Learning Probabilistic Relational Models with Structural Uncertainty

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

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

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

Nir Friedman
School of Computer Sci. & Eng.
Hebrew University
Jerusalem, 91904, Israel
nir@cs.huji.ac.il

Abstract

Most real-world data is stored in relational form. In contrast, most statistical learning methods, e.g., Bayesian network learning, work only with "flat" data representations, forcing us to convert our data into a form that loses much of the relational structure. The recently introduced framework of *probabilistic relational models* (PRMs) allow us to represent much richer dependency structures, involving multiple entities and the relations between them; they allow the properties of an entity to depend probabilistically on properties of *related* entities. Friedman *et al.* showed how to learn PRMs that model the attribute uncertainty in relational data, and presented techniques for learning both parameters and probabilistic dependency structure for the attributes in a relational model. In this work, we propose methods for handling *structural uncertainty* in PRMs. Structural uncertainty is uncertainty over which entities are related in our domain. We propose two mechanisms for modeling structural uncertainty: *reference uncertainty* and *existence uncertainty*. We describe the appropriate conditions for using each model and present learning algorithms for each. We conclude with some preliminary experimental results comparing and contrasting the use of these mechanism for learning PRMs in domains with structural uncertainty.

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. Efforts to extract knowledge from partially structured (e.g., XML) or even raw text data also aim to extract relational information. Recently, there has been growing interest in extracting interesting statistical patterns from these huge amounts of data. These patterns give us a deeper understanding of our domain and the relationships in it. A model can also be used for reaching conclusions about important attributes whose values may be unobserved.

Probabilistic graphical models, and particularly Bayesian networks, have been shown to be a useful way of representing statistical patterns in real-world domains. Recent work (Cooper and Herskovits 1992; Heckerman 1998) develops techniques for learning these models directly from data, and shows that interesting patterns often emerge in this learning process. However, all of these learning techniques apply only to flat-file representations of the data, and not to the richer relational data encountered in many applications.

Probabilistic relational models are a recent development (Koller and Pfeffer 1998; Poole 1993; Ngo and Haddawy 1996) that extend the standard attribute-based Bayesian network representation to incorporate a much richer relational structure. These models allow the specification of a probability model for *classes* of objects rather than simple attributes; they also allow properties of an entity to depend probabilistically on properties of other *related* entities. The model represents a generic dependence, which is then instantiated for specific circumstances, i.e., for particular sets of entities and relations between them.

Friedman *et al.* (Friedman *et al.* 1999) adapt the machinery for learning Bayesian networks from flat data to the task of learning PRMs from structured data. This machinery allows a PRM to be learned from a relational database, an object-oriented database or a frame-based system. This previous work assumes that the relational structure of the domain is determined outside the probabilistic model. Thus, it does not consider the problem of learning a model of the *relational* structure. This assumption has two main implications. Most obviously, it implies that the model can only be used in settings where the relational structure is known. Thus, for example, we cannot use it to conclude that there exists a money-laundering relation between a bank and a drug cartel. A more subtle point is that it can diminish the quality of our model even in cases where the relational structure *is* given, because it ignores interesting correlations between attributes of entities and the relations between them. For example, a "serious" actor is unlikely to appear in many horror movies; hence, we can infer information about an actor's (unknown) attributes based on the Role relation between actors and movies.

In this paper, we examine the problem of learning a probabilistic model of the relational structure. This concept, called *structural uncertainty*, was first introduced by Koller and Pfeffer in (Koller and Pfeffer 1998). They introduced two types of structural uncertainty: *number uncertainty* — uncertainty about the number of individuals to which a particular individual x is related and *reference uncertainty* —

uncertainty about the identity of the individuals to which x is related. In this paper, we provide a substantial extension of reference uncertainty, which makes it suitable for a learning framework; we also introduce a new type of structural uncertainty, called *existence uncertainty*. We present a framework for learning these models from a relational database.

We begin by reviewing the definition of a probabilistic relational model. We then describe two ways of extending the definition to accommodate structural uncertainty. Next we describe learning methods for these models. We conclude with some experimental results, comparing the models learned under the alternate methods.

Probabilistic Relational Models

A *probabilistic relational model (PRM)* specifies a template for a probability distribution over a database. The template includes a relational component, that describes the relational schema for our domain, and a probabilistic component, that describes the probabilistic dependencies that hold in our domain. A PRM, together with a particular database of objects and relations, defines a probability distribution over the attributes of the objects and the relations.

Relational Schema

A schema for a relational model describes a set of *classes*, $\mathcal{X} = X_1, \ldots, X_n$. Each class is associated with a set of *descriptive attributes* and a set of *reference slots* [1].

The set of descriptive attributes of a class X is denoted $\mathcal{A}(X)$. Attribute A of class X is denoted $X.A$, and its domain of values is denoted $V(X.A)$. We assume here that domains are finite. For example, the Person class might have the descriptive attributes *Sex*, *Age*, *Height*, *Income-Level*, etc. The domain for *Person.Age* might be $\{$child, young-adult, middle-aged, senior$\}$.

