5 Probabilistic Relational Models

Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer and Ben Taskar

Probabilistic relational models (PRMs) are a rich representation language for structured statistical models. They combine a frame-based logical representation with probabilistic semantics based on directed graphical models (Bayesian networks). This chapter gives an introduction to probabilistic relational models, describing semantics for attribute uncertainty, structural uncertainty, and class uncertainty. For each case, learning algorithms and some sample results are presented.

5.1 Introduction

Over the last decade, Bayesian networks have been used with great success in a wide variety of real-world and research applications. However, despite their success, Bayesian networks are often inadequate for representing large and complex domains. A Bayesian network for a given domain involves a prespecified set of random variables, whose relationship to each other is fixed in advance. Hence, a Bayesian network cannot be used to deal with domains where we might encounter a varying number of entities in a variety of configurations. This limitation of Bayesian networks is a direct consequence of the fact that they lack the concept of an "object" (or domain entity). Hence, they cannot represent general principles about multiple similar objects which can then be applied in multiple contexts.

Probabilistic relational models (PRMs) [13, 18] extend Bayesian networks with the concepts of objects, their properties, and relations between them. In a way, they are to Bayesian networks as relational logic is to propositional logic. A PRM specifies a template for a probability distribution over a database. The template includes 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 has a coherent formal semantics in terms of probability distributions over sets of relational logic interpretations. Given a set of ground objects, a PRM specifies a probability distribution over a set of interpretations involving these objects (and perhaps other objects as well). A PRM, together with a particular database of objects and relations, defines a probability distribution over the attributes of the objects.

In this chapter, we describe the semantics for PRMs with different types of uncertainty, and at the same time we describe the basic learning algorithms for PRMs. We propose an algorithm for automatically constructing or learning a PRM from an existing database. The learned PRM describes the patterns of interactions between attributes. In the learning problem, our input contains a relational schema that specifies the basic vocabulary in the domain — the set of classes, the attributes associated with the different classes, and the possible types of relations between objects in the different classes. The training data consists of a fully specified instance of that schema in the form of a relational database. Once we have learned a PRM, it serves as a tool for exploratory data analysis and can be used to make predictions and complex inferences in new situations. For additional details, including proofs of all of the theorems, see [9].

5.2 PRM Representation

The two components of PRM syntax are a logical description of the domain of discourse and a probabilistic graphical model template which describes the probabilistic dependencies in the domain. Here we describe the logical description of the domain as a relational schema, although it can be transformed into either a frame-based representation or a logic-based syntax is a relatively straightforward manner. Our probabilistic graphical component is depicted pictorially, although it can also be represented in a logical formalism; for example in the probabilistic relational language of [10]. We begin by describing the syntax and semantics for PRMs which have the simplest form of uncertainty, *attribute uncertainty*, and then move on to describing various forms of *structural uncertainty*.

5.2.1 Relational Language

The relational language allows us to describe the kinds of objects in our domain. For example, figure 5.1(a) shows the schema for a simple domain that we will be using as our running example. The domain is that of a university, and contains professors, students, courses, and course registrations. The classes in the schema are Professor, Student, Course, and Registration.

More formally, 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*. For example, professors may have descriptive attributes such as popularity and teaching ability; courses may have descriptive attributes such as rating and difficulty.

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 space of values is denoted $\mathcal{V}(X.A)$. We assume here that value spaces are finite. For example, the Student class has the descriptive

