Linköping Electronic Articles in
Computer and Information Science
Vol. (): nr

# Learning Structured Statistical Models from Relational Data

Lise Getoor
Computer Science Department / UMIACS
University of Maryland, College Park
College Park, MD 20740, USA
getoor@cs.umd.edu

Nir Friedman
The School of Computer Science & Engineering
HebrewUniversity
Jerusalem 91904, Israel
nir@cs.huji.ac.il

Daphne Koller
Computer Science Department
Stanford University
Stanford, CA 94305
koller@cs.stanford.edu

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis///

*Published on , by*
*Linköping University Electronic Press*
*581 83 Linköping, Sweden*

**Linköping Electronic Articles in**
**Computer and Information Science**
*ISSN 1401-9841*
*Series editor: Erik Sandewall*


*Typeset by the author using* LaTeX
*Formatted using* étendu *style*

**Recommended citation:**
*<Author>. <Title>. Linköping Electronic Articles in*
*Computer and Information Science, Vol. (): nr .*
http://www.ep.liu.se/ea/cis///. , .

*This URL will also contain a link to the author's home page.*

**Abstract**

Here we describe tools for constructing structured statistical models from relational data. Our goal is to learn structured probabilistic models that represent statistical correlations both between the properties of an entity and between the properties of related entities. These statistical models can then be used for a variety of tasks including knowledge discovery, exploratory data analysis, data summarization and anomaly detection. Unfortunately, most statistical learning methods work only with "flat" data representations. Thus, to apply these methods, we are forced to convert the data into a flat form, thereby not only losing its compact representation and structure but also potentially introducing statistical skew. These drawbacks severely limit the ability of current statistical methods to model relational databases. Here we describe two complementary approaches: one approach suited to making probabilistic statements about individuals and the second approach suited to making statements about frequencies in relational data. We describe algorithms for both learning and making inferences in these models.

# 1 Introduction

In this paper we present two frameworks for representing, reasoning and learning structured statistical models (SSMs) of relational data. The first framework, a probabilistic relational model (PRM) [20], supports probabilistic statements about individuals in relational domains. The second framework, a statistical relational model (SRM) [12], supports statements about frequencies in relational domains. The frameworks have much in common syntactically. In addition, they have closely related learning algorithms. However the semantics of the models are quite different: both the types of queries that can be answered using the models and the complexity of doing inference in each of the models are different.

Halpern [15] identified two distinct semantics for first-order logics of probability, which we will call the *possible-worlds* approach and the *domain-frequency* approach. We will see that PRMs are an instance of the possible-worlds approach, while an SRM is an instance of the domain-frequency approach to first-order probabilistic logics.

In the possible-worlds approach, we put a distribution over possible worlds. The distribution can be used to answer degree of belief questions such as 'What is the probability that a particular person, Joe Smith, has purchased a particular tractor part, Widget Supreme?'. In this case, the semantics of the formula is defined by summing up the probability over all of the possible worlds in which Joe Smith has bought Widget Supremes. However, as many have noted [2, 1], there are difficulties with making statistical assertions, such as '75% of the purchases were for widgets'.

The second approach, the domain-frequency approach, is appropriate for giving semantics to statistical queries such as, 'What is the probability that a randomly chosen individual has purchased a widget?'. In this case the semantics are defined in terms of a particular world, and are the result of an experiment performed in this world. In this approach, statements such as '80% of the purchases are made by men' make sense. Whereas questions such as, 'What is the probability Joe Smith purchased a Widget Supreme?' do not make sense; the probability of the purchase, in a particular world, is either 0 or 1.

A PRM allows us to describe a template for a probability distribution. This, together with a set of domain objects, defines a distribution over the attributes of the objects. Such a model can then be used for reasoning about an entity using the entire rich structure of knowledge encoded by the relational representation. We can make inferences about an individual based on its particular relational context; these predictions will be different for a similar individual that is in a significantly different relational context.

An SRM supports answering the statistical queries of the domain-frequency approach. An SRM is a statistical model of a particular database instantiation and is useful for summarizing and compressing the database. Further, rather than query the database, we can use the SRM to give fast (approximate) answers to queries on this database. Our algorithm is the first general approach to giving estimates for queries over multiple tables that does not rely on a predefined restrictions on the structure of the query.

This paper presents a survey of work that has appeared previously on PRMs and SRMs. The main contribution is the presentation of the two models together, and the contrast and comparison of the two models.

# 2 Structured Statistical Models

Here we describe the common syntactic components of PRMs and SRMs. The main components are the relational schema and the probabilistic schema.

## 2.1 The Relational Language

Here we give the vocabulary for our relational language. We use database terminology here, although it is straightforward to map this onto relational logic.

The relational schema defines the structure of our database. It defines the tables and the attributes of the tables. These are defined extensionally (later we will say a bit more about views or intensionally defined relations).

