# Learning Probabilistic Relational Models

**Nir Friedman**[*]
Hebrew University
nir@cs.huji.ac.il

**Lise Getoor**[†]
Stanford University
getoor@cs.stanford.edu

**Daphne Koller**[†]
Stanford University
koller@cs.stanford.edu

**Avi Pfeffer**[†]
Stanford University
avi@cs.stanford.edu

## Abstract

A large portion of real-world data is stored in commercial relational database systems. In contrast, most statistical learning methods work only with "flat" data representations. Thus, to apply these methods, we are forced to convert our data into a flat form, thereby losing much of the relational structure present in our database. This paper builds on the recent work on *probabilistic relational models (PRMs)*, and describes how to learn them from databases. PRMs allow the properties of an object to depend probabilistically both on other properties of that object and on properties of *related* objects. Although PRMs are significantly more expressive than standard models, such as Bayesian networks, we show how to extend well-known statistical methods for learning Bayesian networks to learn these models. We describe both *parameter estimation* and *structure learning* — the automatic induction of the dependency structure in a model. Moreover, we show how the learning procedure can exploit standard database retrieval techniques for efficient learning from large datasets. We present experimental results on both real and synthetic relational databases.

## 1 Introduction

Relational models are the most common representation of structured data. Enterprise business information, marketing and sales data, medical records, and scientific datasets are all stored in relational databases. Indeed, relational databases are a multi-billion dollar industry. Recently, there has been growing interest in making more sophisticated use of these huge amounts of data, in particular *mining* these databases for certain patterns and regularities. By explicitly modeling these regularities, we can gain a deeper understanding of our domain and may discover useful relationships. We can also use our model to "fill in" unknown but important information. For example, we may be interested in predicting whether a person is a potential money-launderer based on their bank deposits, international travel, business connections and arrest records of known associates [Jensen, 1997]. In another case, we may be interested in classifying web pages as belonging to a student, a faculty member, a project, etc., using attributes of the web page and of related pages [Craven *et al.*, 1998].

Unfortunately, few inductive learning algorithms are capable of handling data in its relational form. Most are restricted to dealing with a flat set of instances, each with its own separate attributes. To use these methods, one typically "flattens" the relational data, removing its richer structure. This process, however, loses information which might be crucial in understanding the data. Consider, for example, the problem of predicting the value of an attribute of a certain entity, e.g., whether a person is a money-launderer. This attribute will be correlated with other attributes of this entity, as well as with attributes of related entities, e.g., of financial transactions conducted by this person, of other people involved in these transactions, of other transactions conducted by these people, etc. In order to "flatten" this problem, we would need to decide in advance on a fixed set of attributes that the learning algorithm can use in this task. Thus, we want a learning algorithm that can deal with multiple entities and their properties, and can reach conclusions about an entity's characteristics based on the properties of the entities to which it is related. Until now, *inductive logic programming (ILP)* [Lavrač and Džeroski, 1994] has been the primary learning framework with this capability. ILP algorithms learn logical Horn rules for determining when some first-order predicate holds. While ILP is an excellent solution in many settings, it may be inappropriate in others. The main limitation is the deterministic nature of the rules discovered. In many domains, such as the examples above, we encounter interesting correlations that are far from being deterministic.

Our goal in this paper is to learn more refined probabilistic models, that represent statistical correlations both between the properties of an entity and between the properties of related entities. Such a model can then be used for reasoning about an entity using the entire rich structure of knowledge encoded by the relational representation.

The starting point for our work is the structured representation of probabilistic models, as exemplified in Bayesian networks (BNs). A BN allows us to provide a compact rep-

[*] The Institute of Computer Science, Hebrew University, Jerusalem 91904, ISRAEL

[†] Computer Science Department, Stanford University, Gates Building 1A, Stanford CA 94305-9010

resentation of a complex probability distribution over some fixed set of *attributes* or *random variables*. The representation exploits the locality of influence that is present in many domains. We build on two recent developments in the field of Bayesian networks. The first is the deep understanding of the statistical learning problem in such models [Heckerman, 1998; Heckerman *et al.*, 1995] and the role of structure in providing an appropriate bias for the learning task. The second is the recent development of representations that extend the attribute-based BN representation to incorporate a much richer relational structure [Koller and Pfeffer, 1998; Ngo and Haddawy, 1996; Poole, 1993].

In this paper, we combine these two advances. Indeed, one of our key contributions is to show that many of the techniques of Bayesian network learning can be extended to the task of learning these more complex models. This contribution generalizes [Koller and Pfeffer, 1997]'s preliminary work on this topic. We start by describing the semantics of *probabilistic relational models*. We then examine the problems of *parameter estimation* and *structure selection* for this class of models. We deal with some crucial technical issues that distinguish the problem of learning relational probabilistic models from that of learning Bayesian networks. We provide a formulation of the likelihood function appropriate to this setting, and show how it interacts with the standard assumptions of BN learning. The search over coherent dependency structures is significantly more complex than in the case of learning BN structure and we introduce the necessary tools and concepts to do this effectively. We then describe experimental results on synthetic and real-world datasets, and finally discuss possible extensions and applications.

## 2 Underlying framework

### 2.1 Relational model

We describe our relational model in generic terms, closely related to the language of entity-relationship models. This generality allows our framework to be mapped into a variety of specific relational systems, including the probabilistic logic programs of [Ngo and Haddawy, 1996; Poole, 1993], and the probabilistic frame systems of [Koller and Pfeffer, 1998]. Our learning results apply to all of these frameworks.