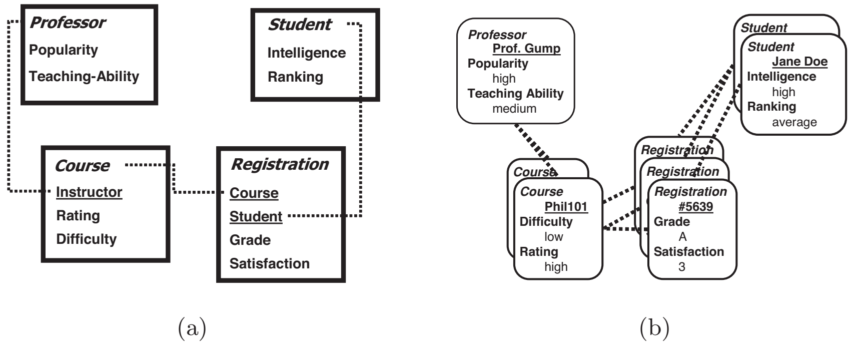


Figure 5.1 (a) A relational schema for a simple university domain. The underlined attributes are reference slots of the class and the dashed lines indicate the types of objects referenced. (b) An example instance of this schema. Here we do not show the values of the reference slots; we simply use dashed lines to indicate the relationships that hold between objects.

attributes *Intelligence* and *Ranking*. The value space for Student.*Intelligence* in this example is {*high, low*}.

In addition, we need a method for allowing an object to refer to another object. For example we may want a course to have a reference to the instructor of the course. And a registration record should refer both to the associated course and to the student taking the course.

The simplest way of achieving this effect is using *reference slots*. Specifically, each class is associated with a set of reference slots. The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use $X.\rho$ 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* $\mathrm{Dom}[\rho]$ is X and the *range type* $\mathrm{Range}[\rho]$ is Y for some class Y in $\mathcal{X}$. For example, the class Course has reference slot *Instructor* with range type Professor, and class Registration has reference slots *Course* and *Student*. In figure 5.1(a) the reference slots are underlined.

There is a direct mapping between our representation and that of relational databases. Each class corresponds to a single table and each attribute corresponds to a column. 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).

For each reference slot ρ, we can define an *inverse slot* ρ^{-1}, which is interpreted as the inverse function of ρ. For example, we can define an inverse slot for the *Student* slot of Registration and call it *Registered-In*. Note that this is not a one-to-one relation, but returns a *set* of Registration objects. More formally, if $\mathrm{Dom}[\rho]$ is X and $\mathrm{Range}[\rho]$ is Y, then $\mathrm{Dom}[\rho^{-1}]$ is Y and $\mathrm{Range}[\rho^{-1}]$ is X.

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 indirectly related. More precisely, we define a *slot chain* $\rho_1, \ldots, \rho_k$ to be a sequence of slots (inverse or otherwise) such that for all i, Range$[\rho_i]$ = Dom$[\rho_{i+1}]$. For example, Student.*Registered-In.Course.Instructor* can be used to denote a student's instructors. Note that a slot chain describes a *set* of objects from a class.[1]

The relational framework we have just described is motivated primarily by the concepts of relational databases, although some of the notation is derived from frame-based and object-oriented systems. However, the framework is a fully general one, and is equivalent to the standard vocabulary and semantics of relational logic.

5.2.2 Schema Instantiation

An *instance* $\mathcal{I}$ of a schema is simply a standard relational logic interpretation of this vocabulary. It specifies: for each class X, the set of objects in the class, $\mathcal{I}(X)$; a value for each attribute $x.A$ (in the appropriate domain) for each object x; and a value y for each reference slot $x.\rho$, which is an object in the appropriate range type, i.e., $y \in$ Range$[\rho]$. Conversely, $y.\rho^{-1} = \{x \mid x.\rho = y\}$. We use $\mathcal{A}(x)$ as a shorthand for $\mathcal{A}(X)$, where x is of class X. For each object x in the instance and each of its attributes A, we use $\mathcal{I}_{x.A}$ to denote the value of $x.A$ in $\mathcal{I}$. For example, figure 5.1(b) shows an instance of the schema from our running example. In this (simple) instance there is one Professor, two Classes, three Registrations, and two Students. The relations between them show that the professor is the instructor in both classes, and that one student ("Jane Doe") is registered only for one class ("Phil101"), while the other student is registered for both classes.