The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use similar notation, $X.\rho$, to denote the reference slot ρ of X. Each reference slot ρ is typed, i.e., the schema specifies the range type of object that may be referenced. More formally, for each ρ in X, the domain type of $\mathrm{Dom}[\rho] = X$ and the range type $\mathrm{Range}[\rho] = Y$, where Y is some class in $\mathcal{X}$. A slot ρ denotes a function from $\mathrm{Dom}[\rho] = X$ to $\mathrm{Range}[\rho] = Y$. For example, we might have a class Movie with the reference slot *Actor* whose range is the class Actor. Or, the class Person might have reference slots *Father* and *Mother* whose range type is also the Person class. In certain cases, if we want to explicitly specify the domain class of a particular slot, we will use the notation $Actor_{\mathsf{Role}}$.

For each reference slot ρ, we can define an *inverse slot* ρ^{-1}, which is interpreted as the inverse function of ρ. Finally, we define the notion of a *slot chain*, which allows us to compose slots, defining functions from objects to other objects to which they are not directly related. More precisely, we define a *slot chain* $\rho_1, \ldots, \rho_k$ be a sequence of slots (inverse or otherwise) such that for all i, $\mathrm{Range}[\rho_i] = \mathrm{Dom}[\rho_{i+1}]$. For example, Person.Mother.Mother can be used to denote a person's maternal grandmother, and $\mathsf{Person}.Actor_{\mathsf{Role}}{}^{-1}$ can be used to denote the set of roles that an actor has had.

It is often useful to distinguish between an *entity* and a *relationship*, as in entity-relationship diagrams. In our language, classes are used to represent both entities and relationships. Thus, entities such as actors and movies are represented by classes, but a relationship such as Role, which relates actors to movies, is also be represented as a class, with reference slots to the class Actor and the class Movie. This approach, which blurs the distinction between entities and relationships, is common, and allows us to accommodate descriptive attributes that are associated with the relation, such as *Role-Type* (which might describe the type of role, such as villain, heroine or extra). We use $\mathcal{X}_{\mathcal{E}}$ to denote the set of classes that represent entities, and $\mathcal{X}_{\mathcal{R}}$ to denote those that represent relationships. (We note that the distinctions are prior knowledge about the domain, and are therefore part of the domain specification.) We use the generic term *object* to refer both to entities and to relationships.

The semantics of this language is straightforward. In an instantiation $\mathcal{I}$, each X is associated with a set of objects $\mathcal{O}^{\mathcal{I}}(X)$. For each attribute $A \in \mathcal{A}(X)$ and each $x \in \mathcal{O}^{\mathcal{I}}(X)$, $\mathcal{I}$ specifies a value $x.A \in V(X.A)$. For each reference slot $\rho \in \mathcal{R}(X)$ $\mathcal{I}$ specifies a value $x.\rho \in \mathcal{O}^{\mathcal{I}}(\mathrm{Range}[\rho])$. For $y \in \mathcal{O}^{\mathcal{I}}(\mathrm{Range}[\rho])$, we use $y.\rho^{-1}$ to denote the set of entities $\{x \in \mathcal{O}^{\mathcal{I}}(X) \ : \ x.\rho = y\}$. The semantics of a slot chain $\tau = \rho_1.\ldots.\rho_k$ are defined via straightforward composition. For $A \in \mathcal{A}(\mathrm{Range}[\rho_k])$ and $x \in \mathcal{O}^{\mathcal{I}}(X)$, we define $x.\tau.A$ to be the *multiset* of values $y.A$ for y in the set $x.\tau$.

Thus, an instantiation $\mathcal{I}$ is a set of objects with no missing values and no dangling references. It describes the set of objects, the relationships that hold between the objects and all the values of the attributes of the objects. For example, we might have a database containing movie information, with entities Movie, Actor and Role, which includes the information for all the Movies produced in a particular year by some studio. In a very small studio, we might encounter the instantiation shown in Figure 1. For a particular actor a, the set of movies the actor has appeared in would be denoted $a.Actor_{\mathsf{Role}}{}^{-1}.Movie$; for example, $ginger.Actor_{\mathsf{Role}}{}^{-1}.Movie = \{m1, m2\}$.

As discussed in the introduction, our goal in this paper is to construct probabilistic models over instantiations. We shall consider various classes of probabilistic models, which vary in the amount of "prior" specification on which the model is based. This specification, a form of "skeleton" of the domain, defines a set of possible instantiations. Our model then defines a probability distribution over this set. Thus, we now define the necessary building blocks which we use to describe such sets of possible instantiations.

An *entity skeleton*, σ_e, specifies a set of entities $\mathcal{O}^{\sigma_e}(X)$ for each class $X \in \mathcal{X}_{\mathcal{E}}$. Our possible instantiations are

[1] We note that there is a direct mapping between our notion of class and the tables used in a relational database. Our descriptive attributes correspond to standard attributes in the table, and our reference slots correspond to attributes that are foreign keys (key attributes of another table).