The vocabulary of a relational model consists of a set of *classes* $X_1, \ldots, X_n$ and a set of *relations* $R_1, \ldots, R_m$. Each entity type is associated with a set of *attributes* $\mathcal{A}(X_i)$. Each attribute $A_j \in \mathcal{A}(X_i)$ takes on values in some fixed domain of values $V(A_j)$. Each relation $R$ is typed. This vocabulary defines a *schema* for our relational model.

Consider a simple genetic model of the inheritance of a single gene that determines a person's blood type. Each person has two copies of the chromosome containing this gene, one inherited from her mother, and one inherited from her father. There is also a possibly contaminated test that attempts to recognize the person's blood type. Our schema contains two classes *Person* and *Blood-Test*, and three relations *Father*, *Mother*, and *Test-of*. Attributes of *Person* are *Name*, *Gender*, *P-Chromosome* (the chromosome inherited from the father), *M-Chromosome* (inherited from the mother). The attributes of *Blood-Test* are *Serial-Number*, *Date*, *Contaminated*, and *Result*.

An *instance* $\mathcal{I}$ of a schema defines a set of entities $\mathcal{O}^{\mathcal{I}}(X_i)$ for each entity type $X_i$. For each entity $x \in \mathcal{O}^{\mathcal{I}}(X_i)$, and each attribute $A_j \in \mathcal{A}(X_i)$, the instance has an associated attribute $x.a_j$; its value in $\mathcal{I}$ is denoted $\mathcal{I}_{x.a_j}$. For each relation $R(X_1, \ldots, X_k)$ and each $\langle x_1, \ldots, x_k \rangle \in \mathcal{O}^{\mathcal{I}}(X_1) \times \cdots \times \mathcal{O}^{\mathcal{I}}(X_k)$, $\mathcal{I}$ specifies whether $R(x_1, \ldots, x_k)$ holds.

We are interested in describing a probability model over instances of a relational schema. However, some attributes, such as a name or social security number, are fully determined. We label such attributes as *fixed*. We assume that they are known in any instantiation of the schema. The other attributes are called *probabilistic*. A *skeleton structure* $\sigma$ of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects $\mathcal{O}^{\sigma}(X_i)$ for each class, the values of the fixed attributes of these objects, and the relations that hold between the objects. However, it leaves the values of probabilistic attributes unspecified. A *completion* $\mathcal{I}$ of the skeleton structure $\sigma$ extends the skeleton by also specifying the values of the probabilistic attributes.

One final definition which will turn out to be useful is the notion of a *slot chain*. If $R(X_1, \ldots, X_k)$ is any relation, we can project $R$ onto its $i$-th and $j$-th arguments to obtain a binary relation $\rho(X_i, X_j)$, which we can then view as a *slot* of $X_i$. For any $x$ in $X_i$, we let $x.\rho$ denote all the elements $y$ in $X_j$ such that $\rho(x, y)$ holds. (In relational algebra notation $x.\rho = \Pi_{X_j} \sigma_{X_i = x} R$.) Objects in this set are called *$\rho$-relatives* of $x$. We can concatenate slots to form longer *slot chains* $\tau = \rho_1. \cdots .\rho_m$, defined by composition of binary relations. (Each of the $\rho_i$'s in the chain must be appropriately typed.)

### 2.2 Probabilistic Relational Models

We now proceed to the definition of probabilistic relational models (PRMs). The basic goal here is to model our uncertainty about the values of the non-fixed, or probabilistic, attributes of the objects in our domain of discourse. In other words, given a skeleton structure, we want to define a probability distribution over all completions of the skeleton.

Our probabilistic model consists of two components: the qualitative dependency structure, $\mathcal{S}$, and the parameters associated with it, $\theta_{\mathcal{S}}$. The dependency structure is defined by associating with each attribute $X.A$ a set of *parents* $\mathrm{Pa}(X.A)$. These correspond to *formal* parents; they will be instantiated in different ways for different objects in $X$. Intuitively, the parents are attributes that are "direct influences" on $X.A$.

We distinguish between two types of formal parents. The attribute $X.A$ can depend on another probabilistic attribute $B$ of $X$. This formal dependence induces a corresponding dependency for individual objects: for any object $x$ in $\mathcal{O}^{\sigma}(X)$, $x.a$ will depend probabilistically on $x.b$. The attribute $X.A$ can also depend on attributes of related objects $X.\tau.B$, where $\tau$ is a slot chain. To understand the semantics of this formal dependence for an individual object $x$, recall that $x.\tau$ represents the *set* of objects that are $\tau$-relatives of $x$. Except in cases where the slot chain is guaranteed to be single-valued, we must specify the probabilistic dependence of $x.a$ on the multiset $\{y.b \ : \ y \in x.\tau\}$. The notion of *aggregation* from database theory gives us precisely the right tool to address

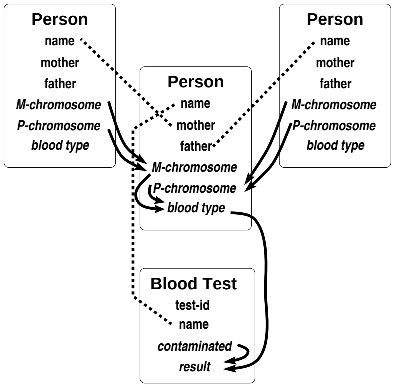

Figure 1: The PRM structure for a simple genetics domain. Fixed attributes are shown in regular font and probabilistic attributes are shown in italic. Dotted lines indicate relations between entities and solid arrows indicate probabilistic dependencies.

this issue; i.e., $x.a$ will depend probabilistically on some aggregate property of this multiset. There are many natural and useful notions of aggregation: the mode of the set (most frequently occurring value); mean value of the set (if values are numerical); median, maximum, or minimum (if values are ordered); cardinality of the set; etc.