**Definition 1:** A *relation schema* $\mathcal{R}$ consists of a set of tables $\mathcal{R} = \{R_1, \ldots, R_m\}$. Each table $R$ is associated with attributes of three types: a single primary key $R.K$,[1] a set $\mathcal{F}(R)$ of foreign keys, and a set $\mathcal{A}(R)$ of descriptive attributes. Each foreign key $R.F$ is associated with a table into which it points, $\mathrm{Dom}[R.F] \in \mathcal{R}$. Each descriptive attribute $R.A$ is associated with a domain of possible values $\mathcal{V}(R.A)$. ∎

**Definition 2:** A database $\mathcal{D}$ over $\mathcal{R}$ consists of a set of tuples $\mathcal{T}_{\mathcal{D}}[R]$ for each table $R$. For each $t \in \mathcal{T}_{\mathcal{D}}[R]$:

- The primary key $t.K$ is unique within $R$.
- For each $F \in \mathcal{F}(R)$, $t.F$ is the primary key of some tuple in $\mathcal{T}_{\mathcal{D}}[S]$ where $S = \mathrm{Dom}[R.F]$.
- For each $A \in \mathcal{A}(R)$, $t.A$ is a value in $\mathcal{V}(R.A)$. ∎

Note that the second bullet in our definition restricts 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 $r.F = s.K$ [27]. We require referential integrity, but our model can easily be extended to accommodate cases where the foreign key takes the value *null*, indicating that there is no related tuple in $\mathrm{Dom}[R.F]$.

## 2.2 The Probabilistic Schema

The probabilistic schema gives a template which describes both the qualitative dependence structure of the attributes and the quantitative local conditional probability distribution (CPD) for an attribute. Here we define a simple generic syntax for the template; as we will see in the next sections, the manners in which the template is used to defining the semantics for a PRM and an SRM are quite distinct.

**Definition 3:** A *probabilistic schema (PS)* $\mathcal{PS}$ for a relational schema $\mathcal{R}$ is a pair $(\mathcal{S}, \theta)$, which specifies a local probability model for each table $R$ and each attribute $A \in \mathcal{A}(R)$, a variable $R.A$; $\mathcal{S}$ specifies a finite set of parents $\mathrm{Pa}(R.A)$ and $\theta$ specifies a CPD $P(R.A \mid \mathrm{Pa}(R.A))$. ∎

---

[1] Our definition can easily be extended to the case where keys are composed of several attributes.

The parents of a variable $R.A$ are defined by projecting a view including $R$ and possibly other relations, onto some subset of variables (not including $R.A$) $\mathrm{Pa}(R.A)$. This view may be defined intentionally and the variables may include aggregates. As an example, suppose we have three tables $R(K_1, A, B, F_1, F_2)$, $S(K_2, C)$, $T(K_3, D)$. We may define the parents of $R.A$, $\mathrm{Pa}(R.A) = V(B, C, D)$ where:

$$V(K_1, B, C, D) : -R(K_1, A, B, F_1, F_2) \& S(F_1, C) \& T(F_2, D)$$

We will require that $K_1$ be a key for this constructed view. If for example, the relationship between $R$ and $S$ is one-many, then we can achieve this by computing an aggregate $\gamma$ over $C$, $C_\gamma$, where $\gamma$ is any single-valued deterministic function over the multiset. More formally, our language allows a notion of an aggregate $\gamma$; $\gamma$ takes a multi-set of values of some ground type, and returns a summary of it. The type of the aggregate can be the same as that of its arguments. However, we allow other types as well, e.g., an aggregate that reports the size of the set. Note also that, in general, we may specify arbitrary binding requirements among the parents.

Here, for simplicity, here we consider parents that are defined by a *chain* of foreign key joins. In this case, we use the short-hand $R.F_1.B$ and $R.F_2.D$ to denote the parents of $R.A$ above.

The CPD describes the generic statistical relationship between each variable and its parents. The CPD which specifies the distribution over the values of $R.A$ given any possible assignment of values to its parents. Let $\mathcal{V}(R.A)$ denote the space of possible values for $R.A$ and $\mathcal{V}(\mathrm{Pa}(R.A))$ denote the space of possible values for the parents of $R.A$. The CPD is *legal* if all of the conditional probabilities are positive, and if, for any particular instantiation of $R.A$, $a \in \mathcal{V}(R.A)$, the sum over all possible instantiations of $R.A$, for any particular instantiation, $\boldsymbol{u}$, of $R.A$'s parents is 1, i.e.,

$$\sum_{a \in \mathcal{V}(R.A)} P(R.A = a \mid \mathrm{Pa}(R.A) = \boldsymbol{u}) = 1.$$

This CPD may be represented in a number of ways. It may be represented as a table, as a set of multinomials. Alternatively, it can be represented as a tree [3], where the interior vertices represent splits on the value of some parent of $R.A$, and the leaves contain distributions over the values of $R.A$. In this representation, we find the conditional distribution over $R.A$ 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 leaf.

# 3 Probabilistic Relational Models: Semantics

PRMs [20, 25] can be seen as extending Bayesian networks with the concepts of objects, their properties, and relations between them. In a way, they are to Bayesian networks as relational logic is to propositional logic. A PRM specifies a template for a probability distribution over a database. The template includes the relational component, that describes the relational schema for our domain, and the probabilistic component, that describes the probabilistic dependencies that hold in our domain, as introduced above. A PRM has a coherent formal semantics in terms of probability distributions over sets of relational logic interpretations. Given a set of ground objects, a

PRM specifies a probability distribution over a set of interpretations involving these objects. A PRM, together with a particular database of objects and relations, defines a probability distribution over the attributes of the objects.

### 3.1 Schema Instantiation

An *instantiation* of a schema is simply a database $\mathcal{D}$ as defined earlier. It is a standard relational logic interpretation of this vocabulary. A PRM defines a probability distribution over a set of instances of a schema. Most simply, we assume that the set of objects and the relations between them are fixed, i.e., external to the probabilistic model. Then, the PRM defines only a probability distribution over the attributes of the objects in the model. The *relational skeleton* defines the possible instantiations that we consider; the PRM defines a distribution over the possible worlds consistent with the relational skeleton.