ACTOR	
name	gender
fred	male
ginger	female
bing	male

MOVIE	
name	genre
m1	drama
m2	comedy

ROLE			
role	movie	actor	role-type
r1	m1	fred	hero
r2	m1	ginger	heroine
r3	m1	bing	villain
r4	m2	bing	hero
r5	m2	ginger	love-interest

Figure 1: An instantiation of the relational schema for a simple movie domain.

then only those $\mathcal{I}$ for which $\mathcal{O}^{\sigma_e}(X) = \mathcal{O}^{\mathcal{I}}(X)$ for each such class X. In our example above, the associated entity skeleton would specify the set of movies and actors in the database: $\mathcal{O}^{\sigma_e}(\mathsf{Actor}) = \{\textit{fred}, \textit{ginger}, \textit{bing}\}$ and $\mathcal{O}^{\sigma_e}(\mathsf{Movie}) = \{m1, m2\}$. The *object skeleton* is a richer structure; it specifies a set of objects $\mathcal{O}^{\sigma_o}(X)$ for each class $X \in \mathcal{X}$. In our example, the object skeleton consistent with $\mathcal{I}$ would specify the same information as the entity skeleton, as well as the fact that $\mathcal{O}^{\sigma_o}(\mathsf{Role}) = \{r1, r2, r3, r4, r5\}$. Note that this information tells us only the unique identifier, or *key*, of the different objects, but not how they relate. In effect, in both the entity and object skeletons, we are told only the cardinality of the various classes. Finally, the *relational skeleton*, σ_r, contains substantially more information. It specifies the set of objects in all classes, as well as all the relationships that hold between them. In other words, it specifies $\mathcal{O}^{\sigma}(X)$ for each X, and for each object $x \in \mathcal{O}^{\sigma}(X)$, it specifies the values of all of the reference slots $x.\rho$. In our example above, it would provide the values for the actor and movie slots of Role.

Probabilistic Model for Attributes

A probabilistic relational model Π specifies a probability distributions over all instantiations $\mathcal{I}$ of the relational schema. It 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* $\text{Pa}(X.A)$.

A parent of $X.A$ can have the form $X.\tau.B$, for some (possibly empty) slot chain τ. To understand the semantics of this dependence, recall that $x.\tau.A$ is a multiset of values S in $V(X.\tau.A)$. We use the notion of *aggregation* from database theory to define the dependence on a multiset; thus, $x.A$ will depend probabilistically on some aggregate property $\gamma(S)$. There are many natural and useful notions of aggregation; in this paper, we simplify our presentation by focusing on particular notions of aggregation: the *median* for ordinal attributes, and the *mode* (most common value) for others. We allow $X.A$ to have as a parent $\gamma(X.\tau.B)$; for any $x \in X$, $x.A$ will depend on the value of $\gamma(x.\tau.B)$.

The quantitative part of the PRM specifies the parameterization of the model. Given a set of parents for an attribute, we can define a local probability model by associating with it a *conditional probability distribution (CPD)*. For each attribute we have a CPD that specifies $P(X.A \mid \text{Pa}(X.A))$.

Definition 1: A *probabilistic relational model (PRM)* Π for a relational schema $\mathcal{S}$ is defined as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$, we have:

- a set of *parents* $\text{Pa}(X.A) = \{U_1, \ldots, U_l\}$, where each U_i has the form $X.B$ or $X.\tau.B$, where τ is a slot chain;
- a *conditional probability distribution (CPD)* that represents $P_\Pi(X.A \mid \text{Pa}(X.A))$. ∎

Given a relational skeleton σ_r, a PRM Π specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with σ_r:

$$P(\mathcal{I} \mid \sigma_r, \Pi) = \prod_{X} \prod_{x \in \mathcal{O}^{\sigma_r}(X)} \prod_{A \in \mathcal{A}(X)} P(x.A \mid \text{Pa}(x.A)) \tag{1}$$

For this definition to specify a coherent probability distribution over instantiations, we must ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. To verify acyclicity, we construct an *object dependency graph* G_{σ_r}. Nodes in this graph correspond to descriptive attributes of entities. Let $X.\tau.B$ be a parent of $X.A$ in our probabilistic dependency schema; for each $y \in x.\tau$, we define an edge in G_{σ_r}: $y.B \rightarrow_{\sigma_r} x.A$. We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to a relational skeleton σ_r if the directed graph G_{σ_r} is acyclic. When G_{σ_r} is acyclic, we can use the chain rule to ensure that Eq. (1) defines a legal probability distribution (as done, for example, in Bayesian networks).

The definition of the object dependency graph is specific to the particular skeleton at hand: the existence of an edge from $y.B$ to $x.A$ depends on whether $y \in x.\tau$, which in turn depends on the interpretation of the reference slots. Thus, it allows us to determine the coherence of a PRM only relative to a particular relational skeleton. When we are evaluating different possible PRMs as part of our learning algorithm, we want to ensure that the dependency structure $\mathcal{S}$ we choose results in coherent probability models for *any* skeleton. We provide such a guarantee using a *class dependency graph*, which describes all possible dependencies among attributes. In this graph, we have an (intra-object) edge $X.B \rightarrow X.A$ if $X.B$ is a parent of $X.A$. If $\gamma(X.\tau.B)$ is a parent of $X.A$, and $Y = \text{Range}[\tau]$, we have an (inter-object) edge $Y.B \rightarrow X.A$. A dependency graph is *stratified* if it contains no cycles. If the dependency graph of $\mathcal{S}$ is stratified, then it defines a legal model for any relational skeleton σ_r (Friedman *et al.* 1999).