More formally, our language allows a notion of an aggregate $\gamma$; $\gamma$ takes a multiset 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 multiset. We allow $X.A$ to have as a parent $\gamma(X.\tau.B)$; the semantics is that for any $x \in X$, $x.a$ will depend on the value of $\gamma(x.\tau.b)$. We define $V(\gamma(X.\tau.b))$ in the obvious way.

Returning to our genetics example, consider the attribute *Blood-Test.Result*. Since the result of a blood test depends on whether it was contaminated, it has *Blood-Test.Contaminated* as a parent. The result also depends on the genetic material of the person tested. Since *Test-of* is single-valued, we add *Blood-Test.Test-of.M-Chromosome* and *Blood-Test.Test-of.P-Chromosome* as parents. Figure 1 shows the structure of a simple PRM for this domain.

Given a set of parents $\text{Pa}(X.A)$ for $X.A$, we can define a local probability model for $X.A$. We associate $X.A$ with a *conditional probability distribution (CPD)* that specifies $P(X.A \mid \text{Pa}(X.A))$. More precisely, let $\mathbf{U}$ be the set of parents of $X.A$. Recall that each of these parents $U_i$ — whether a simple attribute in the same relation or an aggregate of a set of $\tau$ relatives — has a set of values $V(U_i)$ in some ground type. For each tuple of values $\mathbf{u} \in V(\mathbf{U})$, the CPD specifies a distribution $P(X.A \mid \mathbf{u})$ over $V(X.A)$. The parameters in all of these CPDs comprise $\theta_{\mathcal{S}}$.

Given a skeleton structure for our schema, we want to use these local probability models to define a probability distribution over completions of the skeleton. First, note that the skeleton determines the set of objects in our model. We associate a random variable $x.a$ with each probabilistic attribute $A$ of each object $x$. The skeleton also determines the relations between objects, and thereby the set of $\tau$-relatives associated with every object for each relationship chain $\tau$. Also note that by assuming that the relations between objects are always specified by $\sigma$, we are disallowing uncertainty over the relational structure of the model.

To define a coherent probabilistic model over this skeleton, we must ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. Consider the parents of an attribute $X.A$. When $X.B$ is a parent of $X.A$, we define an edge $x.b \rightarrow_\sigma x.a$; when $\gamma(X.\tau.B)$ is a parent of $X.A$ and $y \in x.\tau$, we define an edge $y.b \rightarrow_\sigma x.a$. We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to a skeleton $\sigma$ if the directed graph defined by $\rightarrow_\sigma$ over the variables $x.a$ is acyclic. In this case, we can define a coherent probabilistic model over complete instantiations $\mathcal{I}$ consistent with $\sigma$:

$$P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) = \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \mathcal{O}^{\sigma}(X_i)} P(\mathcal{I}_{x.a} \mid \mathcal{I}_{\text{Pa}(x.a)}) \qquad (1)$$

**Proposition 2.1:** *If $\mathcal{S}$ is acyclic relative to $\sigma$, then (1) defines a distribution over completions $\mathcal{I}$ of $\sigma$.*

We briefly sketch a proof of this proposition, by showing how to construct a BN over the probabilistic attributes of a skeleton using $(\mathcal{S}, \theta_{\mathcal{S}})$. This construction is reminiscent of the *knowledge-based model construction* approach [Wellman *et al.*, 1992]. Here, however, the construction is merely a thought-experiment; our learning algorithm never constructs this network. In this network there is a node for each variable $x.a$ and for aggregate quantities required by parents. The parents of these aggregate random variables are all of the attributes that participate in the aggregation, according to the relations specified by $\sigma$. The CPDs of random variables that correspond to probabilistic attributes are simply the CPDs described by $\theta_{\mathcal{S}}$, and the CPDs of random variables that correspond to aggregate nodes capture the deterministic function of the particular aggregate operator. It is easy to verify that if the probabilistic dependencies are acyclic, then so is the induced Bayesian network. This construction also suggests one way of answering queries about a relational model. We can "compile" the corresponding Bayesian network and use standard tools for answering queries about it.

Although for each skeleton, we can compile a PRM into a Bayesian network, a PRM expresses much more information than the resulting BN. A BN defines a probability distribution over a fixed set of attributes. A PRM specifies a distribution over *any* skeleton; in different skeletons, the set (and number) of entities in the domain will vary, as will the relations between the entities. In a way, PRMs are to BNs as a set of rules in first-order logic is to a set of rules in propositional logic: A rule such as $\forall x, y, z.\textit{Parent}(x, y) \wedge \textit{Parent}(y, z) \Rightarrow \textit{Grandparent}(x, z)$ induces a potentially infinite set of ground (propositional) instantiations.

## 3 Parameter Estimation

We now move to the task of learning PRMs. We begin with learning the parameters for a PRM where the dependency structure is known. In other words, 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. Our learning is based on a particular training set, which we will take to be a complete instance $\mathcal{I}$. While this task is relatively straightforward, it is of interest in and of itself. In addition, it is a crucial component in the structure learning algorithm described in the next section.