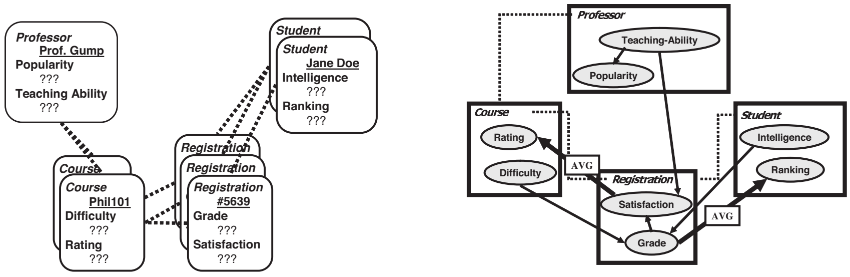
5.2.3 Probabilistic Model

A PRM defines a probability distribution over a set of instances of a schema. Most simply, we assume that the set of objects and the relations between them are fixed, i.e., external to the probabilistic model. Then, the PRM defines only a probability distribution over the attributes of the objects in the model. The *relational skeleton* defines the possible instantiations that we consider; the PRM defines a distribution over the possible worlds consistent with the relational skeleton.

Definition 5.1
A *relational skeleton* σ_r of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects $\sigma_r(X_i)$ for each class and the relations that hold between the objects. However, it leaves the values of the attributes unspecified. ∎

Figure 5.2(a) shows a relational skeleton for our running example. The relational skeleton defines the random variables in our domain; we have a random variable for

[1] It is also possible to define slot chains as *multi-sets* of objects; here we have found it sufficient to make them sets of objects, but there may be domains where multi-sets are desirable.



(a) (b)

Figure 5.2 (a) The relational skeleton for the university domain. (b) The PRM dependency structure for our university example.

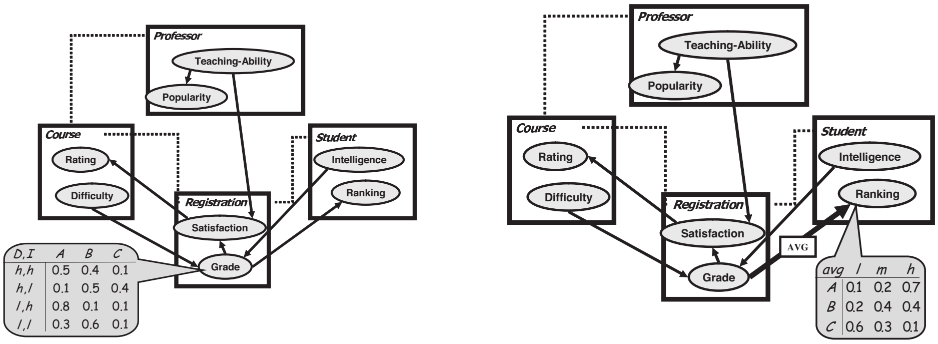
each attribute of each object in the skeleton. A PRM then specifies a probability distribution over *completions* $\mathcal{I}$ of the skeleton.

A PRM 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 $X.A$. In figure 5.2(b), the arrows define the dependency structure.

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 $\sigma_r(X)$, $x.A$ will depend probabilistically on $x.B$. For example, in figure 5.2(b), a professor's *Popularity* depends on her *Teaching-Ability*. The attribute $X.A$ can also depend on attributes of related objects $X.\boldsymbol{K}.B$, where $\boldsymbol{K}$ is a slot chain. In figure 5.2(b), the grade of a student depends on Registration.*Student*.*Intelligence* and Registration.*Course*.*Difficulty*. Or we can have a longer slot chain, for example, the dependence of student satisfaction on Registration.*Course*.*Instructor*.*Teaching-Ability*.

In addition, we can have a dependence of student ranking on Student.*Registered-In*.*Grade*. To understand the semantics of this formal dependence for an individual object x, recall that $x.\boldsymbol{K}$ represents the *set* of objects that are $\boldsymbol{K}$-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 $\{y.B \ : \ y \in x.\boldsymbol{K}\}$. For example, a student's rank depends on the grades in the courses in which he or she are registered. However each student may be enrolled in a different number of courses, and we will need a method of compactly representing these complex dependencies.