Structural Uncertainty

In the model described in the previous section, all relations between attributes are determined by the relational skeleton σ_r; only the descriptive attributes are uncertain. Thus, Eq. (1) determines the likelihood of the attributes of objects, but does not capture the likelihood of the relations between objects. In this section, we extend our probabilistic model to allow for structural uncertainty. In this scenario, we do not treat the relational structural as fixed. Rather, we treat the relations between objects as an uncertain aspect of the domain. Thus, we will describe a probability distribution over different relational structures. We describe two different dependency models that can represent structural uncertainty: *Reference Uncertainty* and *Existence Uncertainty*. Each is useful in different contexts, and we note that these two models do not exhaust the space of possible models.

Reference Uncertainty

In this model, we assume that the objects are prespecified, but relations among them, i.e., slot chains, are subject to random choices. More precisely, we are given an object skeleton σ_o, which specifies the objects in each class. Now, we need to specify a probabilistic model not only over the descriptive attributes (as above), but also about the value of the reference slots $X.\rho$. The domain of a reference slot $X.\rho$ is the set of keys (unique identifiers) of the objects in class $Y = \text{Range}[\rho]$. Thus, we need to specify a probability distribution over the set of all objects in a class.

A naive approach is to simply have the PRM specify a probability distribution directly as a multinomial distribution over $\mathcal{O}^{\sigma_o}(Y)$. This approach has two major flaws. Most obviously, this multinomial would be infeasibly large, with a parameter for each object in Y. More importantly, we want our dependency model to be general enough to apply over all possible object skeletons σ_o; a distribution defined in terms of the objects within a specific object skeleton would not apply to others.

We achieve a representation which is both general and compact as follows. Roughly speaking, we partition the class Y into subsets according to the values of some of its attributes. For example, we can partition the class Movie by *Genre*. We then assume that the value of $X.\rho$ is chosen by first selecting a partition, and then selecting an object within that partition uniformly. For example, we might assume that a movie theater first selects which genre of movie it wants to show, with a possible bias depending (for example) on the type of theater; it then selects uniformly among the movies with the selected genre.

We formalize this intuition by defining, for each slot ρ, a set of *partition attributes* $\Psi[\rho] \subseteq \mathcal{A}(Y)$. In the above example, $\Psi[\rho] = \{Genre\}$. Essentially, we would like to specify the distribution that the reference value of ρ falls into one partition versus another. We accomplish this within the framework of our current model by introducing S_ρ as a new attribute of X, called a *selector attribute*; it takes on values in the space of possible instantiations $V(\Psi[\rho])$. Each of its possible value s_ψ determines a subset of Y from which the value of ρ (the referent) will be selected. More precisely, each value s_ψ of S_ρ defines a subset Y_ψ of the set of objects $\mathcal{O}^{\sigma_o}(Y)$: those for which the attributes in $\Psi[\rho]$ take the values $\psi[\rho]$. We use $Y_{\Psi[\rho]}$ to represent the resulting partition of $\mathcal{O}^{\sigma_o}(Y)$.

We now represent a probabilistic model over the values of ρ by specifying how likely it is to reference objects in one subset in the partition versus another. For example, a movie theater may be more likely to show an action film rather than a documentary film. We accomplish this by introducing a probabilistic model for the selector attribute S_ρ. This model is the same as that of any other attribute: it has a set of parents and a CPD. Thus, the CPD for S_ρ would specify a probability distribution over possible instantiations s_ψ. As for descriptive attributes, we want to allow the distribution of the slot to depend on other aspects of the domain. For example an independent movie theater may be more likely to show foreign movies while a megaplex may be more likely to show action films. We accomplish this effect by having parents; in our example, the CPD of $S_{\mathsf{Theater}.\textit{Current-Movie}}$ might have as a parent $\mathsf{Theater}.\textit{Type}$.[2] The choice of value for S_ρ determines the partition Y_ψ from which the reference value of ρ is chosen; as discussed above, we assume that the choice of reference value for ρ is uniform distributed within this set.

Definition 2: A *probabilistic relational model* Π *with reference uncertainty* has the same components as in Definition 1. In addition, for each reference slot $\rho \in \mathcal{R}(X)$ with $\text{Range}[\rho] = Y$, we have:

- a set of attributes $\Psi[\rho] \subseteq \mathcal{A}(Y)$;
- a new selector attribute S_ρ within X which takes on values in the cross-product space $V(\Psi[\rho])$;
- a set of parents and a CPD for the new selector attribute, as usual. ∎

To define the semantics of this extension, we must define the probability of reference slots as well as descriptive attributes:

$$
\begin{aligned}
P(\mathcal{I} \mid \sigma_o, \Pi) = \\
&\prod_X \prod_{x \in \mathcal{O}^{\sigma_o}(X)} \prod_{A \in \mathcal{A}(X)} P(x.A \mid \text{Pa}(x.A)) \\
&\prod_{\rho \in \mathcal{R}(X)} \frac{P(x.S_\rho = \psi[x.\rho] \mid \text{Pa}(X.S_\rho))}{|Y_\psi|} \qquad (2)
\end{aligned}
$$