**Definition 4:** A *relational skeleton* $\sigma_r$ of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects $\mathcal{T}_{\sigma_r}[R]$ for each table R and for each object the value of the primary key $R.K$ and for all of the foreign keys $F \in \mathcal{F}(R)$. between the objects. However, it leaves the values of the attributes unspecified.

The relational skeleton defines the random variables in our domain; we have a random variable for each attribute of each object in the skeleton. A PRM then specifies a probability distribution over *completions* $\mathcal{D}$ of the skeleton.

The probabilistic schema defines the qualitative dependency structure, $\mathcal{S}$. As we saw, the dependency structure associates with each attribute $R.A$ a set of parents $\mathrm{Pa}(R.A)$. These correspond to *formal* parents; they will be instantiated in different ways for different objects in $\sigma_r$.

**Definition 5:** A *probabilistic relational model (PRM)* $\Pi$ for a relational schema $\mathcal{R}$ is defined as follows. For each table $R \in \mathcal{R}$ and each descriptive attribute $A \in \mathcal{A}(R)$, we have:

- a set of *parents* $\mathrm{Pa}(R.A) = \{U_1, \ldots, U_l\}$, where each $U_i$ has the form $X.B$ or $\gamma(X.\tau.B)$, where $\tau$ is a (possibly empty) chain of foreign keys and $\gamma$ is an aggregate of $X.\tau.B$.
- a legal *conditional probability distribution (CPD)*, $P(R.A \mid \mathrm{Pa}(()R.A)$.

∎

As mentioned in the introduction, PRMs define a distribution over possible worlds. The possible worlds are instantiations of the database that are consistent with the relational skeleton. Given any skeleton, we have a set of random variables of interest: the attributes $t.A$ of the objects in the skeleton. Formally, let $\mathcal{T}_{\sigma_r}[R]$ denote the set of objects in skeleton $\sigma_r$ from table $R$. The set of random variables for $\sigma_r$ is the set of attributes of the form $t.A$ where $t \in \mathcal{T}_{\sigma_r}[R]$ and $A \in \mathcal{A}(R)$ for some table $R$. The PRM specifies a probability distribution over the possible joint assignments of values to all of these random variables.

For a given skeleton $\sigma_r$, the PRM structure induces an *ground* Bayesian network over the random variables $t.A$.

**Definition 6:** A PRM $\Pi$ together with a skeleton $\sigma_r$ defines the following ground Bayesian network:

- There is a node for every attribute of every object $t \in \sigma_r(R)$, $r.A$.
- Each $t.A$ depends probabilistically on parents $\mathrm{Pa}(R.A)$
- The CPD for $t.A$ is $P(R.A \mid \mathrm{Pa}(R.A))$. ∎

As with Bayesian networks, the joint distribution over these assignments is factored. That is, we take the product, over all $t.A$, of the probability in the CPD of the specific value assigned by the instance to the attribute given the values assigned to its parents. Formally, this is written as follows:

$$\begin{aligned} P(\mathcal{D} \mid \sigma_r, \mathcal{S}, \theta_{\mathcal{S}}) &= \prod_{x \in \sigma_r} \prod_{A \in \mathcal{A}(x)} P(\mathcal{D}_{x.A} \mid \mathcal{D}_{\mathrm{Pa}(x.A)}) \\ &= \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \mathcal{T}_{\sigma_r}[X_i]} P(\mathcal{D}_{x.A} \mid \mathcal{D}_{\mathrm{Pa}(x.A)}) \end{aligned}$$

This expression is very similar to the chain rule for Bayesian networks. There are three primary differences. First, our random variables are the attributes of a set of objects. Second, the set of parents of a random variable can vary according to the relational context of the object — the set of objects to which it is related. Third, the parameters are shared; the parameters of the local probability models for attributes of objects in the same class are the identical.

As in any definition of this type, we have to take care that the resulting function from instances to numbers does indeed define a *coherent* probability distribution, i.e., where the sum of the probability of all instances is 1. In Bayesian networks, where the joint probability is also a product of CPDs, this requirement is satisfied if the dependency graph is acyclic: a variable is not an ancestor of itself. A similar condition is sufficient to ensure coherence in PRMs as well.

So far, we have assumed that the skeleton is external to the probabilistic model; in other words, the skeleton is assumed to be part of the input to the PRM. This limitation has two main implications. Most obviously, it implies that the model can only be used in settings where the relational structure is known. The PRM framework can be extended to accommodate uncertainty about the structural relationships between objects as well as about their properties [25, 9]. We have extended our learning algorithms to deal with such *link uncertainty*.

One approach to link uncertainty we have developed is *Existence Uncertainty*. Existence uncertainty is a simple approach to modeling the probability that a relationship exists between any two entities. Suppose we have the relation $Purchase(Key, Person, Item, Price)$. We add a boolean attribute, the *Exists* attribute, and build a probabilistic model for it, just as any other attribute in our domain. We can model the probability that a person purchases a particular item. This can depend on both properties of the person and properties of the item. The approach is described more fully in [9, 10].

We have also investigated the idea of learning PRMs with class hierarchies. Class hierarchies change the language in two important ways. In the non-hierarchical case, all objects in a table must share the same CPDs. Hierarchical models allow us to specialize the CPDs of attributes in different parts of the hierarchy. More interestingly, a hierarchical model allows us to create dependencies that might seem cyclic in the non-hierarchical case.