The notion of *aggregation* from database theory gives us an appropriate tool to address this issue: $x.A$ will depend probabilistically on some aggregate property of this multiset. There are many natural and useful notions of aggregation of a set: its mode (most frequently occurring value); its mean value (if values are numerical);



D,I	A	B	C
h,h	0.5	0.4	0.1
h,l	0.1	0.5	0.4
l,h	0.8	0.1	0.1
l,l	0.3	0.6	0.1

avg	l	m	h
A	0.1	0.2	0.7
B	0.2	0.4	0.4
C	0.6	0.3	0.1

(a)

(b)

Figure 5.3 (a) The CPD for Registration.*Grade* (b) The CPD for an aggregate dependency of Student.*Ranking* on Student.*Registered-In*.*Grade*.

its median, maximum, or minimum (if values are ordered); its cardinality; etc. In the preceding example, we can have a student's ranking depend on her grade point average (GPA), or the average grade in her courses (or in the case where the grades are represented as letters, we may use median; in our example we blur the distinction and assume that average is defined appropriately).

More formally, our language allows a notion of an aggregate γ; γ 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 set. We allow $X.A$ to have as a parent $\gamma(X.\boldsymbol{K}.B)$; the semantics is that for any $x \in X$, $x.A$ will depend on the value of $\gamma(x.\boldsymbol{K}.B)$. In our example PRM, there are two aggregate dependencies defined, one that specifies that the ranking of a student depends on the average of her grades and one that specifies that the rating of a course depends on the average satisfaction of students in the course.

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 $P(X.A \mid \text{Pa}(X.A))$. We require that the CPDs are legal. Figure 5.3 shows two CPDs. Let $\mathbf{U}$ be the set of parents of $X.A$, $\mathbf{U} = \text{Pa}(X.A)$. Each of these parents U_i — whether a simple attribute in the same relation or an aggregate of a set of $\boldsymbol{K}$ relatives — has a set of values $\mathcal{V}(U_i)$ in some ground type. For each tuple of values $\mathbf{u} \in \mathcal{V}(\mathbf{U})$, we specify a distribution $P(X.A \mid \mathbf{u})$ over $\mathcal{V}(X.A)$. This entire set of parameters comprises $\theta_{\mathcal{S}}$.

Definition 5.2

A *probabilistic relational model (PRM)* Π for a relational schema $\mathcal{R}$ 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 $\gamma(X.\boldsymbol{K}.B)$, where $\boldsymbol{K}$ is a slot chain and γ is an aggregate of $X.\boldsymbol{K}.B$;

- a legal *conditional probability distribution (CPD)*, $P(X.A \mid \mathrm{Pa}(X.A))$. ▮

5.2.4 PRM Semantics

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

For a given skeleton σ_r, the PRM structure induces a *ground* Bayesian network over the random variables $x.A$.

Definition 5.3
A PRM Π together with a skeleton σ_r defines the following ground Bayesian network:

- There is a node for every attribute of every object $x \in \sigma_r(X)$, $x.A$.
- Each $x.A$ depends probabilistically on parents of the form $x.B$ or $x.\boldsymbol{K}.B$. If $\boldsymbol{K}$ is not single-valued, then the parent is the aggregate computed from the set of random variables $\{y \mid y \in x.\boldsymbol{K}\}$, $\gamma(x.\boldsymbol{K}.B)$.
- The CPD for $x.A$ is $P(X.A \mid \mathrm{Pa}(X.A))$. ▮

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

$$\begin{aligned} P(\mathcal{I} \mid \sigma_r, \mathcal{S}, \theta_{\mathcal{S}}) &= \prod_{x \in \sigma_r} \prod_{A \in \mathcal{A}(x)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \\ &= \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \sigma_r(X_i)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}). \end{aligned} \tag{5.1}$$

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

5.2.5 Coherence of Probabilistic Model

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

5.2.5.1 Instance Dependency Graph

We want to ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. To do so, we can consider the graph of dependencies among attributes of objects in the skeleton, which we will call the *instance dependency graph*, G_{σ_r}.

