tables in our database. We say that a foreign key $R.F$ that connects to S is* consistent with *$\prec$ if $S \prec R$. A PRM Π is* (table) stratified *if there exists a partial ordering $\prec$ such that whenever $R.F.B$ is a parent of some $R.A$ (where F is a foreign key into S), then $S \prec R$.* ∎

We can now define the minimal extension to a query Q. Let Q be a keyjoin query over the tuple variables $r_1, \ldots, r_k$ (which may or may not refer to the same tables).

DEFINITION 3.3.**:** *Let Q be a keyjoin query. We define the* upward closure *Q^+ for Q to be the minimal query that satisfies the following two conditions:*

1. *Q^+ contains all of the join operations in Q.*
2. *For each r, if there is an attribute $R.A$ with parent $R.F.B$, where $R.F$ points to S, then there is a unique tuple variable s in Q^+ for which Q^+ contains the join $r.F = s.K$.* ∎

We can construct the upward closure for any query and that this set is finite. For example, if our query Q in the TB domain is over a tuple variable c from the Contact, then Q^+ is over the three tuple variables c, p, s where p is a tuple variable over Patient and s is a tuple variable over Strain. Note that there is no direct dependence of attributes of Contact on attributes of Strain, but there are dependencies on Patient, and the introduction of the tuple variable p in turn necessitates the introduction of the tuple variable s. Note that, if we consider a keyjoin query Q' that already contains a tuple variable p with the constraint c.*Patient* = p.*Patient-ID*, then the closure of Q' is identical to the closure of Q; i.e., the process will not introduce a new tuple variable if a corresponding one is already present. We can extend the definition of upward closure to select-keyjoin queries in the obvious way: the select clauses are not relevant to the notion of upward closure.

Upward closing a query does not change its result size:

PROPOSITION 3.4.**:** *Let Q be a query and let Q^+ be its upward closure. Then* $\text{size}_Q[\mathcal{D}] = \text{size}_{Q^+}[\mathcal{D}]$.

This result follows immediately from referential integrity.

Let Q be a keyjoin query, and let Q^+ be its upward closure. Let $r_1, \ldots, r_k$ be the tuple variables in Q^+. Let $P_{\mathcal{D}}(r_1, \ldots, r_k)$ be the distribution obtained by sampling each tuple $r_1, \ldots, r_k$ independently. Then for any query Q' which extends Q^+, the PRM allows us to approximate $P_{\mathcal{D}}(\mathcal{I}_{Q'})$, precisely the quantity required for estimating the query selectivity. We can compute the PRM estimate using the following simple construction:

DEFINITION 3.5.: *Let $\Pi = (\mathcal{S}, \theta)$ be a PRM over $\mathcal{D}$, and let Q be an keyjoin query. We define the* query-evaluation *Bayesian network $\mathcal{B}_\Pi[Q]$ to be a BN as follows:*

- *It has a node $r.A$ for every $r \in Q^+$ and attribute $A \in R.*$. It also has a node $r.J_F$ for every clause $r.F = s.K$ in Q^+.*
- *For every variable $r.X$, the node $r.X$ has the parents specified in Parents$(r.X)$ in $\mathcal{S}$: If $R.B$ is a parent of $R.A$, then $r.B$ is a parent of $r.A$; if $R.F.B$ is a parent of $R.A$, then $s.B$ is a parent of $r.A$. where s is the unique tuple variable for which Q^+ asserts that $r.F = s.K$.*
- *The CPD of $r.X$ is as specified in θ.* ∎

For example, Fig. 3(b) shows the query evaluation BN for the upwardly closed keyjoin query *p.Strain* $=$ *s.Strain-ID*.

We can now use this Bayesian network to estimate the selectivity of any query. Consider a select-keyjoin query Q' which extends the keyjoin query Q. We can estimate $P_{\mathcal{D}}(Q')$ by computing $P_{\mathcal{B}_\Pi[Q]}(\mathcal{I}_{Q'^+})$, where $\mathcal{I}_{r.A=a}$ is itself (as above), and $\mathcal{I}_{r.F=s.K}$ is $r.J_F$ $=$ *true*. For example, to evaluate the probability of the query *p.Age* = *60+*, we would use the BN in Fig. 3(b) (as it upward closes *p*), and compute the probability of (*p.Age* = *60+*,*p.Strain* $=$ *s.Strain-ID*). In reality, we don't need to construct a BN over all the nodes in the closure Q^+; it is sufficient to include only the attributes queried and their ancestors in $\mathcal{B}_\Pi[Q]$.

4. PRM CONSTRUCTION

The previous two sections showed how we can perform query size estimation once we have a PRM that captures the significant statistical correlations in the data distribution. This section addresses the question of how to construct such a model automatically from the relational database. Many of the ideas in this section are simple adaptations of previous work on the topic: the work on learning Bayesian networks from data (e.g., [15]) and the recent extension of this learning framework to PRMs [11].