For example, we can now have a person's purchases of computer software depends on her computer hardware purchases. Note that this dependency is apparently cyclic at the level of the Purchases class, since an attribute of one purchase depends on the same attribute in another purchase. However, the division of purchases into two subclasses resolves this apparent cycle. In effect, this can be used as a bridge between a propositional model (Bayesian networks) and a relational model (Structured Statistical Models), by allowing us to reason about both individuals and classes of individuals. A preliminary discussion of these issues can be found in [11].

# 4 Statistical Relational Models: Semantics

We begin by considering general select-join queries and looking at the distribution induced by the query. A query over a relational schema $\mathcal{R}$ is defined using tuple variables and the standard relational operators $\times$ (cross-product), $\sigma$ (select), and $\bowtie$ (join). A tuple variable $r$ is typed, i.e., $\mathrm{Dom}[r] = R$, for some $R \in \mathcal{R}$, and refers to some tuple $t \in \mathcal{T}_{\mathcal{D}}[R]$. In a number of places we will refer to a set of tuple variables $r_1, \ldots, r_k$, and their associated tables, $R_1, \ldots, R_k$, where $R_i = \mathrm{Dom}[r_i]$.

Let $Q$ be a query over tuple variables $r_1, \ldots, r_k$ (which may or may not refer to the same tables). Let $\sigma_Q$ be the set of equality select clauses in $Q$; i.e., $\sigma_Q = \{r_{i_1}.A_{i_1} = a_{i_1}, \ldots, r_{i_l}.A_{i_l} = a_{i_l}\}$ and let $\bowtie_Q$ be the set of foreign-key joins in Q; i.e., $\bowtie_Q = \{r_{j_1}.F_{j_1} = s_{j_1}.K, \ldots, r_{j_m}.F_{j_m} = s_{j_m}.K\}$. We can write $Q$ as follows:

$$\bowtie_Q (\sigma_Q(R_1 \times \ldots \times R_k)).$$

We introduce an indicator variable $\mathcal{I}_Q$ indicating when the equalities in $Q$ hold.

**Definition 7:** For any query $Q$ over database $\mathcal{D}$, the joins in $Q$ induce a distribution $P_{\mathcal{D}}(Q)$ over the attributes of the tuple variables in $Q$, $R_1 \times \ldots \times R_k$. The probability that the query is satisfied is:

$$P_{\mathcal{D}}(\mathcal{I}_Q) = \frac{|\bowtie_Q (\sigma_Q(R_1 \times \ldots \times R_k))|}{|R_1| \times \ldots \times |R_k|}\ . \ \blacksquare$$

We can also view this joint distribution as generated by an imaginary process, where we independently sample a sequence of tuples $r_1, \ldots, r_k$ from $R_1, \ldots, R_k$, 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 $P_{\mathcal{D}}(Q)$.)

## 4.1 The Universal Foreign-Key Closure

While the expression in Definition 7 is computable for any select-join query, we will find it more useful if we can define a unique distribution induced by our database. This will allow us to make useful assertions about the distribution, for example asserting independence statements. To achieve this, we need to restrict our attention to a finite, acyclic collection of foreign-key joins. We begin by introducing an ordering over the tables in the schema:

**Definition 8:** Let $\prec$ be a partial ordering over the tables in our schema. We say that a foreign key $R.F$ with $\mathrm{Dom}[R.F] = S$ is *consistent with* $\prec$ if

$S \prec R$. Let $\mathbf{F}$ be a subset of the foreign keys in the database, $\mathbf{F} \subseteq \mathcal{F}(DB)$. A schema $\mathcal{R}$ is *(table) stratified* with respect to $\mathbf{F}$ if there exists a partial ordering $\prec$ such that for any $R.F \in \mathbf{F}$, $R.F$ is consistent with $\prec$.

Often times, the set of foreign keys we consider is the entire set of foreign keys in the database. In this case, we will just say that the schema is table stratified, without referring to the set of foreign keys.

Now we can define a new relation $\mathcal{U}$, which is the *universal foreign-key closure* of a database $\mathcal{D}$ with respect to a table stratification. This relation is never actually materialized, we merely use it as a tool in defining a unique distribution induced by our database, and, from this distribution, the set of independence assumptions that hold in our database. Intuitively, the query that we construct introduces a tuple variable for each table in our schema, and has a unique tuple variable for each foreign-key in the schema.

**Definition 9:** Let $\mathcal{D}$ be a database with relational schema $\mathcal{R}$ and let $\prec$ be a table stratification of $\mathcal{D}$ with respect to some subset $\mathbf{F}$ of $\mathcal{D}$'s foreign keys. The *universal foreign-key closure* of $\mathcal{D}$ with respect to $\mathbf{F}$ is defined by the query $\mathcal{U}$ we construct below. $\mathcal{T}(\mathcal{U})$ will be the set of tuples variables in our query. Initially, $\mathcal{T}(\mathcal{U})$ has one tuple variable for each of the tables that are leaves in $\prec$. Each tuple variable is initially marked unprocessed.

We will construct the full set of tuple variables in the query as follows. While there are tuple variables in $\mathcal{T}(\mathcal{U})$ that have not been processed:

- Let $r$ be an unprocessed tuple variable in $\mathcal{T}(\mathcal{U})$.
- Let $R = \mathrm{Dom}[r]$. For each $F \in \mathcal{F}(R)$, where $R.F$ refers to $S$, add a new unique tuple variable $s$ to $\mathcal{T}(\mathcal{U})$. This tuple variable is marked unprocessed. We say that $s$ is the tuple variable *associated* with $r.F$. We also add the join $r.F = s.K$ to $\bowtie_{\mathcal{U}}$.