The key ingredient in parameter estimation is the *likelihood function*, the probability of the data given the model. This function captures the response of the probability distribution to changes in the parameters. As usual, the likelihood of a parameter set is defined to be the probability of the data given the model: $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) = P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}})$. As usual, we typically work with the log of this function:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \log P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[ \sum_{x \in \mathcal{O}^\sigma(X_i)} \log P(\mathcal{I}_{x.a} \mid \mathcal{I}_{\mathrm{Pa}(x.a)}) \right] . (2) \end{aligned}$$

The key insight is that this equation is very similar to the log-likelihood of data given a Bayesian network [Heckerman, 1998]. In fact, it is the likelihood function of the Bayesian network induced by the structure given the skeleton. The main difference from standard Bayesian network parameter learning is that parameters for different nodes in the network are forced to be identical. Thus, we can use the well-understood theory of learning from Bayesian networks. Consider the task of performing *maximum likelihood* parameter estimation. Here, our goal is to find the parameter setting $\theta_{\mathcal{S}}$ that maximizes the likelihood $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S})$ for a given $\mathcal{I}$, $\sigma$ and $\mathcal{S}$. This estimation is simplified by the *decomposition* of log-likelihood function into a summation of terms corresponding to the various attributes of the different classes. Each of the terms in the square brackets in (2) can be maximized independently of the rest. Hence, maximal likelihood estimation reduces to independent maximization problems, one for each CPD.

For multinomial CPDs, maximum likelihood estimation can be done via *sufficient statistics* which in this case are just the counts $C_{X.A}[v, \mathbf{u}]$ of the different values $v, \mathbf{u}$ that the attribute $X.A$ and its parents can jointly take.

**Proposition 3.1:** *Assuming multinomial CPDs, the maximum likelihood parameter setting $\hat{\theta}_{\mathcal{S}}$ is*

$$P(X.A = v \mid Pa(X.A) = \mathbf{u}) \quad = \quad \frac{C_{X.A}[v, \mathbf{u}]}{\sum_{v'} C_{X.A}[v', \mathbf{u}]}$$

As a consequence of this proposition, parameter learning in PRMs is reduced to *counting* sufficient statistics. We need to count one vector of sufficient statistics for each CPD. Such counting can be done in a straightforward manner using standard databases queries.

Note that this proposition shows that learning parameters in PRMs is very similar to learning parameters in Bayesian networks. In fact, we might view this as learning parameters for the BN that the PRM induces given the skeleton. However, as discussed above, the learned parameters can then be used for reasoning about other skeletons, which induce a completely different BN.

In many cases, maximum likelihood parameter estimation is not robust, as it overfits the training data. The Bayesian approach uses a prior distribution over the parameters to smooth the irregularities in the training data, and is therefore significantly more robust. As we will see in Section 4.2, the Bayesian framework also gives us a good metric for evaluating the quality of different candidate structures. Due to space limitations, we only briefly describe this alternative approach.

Roughly speaking, the Bayesian approach introduces a prior over the unknown parameters, and performs Bayesian conditioning, using the data as evidence, to compute a posterior distribution over these parameters. To apply this idea in our setting, recall that the PRM parameters $\theta_{\mathcal{S}}$ are composed of a set of individual probability distribution $\theta_{X.A|\boldsymbol{v}}$ for each conditional distribution of the form $P(X.A \mid \mathrm{Pa}(X.A) = \mathbf{u})$. Following the work on Bayesian approaches for learning Bayesian networks [Heckerman, 1998], we make two assumptions. First, we assume *parameter independence*: the priors over the parameters $\theta_{X.A|\boldsymbol{v}}$ for the different $X, A$ and $\boldsymbol{v}$ are independent. Second, we assume that the prior over $\theta_{X.A|\boldsymbol{v}}$ is a *Dirichlet* distribution. Briefly, a Dirichlet prior for a multinomial distribution of a variable $W$ is specified by a set of *hyperparameters* $\{\alpha[w] : w \in v(W)\}$. A distribution on the parameters of $P(W)$ is Dirichlet if $\Pr(\theta_W) \propto \prod_w \theta_w^{\alpha[w]}$. (For more details see [DeGroot, 1970].)

For a parameter prior satisfying these two assumptions, the posterior also has this form. That is, it is a product of independent Dirichlet distributions over the parameters $\theta_{X.A|\boldsymbol{v}}$, which can be computed easily.

**Proposition 3.2:** *If $\mathcal{I}$ is a complete assignment, and the prior satisfies parameter independence and Dirichlet with hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$, then the posterior $P(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S})$ is a product of Dirichlet distributions with hyperparameters $\alpha'_{X.A}[v, \mathbf{u}] = \alpha_{X.A}[v, \mathbf{u}] + C_{X.A}[v, \mathbf{u}]$.*

Once we have updated the posterior, how do we evaluate the probability of new data? In the case of BN learning, we assume that instances are IID, which implies that they are independent given the value of the parameters. Thus, to evaluate a new instance, we only need the posterior over the parameters. The probability of the new instance is then the probability given every possible parameter value, weighted by the posterior probability over these values. In the case of BNs, this term can be rewritten simply as the instance probability according to the *expected* value of the parameters (i.e., the mean of the posterior Dirichlet for each parameter). This suggests that we might use the expected parameters for evaluating new data. Indeed, the formula for the expected parameters is analogous to the one for BNs:

**Proposition 3.3 :** *Assuming multinomial CPDs, prior independence, and Dirichlet priors, with hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$, we have that:*

$$E[P(X.A = v \mid Pa(X.A) = \mathbf{u}) \mid \mathcal{I}] = \frac{C_{X.A}[v, \mathbf{u}] + \alpha_{X.A}[v, \mathbf{u}]}{\sum_{v'} C_{X.A}[v', \mathbf{u}] + \alpha_{X.A}[v', \mathbf{u}]}$$

Unfortunately, the expected parameters are not the proper Bayesian solution for computing probability of new data. There are two possible complications.