[2]The random variable S_ρ takes on values that are joint assignments to $\Psi[\rho]$. In the current work, we treat this variable as a multinomial random variable over this cross-product space. In general, however, we can represent such a distribution more compactly, e.g., using a Bayesian network. For example, the genre of movies shown by a movie theater might depend on its type (as above). However, the language of the movie can depend on the location of the theater. Thus, the partition will be defined by $\Psi = \{\mathsf{Movie}.\textit{Genre}, \mathsf{Movie}.\textit{Language}\}$, and its parents would be $\mathsf{Theater}.\textit{Type}$ and $\mathsf{Theater}.\textit{Location}$. We can represent this conditional distribution more compactly by introducing a separate variable $S_{\mathsf{Movie}.\textit{Genre}}$, with a parent $\mathsf{Theater}.\textit{Type}$, and another $S_{\mathsf{Movie}.\textit{Language}}$, with a parent $\mathsf{Theater}.\textit{Location}$.

where we take $\psi[x.\rho]$ to refer to the instantiation ψ of the attributes $\Psi[\rho]$ for the object $x.\rho$ in the instantiation $\mathcal{I}$. Note that the last term in Eq. (2) depends on $\mathcal{I}$ in three ways: the interpretation of $x.\rho$, the values of the attributes $\Psi[\rho]$ within the object $x.\rho$, and the size of Y_ψ.

This model gives rise to fairly complex dependencies. Consider a dependency of $X.A$ on $X.\rho.B$. First, note that $x.A$ can depend on $y.B$ for *any* $y \in \text{Range}[\rho]$, depending on the choice of value for $x.\rho$. Thus, the domain dependency graph has a very large number of edges. Second, note that $x.A$ cannot be a parent (or ancestor) of $x.\rho$; otherwise, the value of $x.A$ is used to determine the object referenced by $x.\rho$, and this object in turn affects the value of $x.A$.

As above, we must guarantee that this complex dependency graph is acyclic for every object skeleton. We accomplish this goal by extending our definition of class dependency graph. The graph has a node for each descriptive or selector attribute $X.A$. The graph contains the following edges:

- For any descriptive or selector attribute C, and any of its parents $\gamma(X.\tau.B)$, we introduce an edge from $Y.B$ to $X.G$, where $Y = \text{Range}[\tau]$.
- For any descriptive or selector attribute C, and any of its parents $\gamma(X.\tau.B)$, we add the following edges: for any slot ρ_i along the chain τ, we introduce an edge from $Z.S_{\rho_i}$ to $X.C$, for $Z = \text{Dom}[\rho_i]$.
- For each slot $X.\rho$, and each $Y.B \in \Psi[\rho]$ (for $Y = \text{Range}[\rho]$), we add an edge $Y.B \to X.S_\rho$. This represents the dependence of ρ on the attributes used to partition its range.

The first class of edges in this definition is identical to our the definition of dependency graph above, except that it is extended to deal with selector as well as descriptive attributes. Edges of the second type reflect the fact that the specific value of parent for a node depends on the reference values of the slots in the chain. The third type of edges represent the dependency of a slot on the attributes of the associated partition. To see why this is required, we observe that our choice of reference value for $x.\rho$ depends on the values of the partition attributes $\Psi[X.\rho]$ of all of the different objects in Y. Thus, these attributes must be determined before $x.\rho$ is determined.

Once again, we can show that if this dependency graph is *stratified*, it defines a coherent probabilistic model.

Theorem 3: *Let Π be a PRM with relational uncertainty and stratified dependency graph. Let σ_o be an object skeleton. Then the PRM and σ_o uniquely define a probability distribution over instantiations $\mathcal{I}$ that extend σ_o via Eq. (2).*

Existence Uncertainty

The reference uncertainty model of the previous section assumes that the number of objects is known. Thus, if we consider a division of objects into entities and relations, the number of objects in classes of both types are fixed. Thus, we might need to describe the possible ways of relating 5 movies, 15 actors and 30 roles. The predetermined number of roles might seem a bit artificial, since in some domains it puts an artificial constraint on the relationships between movies and actors. If one movie is a big production and involves many actors, this reduces the number of roles that can be used by other movies.[3]

In this section we consider models where the number of relationship objects is not fixed in advance. Thus, in our example, we will consider all 5×15 possible roles, and determine for each whether it exists in the instantiation. In this case, we are given only the schema and an entity skeleton σ_e. We are not given the set of objects associated with relationship classes. We call the entity classes *determined* and the others *undetermined*. We note that relationship classes typically represent many-many relationships; they have at least two reference slots, which refer to determined classes. For example, our Role class would have reference slots *Actor* to Person and *In-Movie* to Movie. While we know the set of actors and the set of movies, we may be uncertain about which actors have a role in which movie, and thus we have uncertainty over the existence of the Role objects.

Our basic approach in this model is that we allow objects whose existence is uncertain. These are the objects in the undetermined classes. One way of achieving this effect is by introducing into the model all of the entities that can *potentially* exist in it; with each of them we associate a special binary variable that that tells us whether the entity actually exists or not. Note that this construction is purely conceptual; we never explicitly construct a model containing non-existent objects. In our example above, the domain of the Role class in a given instantiation $\mathcal{I}$ is $\mathcal{O}^{\mathcal{I}}(\text{Person}) \times \mathcal{O}^{\mathcal{I}}(\text{Movie})$. Each "potential" object $x = \text{Role}(y_p, y_m)$ in this domain is associated with a binary attribute $x.E$ that specifies whether the person y_p did or did not see movie y_m.