The input to the construction algorithm consists of two parts: a relational schema, that specifies the basic vocabulary in the domain — the set of tables, the attributes associated with each of the tables, and the possible foreign key joins between tuples; and the database itself, which specifies the actual tuples contained in each table.

In the construction algorithm, our goal is to find a PRM $(\mathcal{S}, \theta)$ that best represents the dependencies in the data. In order to provide a formal specification for this task, we first need to define an appropriate notion of "best". Given this criterion, the algorithm will try to find the model that optimizes it. There are two parts to our optimization problem. The *parameter estimation problem* finds best parameter set θ for a given dependency structure $\mathcal{S}$. The *structure selection problem* finds the dependency structure $\mathcal{S}$ that, with the optimal choice of parameters, achieves the maximal score, subject to our space constraints on the model.

4.1 Scoring Criterion

To provide a formal definition of model quality, we make use of basic concepts from information theory [8]. The quality of a model can be measured by the extent to which it summarizes the data. In other words, if we had the model, how many bits would be required, using an optimal encoding, to represent the data. The more informative the model, the fewer bits are required to encode the data.

It is well known that the optimal Shannon encoding of a data set, given the model, uses a number of bits which is the negative logarithm of the probability of the data given the model. In other words, we define the *score* of a model $(\mathcal{S}, \theta)$ using the following *log-likelihood function*:

$$l(\theta, \mathcal{S} \mid \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{S}, \theta) \qquad (3)$$

We can therefore formulate the model construction task as that of finding the model that has maximum log-likelihood given the data.

We note that this criterion is different from those used in the standard formulations of learning probabilistic models from data [15]. In the latter cases, we typically choose a scoring function that trades off fit to data with model complexity. This tradeoff allows us to avoid fitting the training data too closely, thereby reducing our ability to predict unseen data. In this case, our goal is very different: We do not want to generalize to new data, but only to summarize the patterns in the existing data. This difference in focus is what motivates our choice of scoring function.

4.2 Parameter Estimation

We begin by considering the parameter estimation task for a given dependency structure. In other words, having selected a dependency structure $\mathcal{S}$ that determines the set of parents for each attribute, we must "fill in" the numbers θ that parameterize it. The parameter estimation task is a key subroutine in the structure selection step: to evaluate the score for a structure, we must first parameterize it. In other words, the highest scoring model is the structure whose best parameterization has the highest score.

It is well-known that the highest likelihood parameterization for a given structure $\mathcal{S}$ is the one that precisely matches the frequencies in the data. More precisely, consider some attribute A in table R and let $\mathbf{X}$ be its parents in $\mathcal{S}$. Our model contains a parameter $\theta_{a|\mathbf{x}}$ for each value a of A and each assignment of values $\mathbf{x}$ to $\mathbf{X}$. This parameter represents the conditional probability $P(R.A = a \mid \mathbf{X} = \mathbf{x})$. The maximum likelihood value for this parameter is simply the relative frequency of $R.A = a$ within the population of cases $\mathbf{X} = \mathbf{x}$:

$$\theta^*_{a|\mathbf{x}} = \frac{F_{\mathcal{D}}(R.A = a, \mathbf{X} = \mathbf{x})}{F_{\mathcal{D}}(\mathbf{X} = \mathbf{x})} \qquad (4)$$

The frequencies, or counts, used in this expression are called *sufficient statistics* in the statistical learning literature.

This computation is very simple in the case where the attribute and its parents are in the same table. For example, to compute the CPD associated with the Patient.*Gender* attribute in our TB model, we simply execute a count and group-by query on *Gender* and *HIV*, which gives us the counts for all possible values of these two attributes. These counts immediately give us the sufficient statistics in both the numerator and denominator of the entire CPD. This computation requires a linear scan of the data.

The case where some of the parents of an attribute appear in a different table is only slightly more complex. Recall that we restrict dependencies between tuples to those that utilize foreign-key joins. In other words, we can have $r.A$ depending on $s.B$ only if $r.F = s.K$ for a foreign key F in R which is also the primary key of S.

Thus, to compute the CPD for $R.A$ that depends on $S.B$, we simply need to execute a foreign-key join between R and S, and then use the same type of count and group-by query over the result.

For example, in our TB model, Contact.*Age* has the parents Contact.*Contype* and Contact.*Patient.Age*. To compute the sufficient statistics, we simply join Patient and Contact on Patient.*Patient-ID*=Contact.*Patient*, and then group and count appropriately.

We have presented this analysis for the case where $R.A$ depends only on a single "foreign" attribute $S.B$. However, the discussion clearly extends to dependencies on multiple attribute, potentially in different tables. We simply do all of the necessary foreign-key joins, generate a single result table over $R.A$ and all of its parents $\mathbf{X}$, and compute the sufficient statistics.