Let $\mathcal{A}(\mathcal{U}) = \{A_1, \ldots, A_m\}$ be the attributes in $\mathcal{U}$ (in order to avoid ambiguity, assume attributes are prefixed by their associated tuple variable), and let $\mathcal{T}(\mathcal{U}) = \{t_1, \ldots, t_j\}$ be the tuple variables in the query. $\mathcal{U}$ is simply a query over the cross product of the relations with a new copy of the relation introduced for each tuple variable that we add. ∎

Given this query $\mathcal{U}$, we can define the probability distribution $P_{\mathcal{U}}$. As we will see, the distribution is induced by the frequency of occurrence of combinations of values of the attributes of the tuple variables in $\mathcal{U}$ and join events among the tuple variables. $P_{\mathcal{U}}$ is defined over the attributes of the tuple variables in $\mathcal{U}$ and over the join events defined by $\bowtie_{\mathcal{U}}$:

- Let $t_1, \ldots, t_j$ be the tuple variables $\mathcal{T}(\mathcal{U})$, and let $\mathcal{A}(\mathcal{U}) = \{A_1, \ldots, A_m\}$ be the attributes of the tuple variables.
- Let $\bowtie_{\mathcal{U}}$ be the joins of $\mathcal{U}$, then we introduce a join indicator variable $r_i.J_F$ for each $\bowtie$ $(r_i.F = s.K)$. The join indicator $r_i.J_F$ is true iff $r_i.F = s.K$. We use $\mathcal{J}(r_i)$ to denote the join indicators for a tuple variable $r_i$. Let $\mathcal{J}(\mathcal{U}) = \{J_1, \ldots, J_{p-m}\}$ denote all of the join indicators.

The set of random variables $P_{\mathcal{U}}$ is defined over is $\mathcal{V}(\mathcal{U}) = \mathcal{A}(\mathcal{U}) \cup \mathcal{J}(\mathcal{U}) = \{V_1, \ldots, V_p\}$.

### 4.2 Statistical Relational Models

SRMs allow us to compactly model $P_{\mathcal{U}}$. Further they allow us to efficiently answer certain frequency queries by constructing a small query-specific Bayesian network that can be used to compute the desired frequencies. SRMs, like PRMs, extend Bayesian networks [24] 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 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 $S$. As discussed earlier, we can also allow dependencies on attributes in relations that are related to $R$ via more complex relationships; here to simplify the notation, we discuss only dependencies along a single foreign-key join.

**Definition 10:** A *statistical relational model (SRM)* $\Psi$ for a relational schema $\mathcal{R}$ is a pair $(\mathcal{S}, \theta)$, which specifies a local probability model for each of the following variables:

- for each table $R$ and each attribute $A \in \mathcal{A}(R)$, a variable $R.A$;
- for each foreign key $F$ of $R$ into $S$, a Boolean join indicator variable $R.J_F$.

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

- $\mathcal{S}$ specifies a set of parents $\mathrm{Pa}(R.V)$, 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 $S$. If $R.F.B$ is a parent of $R.A$, then $R.J_F$ must also be a parent of $R.A$.
- $\theta$ specifies a CPD $P(R.V \mid \mathrm{Pa}(R.V))$. We also require that the CPD of $R.V$ is only defined in cases where each join indicator parent $R.J_F$ is true, $R.J_F = \mathit{true}$. ∎

We refer to the join-indicator variables that are $R.V$'s parents as $\mathrm{Pa}_J(R.V)$ and the attributes that are $R.A$'s parents $\mathrm{Pa}_A(R.V)$. We use $\mathcal{J}(\Psi)$ to refer to all of the join indicator variables in $\Psi$.

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 SRM 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 SRM model to allow $r.A$ to depend on $s.B$ only if $r.F = s.K$. The CPD of $R.A$ is only defined for cases where each $R.J_F \in \mathrm{Pa}_J(R.V)$ is true. In other words, in the CPD tree for $R.A$, the $R.J_F$ are at the top of the tree, and only the branch in which all $R.J_F = \mathit{true}$ is meaningful.

At this point, an SRM looks suspiciously like a PRM. And, at this syntactic level it is almost identical. Our descriptive attributes are allowed the same form of probabilistic dependencies as the descriptive attributes of PRMs. However, the models are distinguished by their semantics. A PRM is a model over a particular skeleton and can be used to answer questions about a particular individual. Answering queries (performing inference) in a PRM can be quite expensive. A naive approach requires us to construct the entire unrolled Bayesian network. SRMs answer a very different category of queries, and the algorithms for answering the queries are much more efficient.

We now wish to make a connection between an SRM and a database. An SRM describes a set of independence assumptions and the local distributions of attributes given their parents. Intuitively, we say that a database $\mathcal{D}$ is a model of an SRM $\Psi$ if the conditional independence assumptions made in $\Psi$ hold in $\mathcal{D}$ and if the local distributions match the frequencies in $\mathcal{D}$. This is made more precise in [12, 8].

Our goal is to use the SRM to answer queries rather than resorting to querying the database. We will show how to construct a Bayesian network from our SRM that can be use to compute the answer to a query. If $\mathcal{D} \models \Psi$ then the answer we will compute is correct. However this is only true for a certain category of queries we define below.

### 4.3 Queries and the Query Evaluation Network

We begin by restricting attention to a form of select-join queries over multiple tables we call *inverted-tree foreign-key-join* queries. These queries are over a subset of the tuple variables of the universal foreign-key closure. Intuitively, they are over an upwardly closed fragment of the forest defined via the foreign-key joins in $\mathcal{U}$, and may themselves form a forest.