Definition 5.4
The *instance dependency graph* G_{σ_r} for a PRM Π and a relational skeleton σ_r has a node for each descriptive attribute of each object $x \in \sigma_r(X)$ in each class $X \in \mathcal{X}$. Each $x.A$ has the following edges:

1. Type I edges: For each formal parent of $x.A$, $X.B$, we introduce an edge from $x.B$ to $x.A$.
2. Type II edges: For each formal parent $X.\boldsymbol{K}.B$, and for each $y \in x.\boldsymbol{K}$, we define an edge from $y.B$ to $x.A$. ▮

Type I edges correspond to intra-object dependencies and type II edges correspond to inter-object dependencies. We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to a relational skeleton σ_r if the instance dependency graph G_{σ_r} over the variables $x.A$ is acyclic. In this case, we are guaranteed that the PRM defines a coherent probabilistic model over complete instantiations $\mathcal{I}$ consistent with σ_r:

Theorem 5.5
Let Π be a PRM whose dependency structure $\mathcal{S}$ is acyclic relative to a relational skeleton σ_r. Then Π and σ_r define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_r via (5.1).

5.2.5.2 Class Dependency Graph

The instance dependency graph we just described allows us to check whether a dependency structure $\mathcal{S}$ is acyclic relative to a fixed skeleton σ_r. However, we often want stronger guarantees: we want to ensure that our dependency structure is acyclic for any skeleton that we are likely to encounter. How do we guarantee this property based only on the class-level PRM? To do so, we consider potential dependencies at the class level. More precisely, we define a *class dependency graph*, which reflects these dependencies.

Definition 5.6
The *class dependency graph* G_Π for a PRM Π has a node for each descriptive attribute $X.A$, and the following edges:

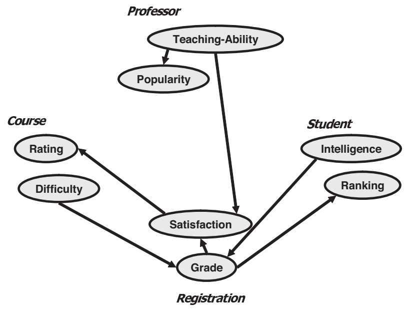


Figure 5.4 The class dependency graph for the school PRM.

1. Type I edges: For any attribute $X.A$ and any of its parents $X.B$, we introduce an edge from $X.B$ to $X.A$.
2. Type II edges: For any attribute $X.A$ and any of its parents $X.\boldsymbol{K}.B$ we introduce an edge from $Y.B$ to $X.A$, where $Y = \text{Range}[X.\boldsymbol{K}]$. ∎

Figure 5.4 shows the dependency graph for our school domain.

The most obvious approach for using the class dependency graph is simply to require that it be acyclic. This requirement is equivalent to assuming a stratification among the attributes of the different classes, and requiring that the parents of an attribute precede it in the stratification ordering. As theorem 5.7 shows, if the class dependency graph is acyclic, we can never have that $x.A$ depends (directly or indirectly) on itself.

Theorem 5.7

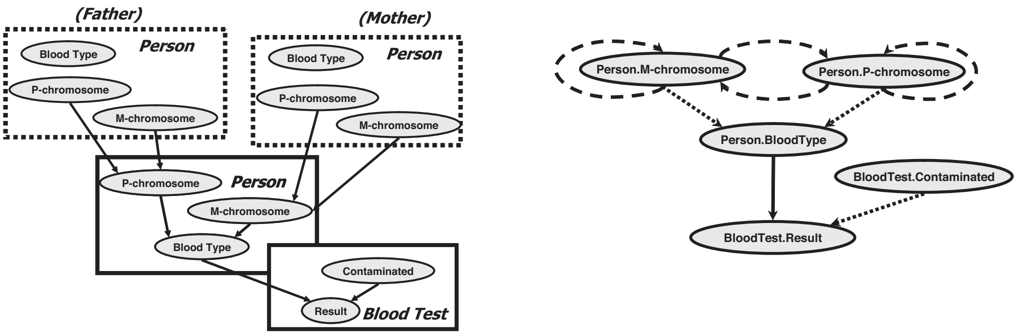
If the class dependency graph G_Π is acyclic for a PRM Π, then for *any* skeleton σ_r, the instance dependency graph is acyclic.