The first problem is that, in our setting, the assumption of IID data is often violated. Specifically, a new instance might not be conditionally independent of old ones given the parameters. Consider the genetics domain, and assume that our new data involves information about the mother $x'$ of some person $x$ already in the database. In this case, the introduction of the new object $x'$ also changes our probability about the attributes of $x'$. We therefore cannot simply use our old posterior about the parameters to reason about the new instance. This problem does not occur if the new data is not related to the training data, that is, when the new data is essentially a disjoint database with the same scheme. More interestingly, the problem also disappears when attributes of new objects are not parents of any attribute in the training set. In the genetics example, this means that we can insert new people into our database, as long as they are not ancestors of people already in the database.

The second problem involves the formal justification for using expected parameters values. This argument depends on the fact that the probability of a new instance is linear in the value of each parameter. That is, each parameter is "used" at most once. This assumption is violated when we consider the probability of a complex database involving multiple instances from the same class. In this case, our integral of the probability of the new data given the parameters can no longer be reduced to computing the probability relative to the expected parameter value. The correct expression is called the *marginal likelihood* of the (new) data; we use it in Section 4.2 for scoring structures. For now, we note that if the posterior is sharply peaked (i.e., we have seen many training instances), we can approximate this term by using the expected parameters of Proposition 3.3, as we could for a single instance. In practice, we will often use these expected parameters as our learned model.

## 4 Structure selection

We now move to the more challenging problem of learning a dependency structure automatically, as opposed to having it given by the user. There are three important issues that need to be addressed. We must determine which dependency structures are legal; we need to evaluate the "goodness" of different candidate structures; and we need to define an effective search procedure that finds a good structure.

### 4.1 Legal structures

When we consider different dependency structures, it is important to be sure that the dependency structure $\mathcal{S}$ we choose results in coherent probability models. To guarantee this property, we see from Proposition 2.1 that the skeleton $\sigma$ must be acyclic relative to $\mathcal{S}$. Of course, we can easily verify for a given candidate structure $\mathcal{S}$ that it is acyclic relative to the skeleton $\sigma$ of our training database. However, we also want to guarantee that it will be acyclic relative to other databases that we may encounter in our domain. How do we guarantee acyclicity for an arbitrary database? A simple approach is to ensure that dependencies among attributes respect some order (i.e., are stratified). More precisely, we say that $X.A$ *directly depends* on $Y.B$ if either (a) $X = Y$ and $X.B$ is a parent of $X.A$, or (b) $\gamma(X.\tau.B)$ is a parent of $X.A$ and the $\tau$-relatives of $X$ are of class $Y$. We then require that $X.A$ directly depends only on attributes that precede it in the order.

While this simple approach clearly ensures acyclicity, it is too limited to cover many important cases. Consider again our genetic model. Here, the genotype of a person depends on the genotype of her parents; thus, we have *Person.P-Chromosome* depending directly on *Person.P-Chromosome*, which clearly violates the requirements of our simple approach. In this model, the apparent cyclicity at the attribute level is resolved at the level of individual objects, as a person cannot be his/her own ancestor. That is, the resolution of acyclicity relies on some prior knowledge that we have about the domain. To allow our learning algorithm to deal with dependency models such as this we must allow the user to give our algorithm prior knowledge. We allow the user to assert that certain slots $\mathcal{R}_{ga} = \{\rho_1, \ldots, \rho_k\}$ are *guaranteed acyclic*; i.e., we are guaranteed that there is a partial ordering $\prec_{ga}$ such that if $y$ is a $\rho$-relative for some $\rho \in \mathcal{R}_{ga}$ of $x$, then $y \prec_{ga} x$. We say that $\tau$ is guaranteed acyclic if each of its components $\rho$'s is guaranteed acyclic.

We use this prior knowledge determine the legality of certain dependency models. We start by building a graph that describes the direct dependencies between the attributes. In this graph, we have a *yellow* edge $X.B \rightarrow X.A$ if $X.B$ is a parent of $X.A$. If $\gamma(X.\tau.B)$ is a parent of $X.A$, we have an edge $Y.B \rightarrow X.A$ which is *green* if $\tau$ is guaranteed acyclic and *red* otherwise. (Note that there might be several edges, of different colors, between two attributes). The intuition is that dependency along green edges relates objects that are ordered by an acyclic order. Thus these edges by themselves or combined with intra-object dependencies (yellow edges) cannot cause a cyclic dependency. We must take care with other dependencies, for which we do not have prior knowledge, as these might form a cycle. This intuition suggests the following definition: A (colored) dependency graph is *stratified* if every cycle in the graph contains at least one green edge and no red edges.