**Definition 11:** Given a database $\mathcal{D}$ with universal foreign-key closure $\mathcal{U}$, let $T \subseteq \mathcal{T}(\mathcal{U})$. $T$ is *closed* with respect to $\mathcal{U}$ if for all $t \in T$, if for each foreign key $t.F$ of $T$ with associated tuple variable $s$, $s \in T$. Let

$$\mathcal{J}(T) = \{t.F \bowtie s.K \mid t \in T \text{ and } s \text{ is the associated tuple variable for } t.F\}.$$

**Definition 12:** Given a database $\mathcal{D}$ with universal foreign-key closure $\mathcal{U}$, Q is a *inverted-tree foreign-key-join* query over tuple variables $T = \{t_1, \ldots, t_k\}$, if $T$ is closed with respect to $\mathcal{U}$, $\bowtie_Q = \mathcal{J}(T)$, and we select on some subset of the attributes $\mathcal{A}(T)$, $\sigma_Q = \{A_i = a_i \mid A_i \in \mathcal{A}(T)\}$:

$$Q = \bowtie_Q (\sigma_Q(R_1 \times \ldots \times R_k)) \quad \blacksquare$$

If a query $Q$ satisfies these conditions, we refer to it as an *inverted-tree foreign-key-join* query. As shorthand, we will simply call them *legal* queries.

Now we show how we can construct a Bayesian network to compute the distribution induced by a legal query.

**Definition 13:** Let $\Psi = (\mathcal{S}, \theta)$ be an SRM over $\mathcal{D}$, and let $Q$ be a legal query. We define the *query-evaluation* Bayesian network $\Upsilon[Q]$ to be a BN as follows:

- For each attribute $r.A$ of every tuple variable $r \in Q$, it has a node $r.A$.
- It also has a node $r.J_F$ for every clause $r.F = s.K$ in $Q$.
- For every variable $r.V$, the node $r.V$ has parents $\mathrm{Pa}_{\mathcal{G}}(r.V)$.
- The CPD of $r.V$ is as specified in $\theta$. $\blacksquare$

The query evaluation Bayesian network defines the following distribution:

**Definition 14:** Let $\Psi = (\mathcal{S}, \theta)$ be an SRM over $\mathcal{D}$, let $Q$ be a legal query and let $\Upsilon[Q]$ be the query evaluation Bayesian network for $Q$. Then $\Upsilon[Q]$ defines the following distribution:

$$\Upsilon[Q] = \prod_{V_i \in \mathcal{V}Q} P_\theta(A_i \mid \mathrm{Pa}_A(A_i), J^*_{\mathcal{U}}(A_i) = \mathbf{T})$$

where $\mathrm{Pa}(A_i)$ are defined by $\Upsilon[Q]$ and $J^*_{\mathcal{U}}(A_i)$ are the join indicator variables that are ancestors of $A_i$.

In [8], we prove the following theorem:

**Theorem 15:** *Suppose database $\mathcal{D} \models \Psi$. Let $Q$ be a legal query and let $\Upsilon[Q]$ be the query evaluation Bayesian network for the query. Then*

$$P_{\mathcal{D}}(Q) = \Upsilon[Q].$$

# 5 Learning

Now that we have seen the different semantics for PRMs and SRMs, we have left the question of how the models can be automatically constructed. It turns out that although the semantics for the models are very different, the learning algorithms are largely the same and are closely related to the work in learning Bayesian networks [16].

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.

As with any data mining algorithm, there are two possible goals. One is the classic statistical inference problem of learning a statistical model from a random sample drawn from the distribution. Here, we would like to learn a model that will perform well on additional, newly generated data; in other words, we would like to ensure good generalization performance. We are interested in constructing a model that trades off fit to data with model complexity. This trade-off allows us to avoid fitting the training data too closely, which would reduce our ability to predict unseen data. The second possible goal is summarization or compression. In this case, the database *is* our distribution, and we would like to learn a compact model of the database. We accomplish this by imposing a storage limitation on the models, so that our task is to find the best model that satisfies our space constraints.

There are two components of the learning task: parameter estimation and structure learning. In the parameter estimation task, we assume that the qualitative dependency structure is known; i.e., the input consists of the schema and training database, as well as a qualitative dependency structure $\mathcal{S}$. The learning task is only to fill in the parameters that define the CPDs of the attributes. For structure learning, our goal is to find an probabilistic schema $\mathcal{PS}(\mathcal{S}, \theta)$ that optimizing some scoring criteria. We will consider two categories of scoring criteria, one suited to the generalization task and another suited to summarization.

## 5.1 Parameter Estimation

Here the dependency structure is known, e.g., we are given the structure $\mathcal{S}$ that determines the set of parents for each attribute, and our task is to learn the parameters $\theta_{\mathcal{S}}$ that define the CPDs for this structure. The key ingredient in parameter estimation is the *likelihood function*, the probability of the data given the model. This function measures the extent to which the parameters provide a good explanation of the data. Intuitively, the higher the probability of the data given the model, the better the ability of the model to predict the data. The likelihood of a parameter set is defined to be the probability of the data given the model: $L(\theta_{\mathcal{S}} \mid \mathcal{D}, \mathcal{S}) = P(\mathcal{D} \mid \mathcal{S}, \theta_{\mathcal{S}})$.