While this process might appear expensive at first glance, it can be executed very efficiently. Recall that we are only allowing dependencies via foreign-key joins. Putting that restriction together with our referential integrity assumption, we know that each tuple r will join with precisely a single tuple s. Thus, the number of tuples in the resulting join is precisely the size of R. The same observation holds when $R.A$ has parents in multiple tables. Assuming that we have a good indexing structure on keys (e.g., a hash index), the cost of performing the join operations is therefore linear in the size of R.

It remains to describe the computation of the CPD for a join indicator variable J_F. In this case, we must compute the probability that a random tuple r from R and a random tuple s from S will satisfy $r.F = s.K$. The probability of the join event can depend on values of attributes in r and s, e.g., on the value of $r.A$ and $s.B$. In our TB domain, the join indicator between Patient and Strain depends on *USBorn* within the Patient table and on *Unique* within the Strain table. To compute the sufficient statistics for $P(J_F \mid r.A, s.B)$, we need to compute the total number of cases where $r.A = a$, $s.B = b$, and then the number within those where $r.F = s.K$. Fortunately, this computation is also easy. The first is simply $F_{\mathcal{D}}(R.A = a) \cdot F_{\mathcal{D}}(S.B = b)$. The latter is $F_{\mathcal{D}}(R.A = a, S.B = b, R.F = S.K)$, which can be computed by joining the two tables and then doing a count and group-by query. The cost of this operation (assuming an appropriate index structure) is again linear in the number of tuples in R and in S.

4.3 Structure Selection

Our second task is the structure selection task: finding the dependency structure that achieves the highest log-likelihood score. The problem here is finding the best dependency structure among the superexponentially many possible ones.[2] This is a combinatorial optimization problem, and one which is known to be NP-hard [7]. We therefore provide an algorithm that finds a good dependency structure using simple heuristic techniques; despite the fact that the optimal dependency structure is not guaranteed to be produced, the algorithm nevertheless performs very well in practice.

4.3.1 Scoring revisited

The log-likelihood function can be reformulated in a way that both facilitates the model selection task and allows its effect to be more easily understood. We first require the following basic definition[8]:

DEFINITION 4.1.: *Let* $\mathbf{Y}$ *and* $\mathbf{Z}$ *be two sets of attributes, and consider some joint distribution* P *over their union. We can define the* mutual information *of* $\mathbf{Y}$ *and* $\mathbf{Z}$ *relative to* P *as:*

$$\boldsymbol{MI}_P(\mathbf{Y};\mathbf{Z}) = \mathbf{E}_P\left[\log \frac{P(\mathbf{y},\mathbf{z})}{\tilde{P}(\mathbf{y},\mathbf{z})}\right] = \sum_{\mathbf{y},\mathbf{z}} P(\mathbf{y},\mathbf{z}) \log \frac{P(\mathbf{y},\mathbf{z})}{\tilde{P}(\mathbf{y},\mathbf{z})}$$

where $\tilde{P}(\mathbf{y},\mathbf{z}) = P(\mathbf{y})P(\mathbf{z})$. ∎

The term inside the expectation is the logarithm of the relative error between $P(\mathbf{y},\mathbf{z})$ and an approximation to it $\tilde{P}$ that makes $\mathbf{y}$ and $\mathbf{z}$ independent, but maintains the probability of each one. The entire expression is simply a weighted average of this relative error over the distribution, where the weights are the probabilities of the events $\mathbf{y},\mathbf{z}$. It is intuitively clear that mutual information is measuring the extent to which $\mathbf{Y}$ and $\mathbf{Z}$ are correlated in P. If they are independent, then the mutual information is zero. Otherwise, the mutual information is always positive. The stronger the correlation, the larger the mutual information.

Consider a particular structure $\mathcal{S}$. Our analysis in the previous section specifies the optimal choice (in terms of likelihood) for parameterizing $\mathcal{S}$. We use $\theta_{\mathcal{S}}$ to denote this set of parameters. Let $P_{\mathcal{D}}$ be the distribution in the database, as above. We can now reformulate the log-likelihood score in terms of mutual information:

$$l(\mathcal{S}, \theta_{\mathcal{S}} \mid \mathcal{D}) = \left[\sum_i |R_i| \sum_{A \in R_i.*} \boldsymbol{MI}_{P_{\mathcal{D}}}(R_i.A; \mathrm{Pa}(R_i.A))\right] + C \qquad (5)$$

where C is a constant that does not depend on the choice of structure. Thus, the overall score of a structure decomposes as a sum, where each component is local to an attribute and its parents. The local score depends directly on the mutual information between a node and its parents in the structure. Thus, our scoring function prefers structures where an attribute is strongly correlated with its parents. We will use $\mathrm{score}_l(\mathcal{S} : \mathcal{D})$ to denote $l(\mathcal{S}, \theta_{\mathcal{S}} \mid \mathcal{D})$.