**Proposition 4.1:** *If the colored dependency graph of $\mathcal{S}$ and $\mathcal{R}_{ga}$ is stratified, then for any skeleton $\sigma$ for which the slots in $\mathcal{R}_{ga}$ are jointly acyclic, $\mathcal{S}$ defines a coherent probability distribution over assignments to $\sigma$.*

This notion of stratification generalizes the two special cases we considered above. When we do not have any guaranteed acyclic relations, all the edges in the dependency graph are colored either yellow or red. Thus, the graph is stratified if and only if it is acyclic. In the genetics example, all the relations would be in $\mathcal{R}_{ga}$. Thus, it suffices to check that dependencies within objects (yellow edges) are acyclic.

**Proposition 4.2:** *Stratification of a colored graph can be determined in time linear in the number of edges in the graph.*

We omit the details of the algorithm for lack of space, but it relies on standard graph algorithms. Finally, we note that it is easy to expand this definition of stratification for situations where our prior knowledge involves several sets of guaranteed acyclic relations, each set with its own order (e.g., objects on a grid with a north-south ordering and an east-west ordering). We simply color the graph with several colors, and check that each cycle contains edges with exactly one color other than yellow, except for red.

## 4.2 Evaluating different structures

Now that we know which structures are legal, we need to decide how to evaluate different structures in order to pick one that fits the data well. We adapt Bayesian *model selection* methods to our framework. Formally, we want to compute the posterior probability of a structure $\mathcal{S}$ given an instantiation $\mathcal{I}$. Using Bayes rule we have that $P(\mathcal{S} \mid \mathcal{I}, \sigma) \propto P(\mathcal{I} \mid \mathcal{S}, \sigma)P(\mathcal{S} \mid \sigma)$. This score is composed of two main parts: the prior probability of the structure, and the probability of the data assuming that structure.

The first component is $P(\mathcal{S} \mid \sigma)$, which defines a prior over structures. We assume that the choice of structure is independent of the skeleton, and thus $P(\mathcal{S} \mid \sigma) = P(\mathcal{S})$. In the context of Bayesian networks, we often use a simple uniform prior over possible dependency structures. Unfortunately, this assumption does not work in our setting. The problem is that there may be infinitely many possible structures. In our genetics example, a person's genotype can depend on the genotype of his parents, or of his grandparents, or of his great-grandparents, etc. A simple and natural solution penalizes long indirect slot chains, by having $\log P(\mathcal{S})$ proportional to the sum of the lengths of the chains $\tau$ appearing in $\mathcal{S}$.

The second component is the *marginal likelihood*:

$$P(\mathcal{I} \mid \mathcal{S}, \sigma) = \int P(\mathcal{I} \mid \mathcal{S}, \theta_{\mathcal{S}}, \sigma) P(\theta_{\mathcal{S}} \mid \mathcal{S}) \, d\theta_{\mathcal{S}}$$

If we use a parameter independent Dirichlet prior (as above, this integral decomposes into a product of integrals each of which has a simple closed form solution. (This is a simple generalization of the ideas used in the Bayesian score for Bayesian networks.)

**Proposition 4.3:** *If $\mathcal{I}$ is a complete assignment, and $P(\theta_{\mathcal{S}} \mid \mathcal{S})$ satisfies parameter independence and is Dirichlet with hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$, then, $P(\mathcal{I} \mid \mathcal{S}, \sigma)$, the marginal likelihood of $\mathcal{I}$ given $\mathcal{S}$, is equal to*

$$\prod_i \prod_{A \in \mathcal{A}(X_i)} \prod_{\mathbf{u} \in V(()\mathrm{Pa}(X_i.A))} DM(\{C_{X_i.A}[v, \mathbf{u}]\}, \{\alpha_{X_i.A}[v, \mathbf{u}]\})$$

*where*
$DM(\{C[v]\}, \{\alpha[v]\}) = \frac{\Gamma(\sum_v \alpha[v])}{\Gamma(\sum_v (\alpha[v] + C[v]))} \prod_v \frac{\Gamma(\alpha[v] + C[v])}{\Gamma(\alpha[v])}$,
*and $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ is the* Gamma *function.*

Hence, the marginal likelihood is a product of simple terms, each of which corresponds to a distribution $P(X.A \mid \mathbf{u})$ where $\mathbf{u} \in V(\mathrm{Pa}(X.A))$. Moreover, the term for $P(X.A \mid \mathbf{u})$ depends only on the hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$ and the sufficient statistics $C_{X.A}[v, \mathbf{u}]$ for $v \in V(X.A)$.

The marginal likelihood term is the dominant term in the probability of a structure. It balances the complexity of the structure with its fit to the data. This balance can be made explicitly via the asymptotic relation of the marginal likelihood to explicit penalization, such as the MDL score (see, e.g., [Heckerman, 1998]).

Finally, we note that the Bayesian score requires that we assign a prior over parameter values for each possible structure. Since there are many (perhaps infinitely many) alternative structures, this is a formidable task. In the case of Bayesian networks, there is a class of priors that can be described by a single network [Heckerman *et al.*, 1995]. These priors have the additional property of being *structure equivalent*, that is, they guarantee that the marginal likelihood is the same for structures that are, in some strong sense, equivalent. These notions have not yet been defined for our richer structures, so we defer the issue to future work. Instead, we simply assume that some simple Dirichlet prior (e.g., a uniform one) has been defined for each attribute and parent set.