As in many cases, it is more convenient to work with the logarithm of this function:

$$
\begin{aligned}
l(\theta_{\mathcal{S}} \mid \mathcal{D}, \mathcal{S}) &= \log P(\mathcal{D} \mid \mathcal{S}, \theta_{\mathcal{S}}) \\
&= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[ \sum_{x \in \mathcal{T}_{\mathcal{D}}[X_i]} \log P(\mathcal{D}_{x.A} \mid \mathcal{D}_{\mathrm{Pa}(x.A)}) \right]. \qquad (1)
\end{aligned}
$$

Here we consider the task of performing *maximum likelihood* parameter estimation. Our goal is to find the parameter setting $\theta_{\mathcal{S}}$ that maximizes the likelihood $L(\theta_{\mathcal{S}} \mid \mathcal{D}, \mathcal{S})$ for a given $\mathcal{D}$, and $\mathcal{S}$. Thus, the maximum likelihood model is the model that best predicts the training data. This estimation is simplified by the *decomposition* of log-likelihood function into a summation of terms corresponding to the various attributes of the different tables. Each of the terms in the square brackets in (1) can be maximized independently of the rest. Hence, maximum likelihood estimation reduces to independent maximization problems, one for each CPD. In fact, a little further work reduces Eq. (1) even further, to a sum of terms, one for each multinomial distribution $\theta_{R.A|\mathbf{u}}$. Furthermore, there is a closed form solution for the parameter estimates. In addition, while we do not describe the details here, we can take a *Bayesian approach* to parameter estimation by incorporating parameter priors. For an appropriate form of the prior and by making standard assumptions, we can also get a closed form solution for the estimates.

## 5.2 Structure Learning

We now move to the more challenging problem of learning a dependency structure automatically. The main problem here is finding a good dependency structure among the huge number of many possible ones. As in most learning algorithms, there are three important components that need to be defined: the **hypothesis space** which specifies which structures are candidate hypotheses that our learning algorithm can return; a **scoring function** that evaluates the “goodness” of different candidate hypotheses relative to the data; and the **search algorithm**, a procedure that searches the hypothesis space for a structure with a high score. We discuss each of these in turn.

### 5.2.1 Hypothesis Space

Fundamentally, our hypothesis space is determined by our representation language: a hypothesis specifies a set of parents for each attribute $R.A$. We must, however, restrict our hypothesis space to ensure that the structure we are learning is a legal one. We want to ensure that the structure we are learning will generate a consistent probability model for any skeleton or query we are likely to see. We can do this by constructing a dependency graph for the candidate structure and ensuring that the class graph is acyclic. We maintain the graph during learning, and consider only models whose dependency structure passes the appropriate test; see [7, 8] for more details.

### 5.2.2 Scoring: Generalization

The second key component is the ability to evaluate different structures in order to pick one that fits the data well. We adapt Bayesian *model selection* methods to our framework. Bayesian model selection utilizes a probabilistic scoring function. In line with the Bayesian philosophy, it ascribes a prior probability distribution over any aspect of the model about which we are uncertain. In this case, we have a prior $P(\mathcal{S})$ over structures, and a prior $P(\theta_{\mathcal{S}} \mid \mathcal{S})$ over the parameters given each possible structure. The *Bayesian score* of a structure $\mathcal{S}$ is defined as the *posterior* probability of the structure given the data $\mathcal{D}$. Formally, using Bayes rule, we have that:

$$P(\mathcal{S} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \mathcal{S}) P(\mathcal{S})$$

where the denominator, which is the marginal probability $P(\mathcal{D})$ is a normalizing constant that does not change the relative rankings of different structures.

This score is composed of two main parts: the prior probability of the structure, and the probability of the data given that structure. The marginal likelihood is a crucial component, which has the effect of penalizing models with a large number of parameters. Thus, this score automatically balances the complexity of the structure with its fit to the data. In the case where $\mathcal{D}$ is a complete assignment, and we make certain reasonable assumptions about the structure prior, there is a closed form solution for the score.

### 5.2.3 Scoring: Summarization

In this setting, to provide a formal definition of model quality, we make use of basic concepts from information theory [5]. 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 *log-likelihood function*:

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

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

In the structure selection task we would like to finding the dependency structure that achieves the highest log-likelihood score. Consider a particular structure $\mathcal{S}$. We consider the optimal choice (in terms of likelihood) for parameterizing $\mathcal{S}$. We use $\hat{\theta_{\mathcal{S}}}$ to denote this set of parameters. Let $P_{\mathcal{D}}$ be the distribution in the database, as above. We can reformulate the log-likelihood score in terms of mutual information:

$$l(\mathcal{S}, \hat{\theta_{\mathcal{S}}} \mid \mathcal{D}) = \left[ \sum_i |R_i| \sum_{A \in \mathcal{A}(R_i)} MI_{P_{\mathcal{D}}}(R_i.A; \mathrm{Pa}(R_i.A)) \right] + C$$

where $C$ is a constant that does not depend on the choice of structure. Again, 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 $l(\mathcal{S}, \hat{\theta_{\mathcal{S}}} \mid \mathcal{D})$.

### 5.2.4 Search Algorithm

Now that we have a hypothesis space and a scoring function that allows us to evaluate different hypotheses, we need only provide a procedure for finding a high-scoring hypothesis in our space. For Bayesian networks, we know that the task of finding the highest scoring network is NP-hard [4]. As SSM learning is at least as hard as Bayesian network learning (a Bayesian network is simply a SSM with one table and no relations), we cannot hope to find an efficient procedure that always finds the highest scoring structure. Thus, we must resort to heuristic search.