The following corollary follows immediately:

Corollary 5.8

Let Π be a PRM whose class dependency structure $\mathcal{S}$ is acyclic. For any relational skeleton σ_r, Π, and σ_r define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_r via (5.1).

For example, if we examine the PRM of figure 5.2(b), we can easily convince ourselves that we cannot create a cycle in any instance. Indeed, as we saw in figure 5.4, the class dependency graph is acyclic. Note, however, that if we introduce additional dependencies we can create cycles. For example, if we make Professor.*Teaching-Ability* depend on the rating of courses she teaches (e.g., if high teaching ratings increase her motivation), then the resulting class dependency graph is cyclic, and there is no stratification order that is consistent with the PRM structure. An inability to stratify the class dependency graph implies that there are skeletons for which the PRM will induce a distribution with cyclic dependencies.



(a) (b)

Figure 5.5 (a) A simple PRM for the genetics domain. (b) The corresponding dependency graph. Dashed edges correspond to "green" dependencies, dotted edges correspond to "yellow" dependencies, and solid edges correspond to "red" dependencies.

5.2.5.3 Guaranteed Acyclic Relationships

In some important cases, a cycle in the class dependency graph is not problematic, it will not result in a cyclic instance dependency graph. This can be the case when we have additional domain constraints on the form of skeletons we may encounter. Consider, for example, a simple genetic model of the inheritance of a single gene that determines a person's blood type, shown in figure 5.5(a). 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 BloodTest. Class Person has reference slots *Mother* and *Father* and descriptive attributes *Gender*, *P-Chromosome* (the chromosome inherited from the father), and *M-Chromosome* (inherited from the mother). *BloodTest* has a reference slot *Test-Of* (not shown explicitly in the figure) that points to the owner of the test, and descriptive attributes *Contaminated* and *Result*.

In our genetic model, the genotype of a person depends on the genotype of her parents; thus, at the class level, we have *Person.P-Chromosome* depending directly on *Person.P-Chromosome*. As we can see in figure 5.5(b), this dependency results in a cycle that clearly violates the acyclicity requirements of our simple class dependency graph. However, it is clear to us that the dependencies in this model are not actually cyclic for any skeleton that we will actually encounter in this domain. The reason is that, in "legitimate" skeletons for this schema, a person cannot be his own ancestor, which disallows the situation of the person's genotype depending (directly or indirectly) on itself. In other words, although the model appears to be cyclic at the class level, we know that this cyclicity is always resolved at the level of individual objects.

Our ability to guarantee that the cyclicity is resolved relies on some prior knowledge that we have about the domain. We want to allow the user to give us information such as this, so that we can make stronger guarantees about acyclicity and allow richer dependency structures in the PRM. In particular, the user can specify that certain reference slots are *guaranteed acyclic*. In our genetics example, *Father* and *Mother* are guaranteed acyclic; cycles involving these attributes may in fact be legal. Moreover, they are mutually guaranteed acyclic, so that compositions of the slots are also guaranteed acyclic. Figure 5.5(b) shows the class dependency graph for the genetics domain, with guaranteed acyclic edges shown as dashed edges.

We allow the user to assert that certain reference 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 ρ-relative for some $\rho \in \mathcal{R}_{ga}$ of x, then $y \prec_{ga} x$. We say that a slot chain $\boldsymbol{K}$ is guaranteed acyclic if each of its component ρ's is guaranteed acyclic.

This prior knowledge allows us to guarantee the legality of certain dependency models. We start by building a *colored class dependency graph* that describes the direct dependencies between the attributes.