## 4.3 Structure search

Now that we have a test for determining whether a structure is "legal", and a scoring function that allows us to evaluate different structures, we need only provide a procedure for finding legal high-scoring structures. For Bayesian networks, we know that this task is NP-Hard [Chickering, 1996]. As PRM learning is at least as hard as BN learning (a BN is simply a PRM with one class 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 such 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 deal with local maxima using random restarts.

As in Bayesian networks, the decomposability property of the score has significant impact on the computational efficiency of the search algorithm. First, we decompose the score into a sum of *local scores* corresponding to individual attributes and their parents. Now, if our search algorithm considers a modification to our current structure where the parent set of a single attribute $X.A$ is different, only the component of the score associated with $X.A$ will change. Thus, we need only reevaluate this particular component, leaving the others unchanged; this results in major computational savings.

There are two problems with this simple approach. First, as discussed in the previous section, we have infinitely many possible structures. Second, even the atomic steps of the search are expensive; the process of computing sufficient statistics requires expensive database operations. Even if we restrict the set of candidate structures at each step of the search, we cannot afford to do all the database operations necessary to evaluate all of them.

We propose a heuristic search algorithm that addresses both these issues. At a high level, the algorithm proceeds in phases. At each phase $k$, we have a set of potential parents $\mathit{Pot}_k(X.A)$ for each attribute $X.A$. We then do a standard structure search restricted to the space of structures in which the parents of each $X.A$ are in $\mathit{Pot}_k(X.A)$. The advantage of this approach is that we can precompute the view corresponding to $X.A$, $Pot_k(X.A)$; most of the expensive computations — the joins and the aggregation required in the definition of the parents — are precomputed in these views. The sufficient statistics for any subset of potential parents can easily be derived from this view. The above construction, together with the decomposability of the score, allows the steps of the search (say, greedy hill-climbing) to done very efficiently.

The success of this approach depends on the choice of the potential parents. Clearly, a wrong initial choice can result to poor structures. Following [Friedman *et al.*, 1999], which examines a similar approach in the context of learning Bayesian networks, we propose an iterative approach that starts with some structure (possibly one where each attribute does not have any parents), and select the sets $Pot_k(X.A)$ based on this structure. We then apply the search procedure and get a new, higher scoring, structure. We choose new potential parents based on this new structure and reiterate, stopping when no further improvement is made.

It remains only to discuss the choice of $Pot_k(X.A)$ at the different phases. Perhaps the simplest approach is to begin by setting $Pot_1(X.A)$ to be the set of attributes in $X$. In successive phases, $Pot_{k+1}(X.A)$ would consist of all of $\mathrm{Pa}_k(X.A)$, as well as all attributes that are related to $X$ via slot chains of length $< k$. Of course, these new attributes would require aggregation; we sidestep the issue by predefining possible aggregates for each attribute.

This scheme expands the set of potential parents at each iteration. However, it usually results in large set of potential parents. Thus, we actually use a more refined algorithm that only adds parents to $Pot_{k+1}(X.A)$ if they seem to "add value" beyond $\mathrm{Pa}_k(X.A)$. There are several reasonable ways of evaluating the additional value provided by new parents. Some of these are discussed in [Friedman *et al.*, 1999] in the context of learning Bayesian networks. Their results suggest that we should evaluate a new potential parent by measuring the change of score for the family of $X.A$ if we add the $\gamma(X.\tau.B)$ to its current parents. We then choose the highest scoring of these, as well as the current parents, to be the new set of potential parents. This approach allows us to significantly reduce the size of the potential parent set, and thereby of the resulting view, while being unlikely to cause significant degradation in the quality of the learned model.

## 5 Implementation and experimental results

We implemented our learning algorithm on top of the Postgres object-relational database management system. All required counts were obtained simply through database selection queries, and cached to avoid performing the same query twice. During the search process, we created temporary materialized views corresponding to joins between different relations, and these views were then used for computing the counts.

We tested our proposed learning algorithm on two domains, one real and one synthetic. The two domains have very different characteristics. The first is a movie database[1] that contains three relations: *Movie*, *Actor* and *Appears*, which relates actors to movies in which they played. The database contains about 11000 movies and 7000 actors. While this database has a simple structure, it presents the kind of problems one often encounters when dealing with real data: missing values, large domains for attributes, and inconsistent use of values. The fact that our algorithm was able to deal with this kind of real-world problem is quite promising. Our algorithm learned the model shown in Figure 2(a). This model is reasonable, and close to one that we would consider to be "correct". It learned that the *Genre* of a movie depended on its *Decade* and its film *Process* (color, black & white, technicolor etc.) and that the *Decade* depended on its film *Process*. It also learned an interesting dependency combining all three relations: the *Role-Type* played by an actor in a movie depends on the *Gender* of the actor and the *Genre* of the movie.

The second database, an artificial genetic database similar to the example in this paper, presented quite different challenges. For one thing, the recursive nature of this domain allows arbitrarily complex joins to be defined. In addition, the probabilistic model in this domain is fairly subtle. Each person has three relevant attributes — *P-Chromosome*, *M-Chromosome*, and *BloodType* — all with the same domain and all related somehow to the same attributes of the person's mother and father. The gold standard is the model used to generate the data; the structure of that model was shown earlier in Figure 1. We trained our algorithm on datasets of various sizes ranging up to 800. A data set of size $n$ consisted of a family tree containing $n$ people, with an average of 0.6 blood tests per person. We evaluated our algorithm on a test set of size 10,000. Figure 2(b) shows the log-likelihood of the test set for the learned models. In most cases, our algorithm learned a model with the correct structure, and scored well. However, in a small minority of cases, the algorithm got stuck in local maxima, learning a model with incorrect structure that scored quite poorly. This can be seen in the scatter plots of Figure 2(b) which show that the median log-likelihood of the learned models is quite reasonable, but there are a few outliers. Standard techniques such as random restarts can be used to deal with local maxima.