The simplest heuristic search algorithm is greedy hill-climbing search, using our score as a metric. We maintain our current candidate structure and iteratively improve it. At each iteration, we consider a set of simple local transformations to that structure, score all of them, and pick the one with highest score. We restrict attention to simple transformations such as adding, deleting or reversing an edge. We can show that, as in Bayesian network learning, each of these local changes requires that we recompute only the contribution to the score for the portion of the structure that has changed in this step; this has a significant impact on the computational efficiency of the search algorithm. We stop when we have reached a local optimum in the model space. We can make our search procedure more robust by incorporating any of the standard extensions to greedy search such as random restarts, maintaining a tabu list or using a simulated annealing form of search algorithm.

In the case where we are using the summarization scoring criterion, we alter our search strategy slightly. In our greedy search, we need to consider 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 $\Delta_l(\mathcal{S}', \mathcal{S}) = \text{score}_l(\mathcal{S}' : \mathcal{D}) - \text{score}_l(\mathcal{S} : \mathcal{D})$. However, this approach is very shortsighted, as it ignores the cost of the transformation in terms of increasing the size of the structure relative to the storage constraint we have for the model. Instead, we maximize the gain rate [14].

## 6 Related Work

Other than PRMs (and the related approaches of [23, 26]), the main relational learning approaches are variants of inductive logic programming (ILP). Most ILP approaches are deterministic classification approaches, which do not attempt to model a probability distribution, but rather only to predict (classify) a particular predicate. However, there are two recent developments within the ILP community that are related to PRMs: stochastic logic programs (SLPs)[21, 6] and Bayesian logic programs (BLPs)[17, 18]. The semantics for these two approaches are quite different, with the BLP semantics being the closest to PRMs. An SLP defines a sampling distribution over logic programming proofs; as a consequence, it induces a probability distribution over the possible ground facts for a given predicate. On the other hand, a BLP consists of a set of rules, along with conditional probabilities and a combination rule; following the approach of knowledge-based model construction [28], the BLP essentially specifies a propositional Bayesian network. This approach is very similar to the probabilistic logic programs of [23, 26].

Learning algorithms for these approaches are being developed. Methods for learning SLPs are described in [22]. A maximum likelihood approach is taken for parameter estimation for an SLP which is based on maximizing the posterior probability of the program. The task of learning the structure of an SLP is quite different from learning a PRM structure and is based on more traditional ILP approaches. The learning algorithms proposed for BLPs [19] also focus on learning the relational (logical) structure of the domain and do not consider learning the probabilistic dependencies (the combining rules are taken as given). In contrast, the learning algrithms for SSMs focus on learning the dependency structure, assuming that the (potential) relational structure is given.

# 7 Conclusion

We have presented two approaches to structured statistical models: probabilistic relational models and statistical relational models. Experimental results using these models have been presented elsewhere: for PRMs, we have shown that they are useful for classifying web pages and bibliographic citations [10] and for discovering patterns in tuberculosis epidimiology [13]; for SRMs, we have shown that they are useful for selectivity estimation in a variety of realworld domains [14]. Here we have attempted to present a concise summary of the two approaches, leaving out some of the technicals details, in an effort to make the methods more accessible and to highlight the commonalities and differences between the approaches.

# References

[1] F. Bacchus. On probability distributions over possible worlds. In *Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence*, pages 15–21. Morgan Kaufman, 1988.

[2] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* MIT Press, Cambridge, MA, 1990.

[3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 115–123. Morgan Kaufman, August 1996.

[4] D. M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H.-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer Verlag, 1996.

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

[6] J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the Fifteenth Annual Conference on Uncertainty*, pages 126–133, Stockholm, Sweden, 1999. Morgan Kaufman.

[7] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufman.

[8] L. Getoor. *Learning Statistical Models from Relational Data.* PhD thesis, Stanford University, 2001.

[9] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic relational models with link uncertainty. In *Proceedings of the International Conference on Machine Learning*, pages 170—177. Morgan Kaufman, 2001.

[10] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models with link uncertainty. *Journal of Machine Learning Research*, 2003. to appear.

[11] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries*, pages 25–34, 2000.

[12] L. Getoor, D. Koller, and B. Taskar. Statistical models for relational data. In *Proceedings of the KDD-2002 Workshop on Multi-Relational Data Mining*, pages 36–55, Edmonton, CA, 2002. ACM Press.

[13] L. Getoor, J. Rhee, D. Koller, and P. Small. Understianding tuberculosis epidemiology using probabilistic relational models. *Journal of Artificial Intelligence in Medicine*, 2003. to appear.

[14] L. Getoor, B. Taskar, and D. Koller. Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 461—472. ACM Press, 2001.

[15] J. Halpern. An analysis of first-order logics for probability. *Artificial Intelligence*, 46:311–350, 1990.

[16] D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 301–354. MIT Press, Cambridge, MA, 1998.

[17] K. Kersting, L. de Raedt, and S. Kramer. Interpreting Bayesian logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 29–35. AAAI Press, 2000.

[18] K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report No. 151, Institute for Computer Science, University of Freiburg, Germany, 2001.

[19] K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs. Technical Report No. 174, Institute for Computer Science, University of Freiburg, Germany, 2002.

[20] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 580–587, Madison, WI, 1998. AAAI Press.

[21] S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, Amsterdam, 1996.

[22] S.H. Muggleton. Learning stochastic logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 36–41. AAAI Press, 2000.

[23] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1996.

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

[25] A. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, 2000.

[26] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[27] J. Ullman. *Principles of Database and Knowledge-Base Systems, volume 1*. Computer Science Press, 1988.

[28] M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, 1992.