Definition 4: We define an *undetermined* class X as follows. Let $\rho_1, \ldots, \rho_k$ be the set of reference slots of X, and let $Y_i = \text{Range}[\rho_i]$. In any instantiation $\mathcal{I}$, we require that $\mathcal{O}^{\mathcal{I}}(X) = \mathcal{O}^{\mathcal{I}}(Y_1) \times \cdots \mathcal{O}^{\mathcal{I}}(Y_k)$. For $(y_1, \ldots, y_k) \in \mathcal{O}^{\mathcal{I}}(Y_1) \times \cdots \mathcal{O}^{\mathcal{I}}(Y_k)$, we use $X[y_1, \ldots, y_k]$ to denote the corresponding object in X. Each X has a special *existence* attribute $X.E$ whose values are $V(E) = \{\textit{true}, \textit{false}\}$. For uniformity of notation, we introduce an E attribute for all classes; for classes that are determined, the E value is defined to be always *true*. We require that all of the reference slots of a determined class X have a range type which is also a determined class. ▮

The existence attribute for an undetermined class is treated in the same way as a descriptive attribute in our dependency model, in that it can have parents and children, and is associated with a CPD.

[3] We note that in many real life applications, we use models to compute conditional probabilities. In such cases we compute the probability given a partial skeleton that determines some of the references and attributes in the domain and queries the conditional probability over the remaining aspects of the instance. In such a situation, fixing the number of objects might not seem artificial.

Somewhat surprisingly, our definitions are such that the semantics of the model does not change. More precisely, by defining the existence events to be attributes, and incorporating them appropriately into the probabilistic model, we have set things up so that the semantics of Eq. (1) applies unchanged.

Of course, we must place some restrictions on our model to ensure that our definitions lead to a coherent probability model. First, a problem can arise if the range type of a slot of an undetermined class refers to itself, i.e., $\mathrm{Range}[X.\rho] = X$. In this case, the set $\mathcal{O}^{\mathcal{I}}(X)$ is defined circularly, in terms of itself. To ensure semantic coherence, we impose the following restrictions on our models: Let X be an undetermined class. An attribute $X.A$ cannot be an ancestor of $X.E$. In addition, an object can only exist if all the objects it refers to exist, i.e., for every slot $\rho \in \mathcal{R}(X)$, $P(x.E = \mathit{false} \mid x.\rho.E = \mathit{false}) = 1$. We also require that dependencies can only "pass through" objects that exist. For any slot $Y.\rho$ of range-type X, we define the *usable slot* $\overline{\rho}$ as follows: for any $y \in \mathcal{O}^{\mathcal{I}}(Y)$, we define $y.\overline{\rho} = \{x \in y.\rho \ : \ x.E = \mathit{true}\}$. We allow only $\overline{\rho}$ to be used in defining parent slot chains in the dependency model $\mathcal{S}$.

It is easy to capture these requirements in our class dependency graph. For every slot $\rho \in \mathcal{R}(X)$ whose range type is Y, we have an edge from $Y.E$ to $X.E$. For every attribute $X.A$, every $X.\overline{\rho_1}.\ldots.\overline{\rho_k}.B \in \mathrm{Pa}(X.A)$, and every $i = 1, \ldots, k$, we have an edge from $\mathrm{Range}[\rho_i].E$ to $X.A$. As before, we require that the attribute dependency graph is stratified.

It turns out that our requirements are sufficient to guarantee that the entity set of every undetermined entity type is well-defined, and allow our extended language to be viewed as a standard PRM. Hence, it follows easily that our extended PRM defines a coherent probability distribution:

Theorem 5: *Let Π be a PRM with undetermined classes and a stratified class dependency graph. Let σ_e be an entity skeleton. Then the PRM and σ_e uniquely define a relational skeleton σ_r over all classes, and a probability distribution over instantiations $\mathcal{I}$ that extends σ_e via Eq. (1).*

Note that a full instantiation $\mathcal{I}$ also determines the existence attributes for undetermined classes. Hence, the probability distribution induced by the PRM also specifies the probability that a certain entity will exist in the model.

We note that real-world databases do not specify the descriptive attributes of entities that do not exist. Thus, these attributes are unseen variables in the probability model. Since we only allow dependencies on objects that exist (for which $x.E = \mathit{true}$), then nonexistent objects are *leaves* in the model. Hence, they can be ignored in the computation of $P(\mathcal{I} \mid \sigma_e, \Pi)$. The only contribution of a nonexistent entity x to the probability of an instantiation is the probability that $x.E = \mathit{false}$.

Learning PRMs

In the previous sections we discussed three variants of PRM models that differ in their expressive power. Our aim is to *learn* such models from data. Thus, the task is as follows: given an instance and a schema, construct a PRM that describes the dependencies between objects in the schema. We stress that, when learning, all three variants we described use the same form of training data: a complete instantiation that describes a set of objects, their attribute values and their reference slots. However, in each variant, we attempt to learn somewhat different structure from this data. In the basic PRM learning of (Friedman *et al.* 1999), we learn the probability of attributes given other attributes; in learning PRMs with reference uncertainty, we also attempt to learn the rules that govern the choice of slot references; and in learning PRMs with existence uncertainty, we attempt to learn the probability of existence of relationship objects.