4.3.2 Model space

An important design decision is the space of dependency structures that we allow the algorithm to consider. Several constraints are imposed on us by the semantics of our models. Bayesian networks and PRMs only define a coherent probability distribution if the dependency structure is acyclic, i.e., there is no directed path from an attribute back to itself. Thus, we restrict attention to dependency structures that are directed acyclic graphs. Furthermore, we have placed certain requirements on inter-table dependencies: a dependency of $R.A$ on $S.B$ is only allowed if $S.K$ is a foreign key in R, and if the join indicator variable J_F is also a parent of $R.A$, and plays the appropriate role in the CPD tree. Finally, we have required that the dependency structure be stratified along tables, as specified above.

A second set of constraints is implied by computational considerations. A database system typically places a bound on the amount of space used to specify the statistical model. We therefore place a bound on the size of the models constructed by our algorithm. In our case, the size is typically the number of parameters used in the CPDs for the different attributes, plus some small amount required to specify the structure. A second computational consideration is the size of the intermediate group-by tables constructed to compute the CPDs in the structure. If these tables get very large, storing and manipulating them can get expensive. Therefore, we often choose to place a bound on the number of parents per node.

4.3.3 Search algorithm

Given a set of legal candidate structures, and a scoring function that allows us to evaluate different structures, we need only provide a procedure for finding a high-scoring hypothesis in our space. The simplest heuristic search algorithm is greedy hill-climbing search, using random steps to escape local maxima. We maintain our current candidate structure $\mathcal{S}$ and iteratively improve it. At each it-

[2] If we have m attributes in a single table, the number of possible dependency structures is $2^{O(m^2)}$. If we have multiple tables, the expression is slightly more complicated, because not all dependencies between attributes in different tables are legal.

eration, we consider a set of simple local transformations to that structure. For each resulting candidate successor $\mathcal{S}'$, we check that it satisfies our constraints, and select the best one. We restrict attention to simple transformations such as adding or deleting an edge, and adding or deleting a split in a CPD tree. This process continues until none of the possible successor structures $\mathcal{S}'$ have a higher score than $\mathcal{S}$. At this point, the algorithm can take some number of random steps, and then resume the hill-climbing process. After some number of iterations of this form, the algorithm halts and outputs the best structure discovered during the entire process.

One important issue to consider is how to choose among the possible successor structures $\mathcal{S}'$ of a given structure $\mathcal{S}$. The most obvious approach is to simply choose the structure $\mathcal{S}'$ that provides the largest improvement in score, i.e., that maximizes $\Delta_l(\mathcal{S}', \mathcal{S}) = \mathrm{score}_l(\mathcal{S}' : \mathcal{D}) - \mathrm{score}_l(\mathcal{S} : \mathcal{D})$. However, this approach is very shortsighted, as it ignores the cost of the transformation in terms of increasing the size of the structure. We now present two approaches that address this concern.

The first approach is based on an analogy between this problem and the *weighted knapsack problem*: We have a set of items, each with a value and a volume, and a knapsack with a fixed volume; our goal is to select the largest value set of items that fits in the knapsack. Our goal here is very similar: every edge that we introduce into the model has some value in terms of score and some cost in terms of space. A standard heuristic for the knapsack problem is to greedily add the item into the knapsack that has, not the maximum value, but the largest value to volume ratio. In our case, we can similarly choose the edge for which the likelihood improvement normalized by the additional space requirement:

$$\Delta_n(\mathcal{S}', \mathcal{S}) = \frac{\Delta_l(\mathcal{S}', \mathcal{S})}{\mathrm{space}(\mathcal{S}') - \mathrm{space}(\mathcal{S})}$$

is largest.[3] We refer to this method of scoring as storage size normalized (SSN).

The second idea is to use a modification to the log-likelihood scoring function called MDL (minimum description length). This scoring function is motivated by ideas from information and coding theory. It scores a model using not simply the negation of the number of bits required to encode the data given the model, but also the number of bits required to encode the model itself. This score has the form

$$\mathrm{score}_{mdl}(\mathcal{S} : \mathcal{D}) = l(\mathcal{S}, \theta_{\mathcal{S}} \mid \mathcal{D}) - \mathrm{space}(\mathcal{S}).$$

We define $\Delta_m(\mathcal{S}', \mathcal{S}) = \mathrm{score}_{mdl}(\mathcal{S}' : \mathcal{D}) - \mathrm{score}_{mdl}(\mathcal{S} : \mathcal{D})$.

We have experimentally compared the naive approach with the two ideas outlined above on the Census dataset described in Section 2.2. Both SSN and MDL scoring achieved higher log-likelihood than the naive approach for a fixed amount of space. In fact, SSN and MDL performed almost identically for the entire range of allocated space, and no clear winner was evident.