## 6 Discussion and conclusions

In this paper, we defined a new statistical learning task: learning probabilistic relational models from data. We have shown that many of the ideas from Bayesian network learning carry over to this new task. However, we have also shown that it also raises many new challenges.

Scaling these ideas to large databases is an important issue. We believe that this can be achieved by a closer integration with the technology of database systems, including indices and query optimization. Furthermore, there has been a lot of recent work on extracting information from massive data sets, including work on finding frequently occurring combinations of values for attributes. We believe that these ideas will help significantly in the computation of sufficient statistics.

There are also several important possible extensions to this work. Perhaps the most obvious one is the treatment of missing data and hidden variables. We can extend standard techniques (such as Expectation Maximization for missing data)

[1] Obtained from `http://www-db.stanford.edu/pub/movies/doc.html`

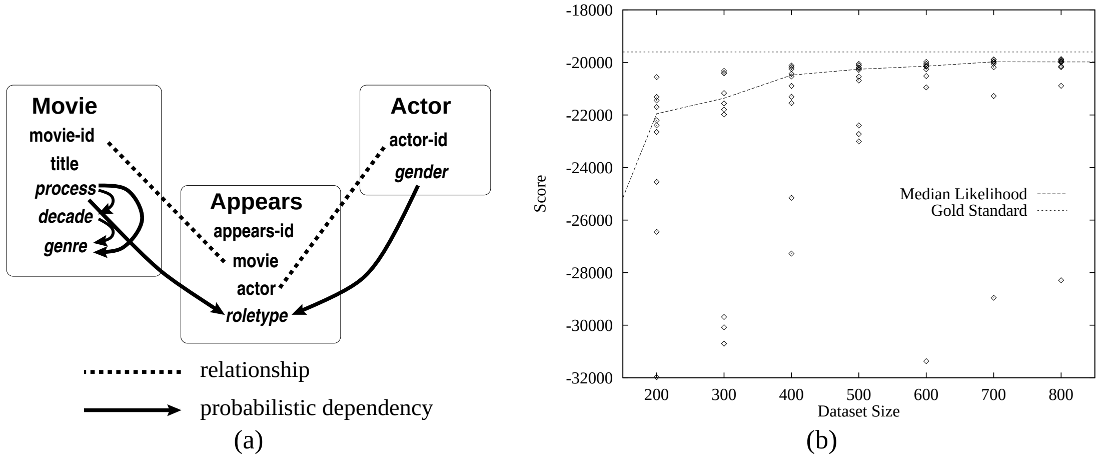

Figure 2: (a) The PRM learned for the movie domain, a real-world database containing about 11000 movies and 7000 actors. (b) Learning curve showing the generalization performance of PRMs learned in the genetic domain. The $x$-axis shows the databases size; the $y$-axis shows log-likelihood of a test set of size 10,000. For each sample size, we show 10 independent learning experiments. The curve shows median log-likelihood of the models as a function of the sample size.

to this task (see [Koller and Pfeffer, 1997] for some preliminary work on related models.) However, the complexity of inference on large databases with many missing values make the cost of a naive application of such algorithms prohibitive. Clearly, this domain calls both for new inference algorithms and for new learning algorithms that avoid repeated calls to inference over these very large problems. Even more interesting is the issue of automated discovery of hidden variables. There are some preliminary answers to this question in the context of Bayesian networks [Friedman, 1997], in the context of ILP [Lavrač and Džeroski, 1994], and very recently in the context of simple binary relations [Hofmann *et al.*, 1998]. Combining these ideas and extending them to this more complex framework is a significant and interesting challenge.

Another direction extends the class of models we consider. Here, we assumed that the relational structure is specified before the probabilistic attribute values are determined. A richer class of PRMs (e.g., that of [Koller and Pfeffer, 1998]) would allow probabilities over the *structure* of the model; for example: uncertainty over the set of objects in the model, e.g., the number of children a couple has, or over the relations between objects, e.g., whose is the blood that was found on a crime scene. Ultimately, we would want these techniques to help us automatically discover interesting entities and relationships that hold in the world.

## Acknowledgments

Nir Friedman was supported by a grant from the Michael Sacher Trust. Lise Getoor, Daphne Koller, and Avi Pfeffer were supported by ONR contract N66001-97-C-8554 under DARPA's HPKB program, by ONR grant N00014-96-1-0718, by the ARO under the MURI program "Integrated Approach to Intelligent Systems," and by the generosity of the Sloan Foundation and of the Powell foundation.

## References

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*. Springer Verlag, 1996.

M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proc. AAAI*, 1998.

M. H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

N. Friedman, I. Nachman, and D. Peér. Learning of Bayesian network structure from massive datasets: The "sparse candidate" algorithm. Submitted, 1999.

N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. ICML*, 1997.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

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

T. Hofmann, J. Puzicha, and M. Jordan. Learning from dyadic data. In *NIPS 12*, 1998. To appear.

D. Jensen. Prospective assessment of ai technologies for fraud detection: A case study. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997.

D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proc. IJCAI*, pages 1316–1321, 1997.

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

N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 1996.

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

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.