We start by reviewing the approach of (Friedman *et al.* 1999) for learning PRMs with attribute uncertainty and then describe new algorithms for learning PRMs with structural uncertainty. Conceptually, we can separate the learning problem into two basic questions: how to evaluate the "goodness" of a candidate structure, and how to search the space of legal candidate structures. We consider each question separately.

Model Scoring

For scoring candidate structures, we adapt Bayesian *model selection* (Heckerman 1998). We 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 σ, and the probability of the instantiation assuming the structure is σ. By making fairly reasonable assumptions about the prior probability of structures and parameters, this term can be *decomposed* into a product of terms. Each term in the decomposed form measures how well we predict the values of $X.A$ given the values of its parents. Moreover, the term for $P(X.A \mid \mathbf{u})$ depends only on the *sufficient statistics* $C_{X.A}[v, \mathbf{u}]$, that count the number of entities with $x.A = v$ and $\mathrm{Pa}(x.A) = \mathbf{u}$. These sufficient statistics can be computed using standard relational database queries.

The extension of the Bayesian score to PRMs with existence uncertainty is straightforward. The only new issue is how to compute sufficient statistics that include existence attributes $x.E$ without explicitly adding all nonexistent entity into our database. This, however, can be done in a straightforward manner. Let $\mathbf{u}$ be a particular instantiation of $\mathrm{Pa}(X.E)$. To compute $C_{X.E}[\mathit{true}, \mathbf{u}]$, we can use standard database query to compute how many objects $x \in \mathcal{O}^{\sigma}(X)$ have $\mathrm{Pa}(x.E) = \mathbf{u}$. To compute $C_{X.E}[\mathit{false}, \mathbf{u}]$, we need to compute the number of *potential* entities. We can do this without explicitly considering each $(x_1, \ldots, x_k) \in \mathcal{O}^{\mathcal{I}}(Y_1) \times \cdots \mathcal{O}^{\mathcal{I}}(Y_k)$ by decomposing the computation as follows: Let ρ be a reference slot of X with $\mathrm{Range}[\rho] = Y$. Let $\mathrm{Pa}_{\rho}(X.E)$ be the subset of parents of $X.E$ along slot ρ and let $\mathbf{u}_{\rho}$ be the corresponding instantiation. We count the number of y consistent with $\mathbf{u}_{\rho}$. If $\mathrm{Pa}_{\rho}(X.E)$ is empty, this count is simply the $|\mathcal{O}^{\mathcal{I}}(Y)|$. The product of these counts is the number of potential entities. To compute $C_{X.E}[\mathit{false}, \mathbf{u}]$, we simply subtract $C_{X.E}[\mathit{true}, \mathbf{u}]$ from this number.

The extension required to deal with reference uncertainty is also not a difficult one. Once we fix the set partition attributes $\Psi[\rho]$, we can treat the variable S_ρ as any other attribute in the PRM. Thus, scoring the success in predicting the value of this attribute given the value of its parents is done using the standard Bayesian methods we use of attribute uncertainty (e.g., using a standard conjugate Dirichlet prior).

Model Search

To find a high-scoring structure, we use a heuristic search procedure that iteratively broadens the search space to consider longer slot chains in directions with promising dependencies. At each iteration, we use a search procedure that considers operators ω such as adding, deleting, or reversing edges in the dependency model $\mathcal{S}$. The search procedure performs greedy hill-climbing search in this space, using the Bayesian score to evaluate models.

No extensions to the search algorithm are required to handle existence uncertainty. We simply introduce the new attributes $X.E$, and integrate them into the search space, as usual. The only difference is that we must enforce the constraints on the model, by properly maintaining the class dependency graph described earlier.

The extension for incorporating reference uncertainty is more subtle. Initially, the partition of the range class for a slot $X.\rho$ is not given in the model. Therefore, we must also search for the appropriate set of attributes $\Psi[\rho]$. We introduce two new operators **refine** and **abstract**, which modify the partition by adding and deleting attributes from $\Psi[\rho]$. Initially, $\Psi[\rho]$ is empty for each ρ. The **refine** operator adds an attribute into $\Psi[\rho]$; the **abstract** operator deletes one.

These newly introduced operators are treated as any other operator in our greedy hill-climbing search algorithm. They are considered by the search algorithm at the same time as the standard operators that manipulate edges in the dependency model $\mathcal{S}$; the change in the score is evaluated for each possible operator; and the algorithm selects the best one to execute.

We note that, as usual, the decomposition of the score can be exploited to substantially speed up the search. In general, the score change resulting from an operator ω is reevaluated only after applying an operator ω' that modifies the parent set of an attribute that ω modifies. This is true also when we consider operators that modify the parent of selector attributes and existence attributes.

Results

We have tested our algorithms on a number of domains. In many cases, the two methods we have introduced for modeling structural uncertainty are not comparable, as they are applicable under different circumstances. However, here we present results for a domain in which both models are applicable, so that we have some basis for comparison. We present results for a dataset that we have constructed from information about movies and actors from the Internet Movie Database[4] and information about people's ratings of movies from the Each Movie dataset[5]. We extended the demographic information we had for the people by including census information available for a person's zip-code. From these, we constructed five classes (each of the size shown): Movie (1628), Actor (37,284); Role (52,563), Person (27,550), and Votes (320,751). Figure 2 shows the classes and their attributes.