All three approaches involve the computation of $\Delta_l(\mathcal{S}', \mathcal{S})$. Eq. (5) then provides a key insight for improving the efficiency of this computation. As we saw, the score decomposes into a sum, each of which is associated only with a node and its parents in the structure. Thus, if we modify the parent set or the CPD of only a single attribute, the terms in the score corresponding to other attributes remain unchanged [15]. Thus, to compute the score corresponding to a slightly modified structure, we need only recompute the local score for the one attribute whose dependency model has changed.

[3] This heuristic has provable performance guarantees for the knapsack problem. Unfortunately, in our problem the values and costs are not linearly additive, so there is no direct mapping between the problems and the same performance bounds do not apply.

Furthermore, after taking a step in the search, most of the work from the previous iteration can be reused.

To understand this idea, assume that our current structure is $\mathcal{S}$. To do the search, we evaluate a set of changes to $\mathcal{S}$, and select the one that gives us the largest improvement. Say that we have chosen to update the local model for $R.A$ (either its parent set or its CPD tree). The resulting structure is $\mathcal{S}'$. Now, we are considering possible local changes to $\mathcal{S}'$. The key insight is that the component of the score corresponding to another attribute $S.B$ has not changed. Hence, the change in score resulting from the change to $S.B$ is the same in $\mathcal{S}$ and in $\mathcal{S}'$, and we can simply reuse that number unchanged. Only changes to $R.A$ need to be re-evaluated after moving to $\mathcal{S}'$.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results for a variety of real-world data, including: a census dataset [5]; a subset of the database of financial data used in the 1999 European KDD Cup [4]; and a database of tuberculosis patients in San Francisco [3]. We begin by evaluating the accuracy of our methods on select queries over a single relation. We then consider more complex, select-join queries over several relations. Finally, we discuss the running time for construction and estimation for our models.

Select Queries. We evaluated accuracy for selects over a single relation on a dataset from Census database described above (approximately 150K tuples). We performed two sets of experiments.

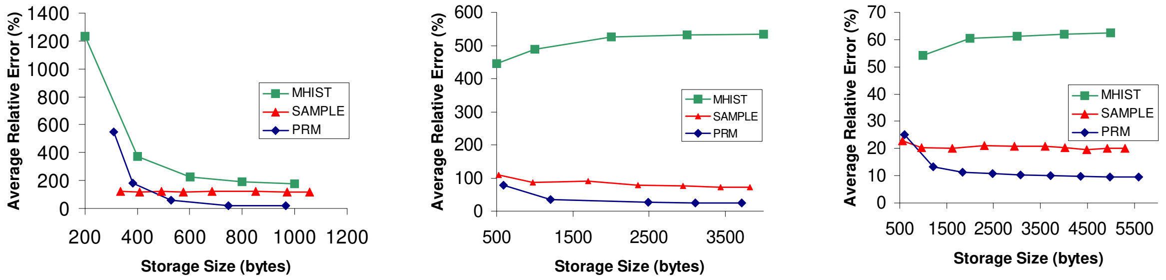
In the first set of experiments, we compared our approach to an existing selectivity estimation technique — multidimensional histograms. Multidimensional histograms are typically used to estimate the joint over some small subset of attributes that participate in the query. To allow a fair comparison, we applied our approach (and others) in the same setting. We selected subsets of two, three, and four attributes of the Census dataset, and estimated the query size for the set of all equality select queries over these attributes.

We compared the performance of four algorithms. **AVI** is a simple estimation technique that assumes attribute value independence: for each attribute a one dimensional histogram is maintained. In this domain, the domain size of each attribute is small, so it is feasible to maintain a bucket for each value. This technique is representative of techniques used in existing cost-based query optimizers such as System-R. **MHIST** builds a multidimensional histogram over the attributes, using the V-Optimal(V,A) histogram construction of Poosala *et al.* [25].[4] This technique constructs buckets that minimize the variance in area (frequency $\times$ value) within each bucket. Poosala *et al.* found this method for building histograms to be one of the most successful in experiments over this domain. **SAMPLE** constructs a random sample of the table and estimates the result size of a query from the sample. **PRM** uses our method for query size estimation. Unless stated otherwise, PRM uses tree CPDs and the SSN scoring method.

We compare the different methods using the adjusted relative error of the query size estimate: If S is the actual size of our query and $\hat{S}$ is our estimate, then the adjusted relative error is $(|S - \hat{S}|)/\max(1, S)$. For each experiment, we computed the average adjusted error over all possible instantiations for the select values of the query; thus each experiment is typically the average over several thousand queries.