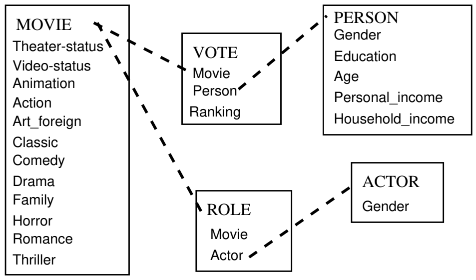


Figure 2: The schema for our movie domain

We learned a PRM for this domain using our two different methods for modeling structural uncertainty. For reference uncertainty, the model we learn (Π_{RU}) allows uncertainty over the *Movie* and *Actor* reference slots of Role, and the *Movie* and *Person* reference slots of Votes. For existence uncertainty, the model we learn (Π_{EU}) allows uncertainty over the existence of Role, and the existence of Votes.

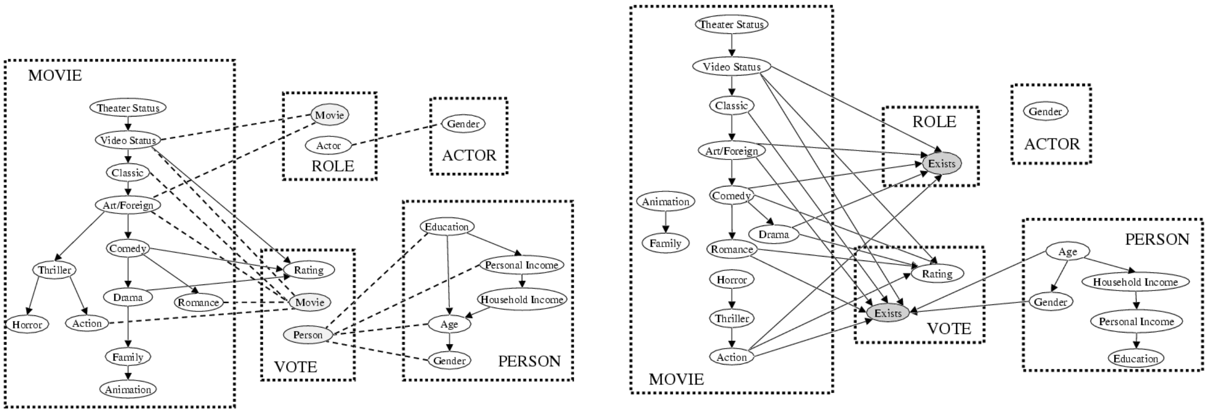
Figure 3(a) shows Π_{RU}. The edges indicate the parents of attributes in the dependency model. The dashed lines between a reference slot ρ and a descriptive attribute of the range class indicate attributes used in the partitioning $\Psi[\rho]$. Figure 3(b) shows Π_{EU}.

To evaluate the predictive performance of the two models (Π_{EU}, Π_{RU}), we learned PRMs using a training portion of the data and computed the log likelihood score of the held-out test subset of the data. The training set contained tables of the following sizes: Movie(1467), Vote(243333), Person(5210), Actor(34054), Role(46794). The test set contained the remaining data: Movie(161), Vote(5052), Person(585), Actor(5415), Role(5769). Note that the scores on the test set according to the two models are not directly comparable, since the space of instantiations they permit is different. Hence we compare the score of each model to the score of the model with uniform distribution over the existence of objects and values of reference slots (Π_{EU_U}, Π_{RU_U}). We also separately evaluated log-likelihood of the Person-Vote-Movie and Actor-Role-Movie portions of the test dataset.

Table 1 shows the results. Both models perform significantly better than their uniform counter parts. In particular, both models achieved better score gain on the Vote part of the data, but reference uncertainty model performed only

[4] ©1990-2000 Internet Movie Database Limited

[5] http://www.research.digital.com/SRC/EachMovie



(a) (b)

Figure 3: (a) Π_{RU}, the PRM learned using reference uncertainty. Shaded nodes indicate reference slots. Dashed lines indicate attributes used in defining the partition. (b) Π_{EU}, the PRM learned using existence uncertainty. Shaded nodes indicate exist attributes.

	Π_{EU}	Π_{EU_U}	Δ	Π_{RU}	Π_{RU_U}	Δ
All	-210044	-213798	3754	-149705	-152280	2575
Vote	-87965	-89950	1985	-67641	-70079	2438
Role	-122836	-123999	1163	-82827	-82963	136

Table 1: Evaluation of Π_{EU} and Π_{RU} on test set.

marginally better on the Role portion of the data. This may be due to the relative sparseness of the Role table, as few actors have more than one role.

Conclusion

In this paper, we present two representations for structural uncertainty: reference uncertainty and existence uncertainty. Reference uncertainty is relevant when we have cardinality information for our uncertain relations. Existence uncertainty is applicable for modeling uncertainty over many-many relations. We have shown how to integrate them with our learning framework, and presented results showing that they allow interesting patterns to be learned.

The ability to represent uncertainty over relational structure is a unique feature of our more expressive language. Our treatment here only scratches the surface of this capability. In particular, we note that neither of these representations for structural uncertainty is particularly satisfying when viewed as a generative model, although we have found each of them to be useful in a variety of domains. It is clear that there are many other possible ways to represent uncertainty over relational structure. We hope to provide a unified framework for this type of uncertainty in future work.

References

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

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

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

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

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.