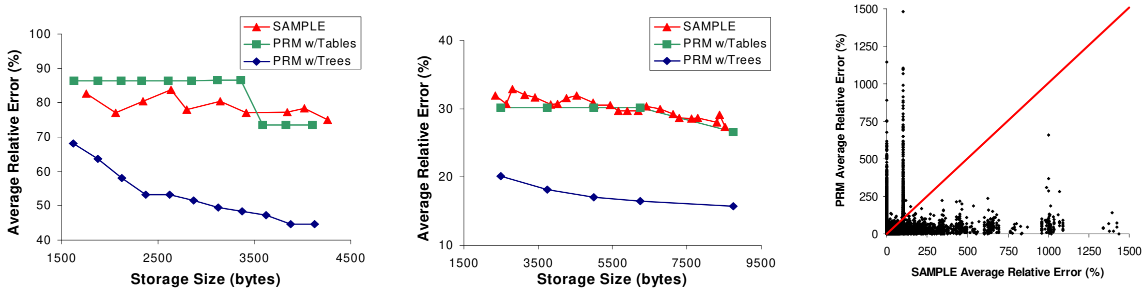
We evaluated the accuracy of these methods as we varied the space allocated to each method (with the exception of AVI, where

[4] We would like to thank Vishy Poosala for making this code available to us for our comparisons.



(a) (b) (c)

Figure 4: Relative error vs. storage size for a query suite over the Census dataset. (a) Two attribute query (Age and Income). The relative error for AVI is 7395. (b) Three attribute query (Age, HoursPerWeek and Income). The relative error for AVI is 865. (c) Four attribute query (Age, Education, HoursPerWeek and Income). The relative error for AVI is 70.19.



(a) (b) (c)

Figure 5: Relative error vs. storage size for a query over the Census dataset, with models constructed over 12 attributes. (a) Three attribute query (WorkerClass, Education, and MaritalStatus) (b) Four attribute query (Income, Industry, Age and EmployType). (c) A scatter plot showing the error on individual queries for a three attribute query (Income, Industry, Age) for SAMPLE and PRM (using 9.3K bytes of storage). In the scatter plot, each point represents a query, where the x coordinate is the relative error using SAMPLE and the y coordinate is the relative error using PRM. Thus, points above the diagonal line correspond to queries on which SAMPLE outperforms PRM and points below the diagonal line correspond to queries on which PRM performs better.

the model size is fixed). Fig. 4 shows results on Census for three query suites: over two, three, and four attributes. MHIST and SAMPLE, and all methods significantly outperform AVI. Note that a BN with tree CPDs over two attributes is simply a slightly different representation of a multi-dimensional histogram. Thus, it is interesting that our approach still dominates MHIST, even in this case. As the power of the representations is essentially equivalent here, the success of PRMs in this setting is due to the different scoring function for evaluating different models, and the associated search algorithm.

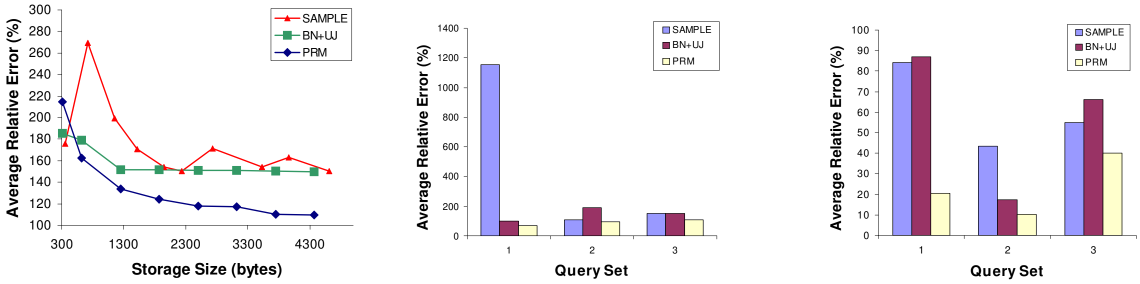
In the second set of experiments, we consider a more challenging setting, where a single model is built for the entire table, and then used to evaluate any select query over that table. In this case, MHIST is no longer applicable, so we compared the accuracy of PRM to SAMPLE, and also compared to PRMs with table CPDs. We built a PRM (BN) for the entire set of attributes in the table, and then queried subsets of three and four attributes. Similarly, for SAMPLE, the samples included all 12 attributes.

We tested these approaches on the Census dataset with 12 attributes. The results for two different query suites are shown in Fig. 5(a) and (b). Although for very small storage size, SAMPLE achieves lower errors, PRMs with tree CPDs dominates as the storage size increases. Note also that tree CPDs consistently outperform table CPDs. The reason is that table CPDs force us to split all bins in the CPD whenever a parent is added, wasting space on making distinctions that might not be necessary. Fig. 5(c) shows the performance on a third query suite in more detail. The scatter plot compares performance of SAMPLE and PRM for a fixed storage size (9.3K bytes). Here we see that PRM outperforms SAMPLE on the majority of the queries. (The spike in the plot at SAMPLE error 100% corresponds to the large set of query results estimated to be of size 0 by SAMPLE.)

Select-Join Queries. We evaluate the accuracy of estimation for select-join queries on two real-world datasets. Our financial database (FIN) has three tables: Account (4.5K tuples), Transaction (106K tuples) and District (77 tuples); Transaction refers through a foreign key to Account and Account refers to District. The tuberculosis database (TB) also has three tables: Patient (2.5K tuples), Contact (19K tuples) and Strain (2K tuples); Contact refers through a foreign key to Patient and Patient refers to Strain. Both databases satisfy the referential integrity assumption.

We compared the following techniques. **SAMPLE** constructs a random sample of the join of all three tables along the foreign keys and estimates the result size of a query from the sample. **BN+UJ** is a restriction of the PRM that does not allow any parents for the join indicator variable and restricts the parents of other attributes to be in the same relation. This is equivalent to a model with a BN for each relation together with the uniform join assumption. **PRM** uses unrestricted PRMs. Both PRM and BN+UJ were constructed using tree-CPDs and SSN scoring.

We tested all three approaches on a set of queries that joined all



(a) (b) (c)

Figure 6: (a) Relative error vs. storage size for a select-join query over three tables in the TB domain with selection on 3 attributes. (b) Relative error for three query sets on TB (c) Relative error for three query sets on FIN

three tables (although all three can also be used for a query over any subset of the tables). The queries select one or two attributes from each table. For each query suite, we averaged the error over all possible instantiations of the selected variables. Note that all three approaches were run so as to construct general models over all of the attributes of the tables, and not in a way that was specific to the query suite.

Fig. 6(a) compares the accuracy of the three methods for various storage sizes on a three attribute query in the TB domain. The graph shows both BN+UJ and PRM outperforming SAMPLE for most storage sizes. Fig. 6(b) compares the accuracy of the three methods for several different query suites on TB, allowing each method 4.4K bytes of storage. Fig. 6(c) compares the accuracy of the three methods for several different query suites on FIN, allowing 2K bytes of storage for each. These histograms show that PRM always outperforms BN+UJ and SAMPLE.

Running Time. Finally, we examine the running time for construction and estimation for our models. These experiments were performed on a Sparc60 workstation running Solaris2.6 with 256MB of internal memory.

We first consider the time required by the offline construction phase, shown in Fig. 7(a). As we can see, the construction time varies with the amount of storage allocated for the model: Our search algorithm starts with smallest possible model in its search space (all attributes independent of each other), so that more search is required to construct the more complex models that take advantage of the additional space. Note that table CPDs are orders of magnitude easier to construct than tree CPDs; however, as we discussed, they are also substantially less accurate.

The running time for construction also varies with the amount of data in the database. Fig. 7(b) shows construction time versus dataset size for tree CPDs and table CPDs for fixed model storage size (3.5K bytes). Note that, for table CPDs, running time grows linearly with the data size. For tree CPDs, running time has high variance and is almost independent of data size, since the running time is dominated by the search for the tree CPD structure once sufficient statistics are collected.

The online estimation phase is, of course, more time-critical than construction, since it is often used in the inner loop of query optimizers. The running time of our estimation technique varies roughly with the storage size of the model, since models that require a lot of space are usually highly interconnected networks which require somewhat longer inference time. The experiments in Fig. 7(c) illustrate the dependence. The estimation time for both methods is quite reasonable. The estimation time for tree CPDs is significantly higher, but this is using an algorithm that does not fully exploit the tree-structure; we expect that an algorithm that is optimized for tree CPDs would perform on a par with the table estimation times.

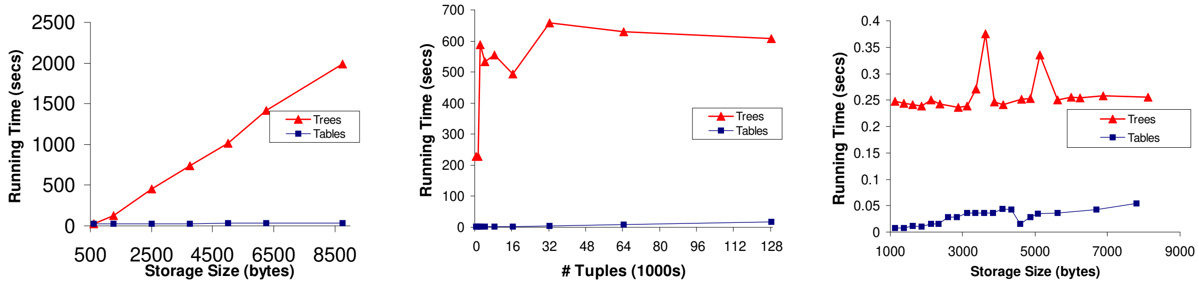
6. CONCLUSIONS

In this paper, we have presented a novel approach for estimating query selectivity using probabilistic graphical models — Bayesian networks and their relational extension. Our approach utilizes probabilistic graphical models, which exploit conditional independence relations between the different attributes in the table to allow a compact representation of the joint distribution of the database attribute values. We have tested our algorithm on several real-world databases in a variety of domains — medical, financial, and social. The success of our approach on all of these datasets indicates that the type of structure exploited by our methods is very common, and that our approach is a viable option for many real-world databases.

Our approach has several important advantages. To our knowledge, it is unique in its ability to handle select and join operators in a single unified framework, thereby providing estimates for complex queries involving several select and join operations. Second, our approach circumvents the dimensionality problems associated with multi-dimensional histograms. Multi-dimensional histograms, as the dimension of the table grows, either grow exponentially or become less and less accurate. Our approach estimates the high-dimensional joint distribution using a set of lower-dimensional conditional distributions, each of which is quite accurate. As we saw, we can put these conditional distributions together to get a good approximation to the entire joint distribution. Thus, our model is not limited to answering queries over a small set of predetermined attributes that happen to appear in a histogram together; it can be used to answer queries over an arbitrary set of attributes in the database.

There are several important topics that we have not fully addressed in this paper. One is the incremental maintenance of the PRM as the database changes. It is straightforward to extend our approach to adapt the parameters of the PRM over time, keeping the structure fixed. To adapt the structure, we can apply a variant of the approach of [13]. We can also keep track of the model score, relearning the structure if the score decreases drastically.

Another important topic that we have not discussed is joins over non-key attributes. In our presentation and experiments, our queries use only foreign key joins. While this category of queries stands to benefit most from the probabilistic models that we propose, our methods are more general. We can compute estimates for queries that join non-key attributes by summing over the possible values of the joined attributes, and our estimates are likely to be more



(a) (b) (c)

Figure 7: (a) The construction time for a PRM for Census using tree and table CPDs as a function of model storage space. (b) The construction time for a PRM for Census using tree and table CPDs as a function of data size. (c) The running time for query size estimation as a function of model size.

accurate than methods that do not model any of the dependencies between tuples. However an empirical investigation is required to evaluate our methods on this category of queries.

There are many interesting extensions to our approach, which we intend to investigate in future work. First, we want to extend our techniques to handle much larger databases; we believe that an initial single pass over the data can be used to "home in" on a much smaller set of candidate models, the sufficient statistics for which can then be computed very efficiently in batch mode. More interestingly, there are obvious applications of our techniques to the task of approximate query answering, both for OLAP queries and for general database queries (even queries involving joins).

Acknowledgments. We would like to thank Chris Olsten and Vishy Poosala for their helpful feedback. This work was supported ONR contract N66001-97-C-8554 under DARPA's HPKB program, and by the generosity of the Sloan Foundation and the Powell foundation.

REFERENCES

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*. ACM Press, 1999.

[2] D. Barbará, W. DuMouchel, C. Faloutsos, P. Haas, J. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. Sevcik. The New Jersey data reduction report. *Data Engineering Bulletin*, 20(4), 1997.

[3] M.A. Behr, M.A. Wilson, W.P. Gill, H. Salamon, G.K. Schoolnik GK, S. Rane S, and P.M. Small. Comparative genomics of BCG vaccines by whole genome DNA microarray. *Science*, 284, 1999.

[4] P. Berka. PKDD99 discovery challenge. `http://lisp.vse.cz/pkdd99/chall.htm`, 1999.

[5] U.S. Census Bureau. Census bureau databases. `http://www.census.gov`.

[6] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB*. Morgan Kaufmann, 2000.

[7] D. M. Chickering. Learning Bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*. Springer Verlag, 1996.

[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[9] M. desJardins, L. Getoor, and D. Koller. Using feature hierarchies in bayesian network learning. *Lecture Notes in Artificial Intelligence*, 1864, 2000.

[10] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *TODS*, 21(3), 1996.

[11] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. IJCAI*, 1999.

[12] N. Friedman and M. Goldszmidt. Discretization of continuous attributes while learning Bayesian networks. In *Proc. ICML*, 1996.

[13] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proc. UAI*, 1997.

[14] N. Fuhr. A probabilistic relational model for the integration of ir and databases. In *ACM-SIGIR*. ACM, 1993.

[15] D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1998.

[16] D. Heckerman, J. Breese, and K. Rommelse. Troubleshooting under uncertainty. Technical Report MSR-TR-94-07, Microsoft Research, 1994.

[17] M. I. Jordan, editor. *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

[18] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. AAAI*, 1998.

[19] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2), 1988.

[20] R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *SIGMOD*. ACM Press, 1990.

[21] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*. ACM Press, 1998.

[22] S. Monti and G. Cooper. A multivariate discretization method for learning Bayesian networks from data. In *Proc. UAI*, 1998.

[23] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensionnal queries. In *SIGMOD*. ACM Press, 1988.

[24] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[25] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*. Morgan Kaufmann, 1997.

[26] M. Pradhan, G.M. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proc. UAI*, 1994.

[27] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*. ACM Press, 1999.