Definition 5.9
The *colored class dependency graph* G_Π for a PRM Π has the following edges:

1. **Yellow edges:** If $X.B$ is a parent of $X.A$, we have a *yellow* edge $X.B \rightarrow X.A$.
2. **Green edges:** If $\gamma(X.\boldsymbol{K}.B)$ is a parent of $X.A$, $Y = \text{Range}[X.\boldsymbol{K}]$, and $\boldsymbol{K}$ is guaranteed acyclic, we have a green edge $Y.B \rightarrow X.A$.
3. **Red edges:** If $\gamma(X.\boldsymbol{K}.B)$ is a parent of $X.A$, $Y = \text{Range}[X.\boldsymbol{K}]$, and $\boldsymbol{K}$ is not guaranteed acyclic, we have a red edge $Y.B \rightarrow X.A$. ∎

Note that there might be several edges, perhaps 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, however, 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:

Definition 5.10
A (colored) dependency graph is *stratified* if every cycle in the graph contains at least one green edge and no red edges. ∎

Theorem 5.11
If the colored class dependency graph is stratified for a PRM Π, then for *any* skeleton σ_r, the instance dependency graph is acyclic.

In other words, if the colored dependency graph of $\mathcal{S}$ and $\mathcal{R}_{ga}$ is stratified, then for any skeleton σ_r for which the slots in $\mathcal{R}_{ga}$ are jointly acyclic, $\mathcal{S}$ defines a coherent probability distribution over assignments to σ_r.

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. Then the graph is stratified if and only if it is acyclic. In the genetics example, all the parent relations would be in $\mathcal{R}_{ga}$. The only edges involved in cycles are green edges.

We can also support multiple guaranteed acyclic relations by using different shades of green for each set of guaranteed acyclic relations. Then a cycle is safe as long as it contains at most one shade of green edge.

5.3 The Difference between PRMs and Bayesian Networkss

The PRM specifies a probability distribution using the same underlying principles used in specifying Bayesian networks. The assumption is that each of the random variables in the PRM — in this case the attributes $x.A$ of the individual objects x — is directly influenced by only a few others. The PRM therefore defines for each $x.A$ a set of parents, which are the direct influences on it, and a local probabilistic model that specifies the dependence on these parents. In this way, the PRM is like a Bayesian Network.

However, there are two primary differences between PRMs and Bayesian networks. First, a PRM defines the dependency model at the class level, allowing it to be used for any object in the class. In some sense, it is analogous to a universally quantified statement. Second, the PRM explicitly uses the relational structure of the skeleton, in that it allows the probabilistic model of an attribute of an object to depend also on attributes of related objects. The specific set of related objects can vary with the skeleton σ_r; the PRM specifies the dependency in a generic enough way that it can apply to an arbitrary relational structure.

One can understand the semantics of a PRM together with a particular relational skeleton σ_r by examining the ground Bayesian network defined earlier. The network has a node for each attribute of the objects in the skeleton. The local probability models for attributes of objects in the same class are identical (we can view the parameters as being shared); however, the distribution for a node will depend on the values of its parents, and the parents of each node are determined by the skeleton.

It is important to note the construction of the ground Bayesian Network is just a thought experiment; in many cases there is no need to actually construct this large underlying Bayesian network.

5.4 PRMs with Structural Uncertainty

The previous section gave the syntax and semantics for the most basic type of PRM, a PRM in which there is uncertainty over the the attributes of the objects in the relational skeleton. As discussed in the last section, this is already a significant generalization beyond propositional Bayesian networks. In this section, we propose probabilistic models for the attributes of the objects in a relational model and also for the relational or link structure *itself*. In other words, we model the probability that certain relationships hold between objects. We propose two mechanisms for modeling link uncertainty: *reference uncertainty* and *existence uncertainty*.

The PRM framework presented so far focuses on modeling the distribution over the attributes of the objects in the model. It takes the relational structure itself — the objects and the relational links between entities — to be background knowledge, determined outside the probabilistic model. This assumption implies that the model cannot be used to predict the relational structure itself. A more subtle yet very important point is that the relational structure is informative in and of itself. For example, the links from and to a webpage are very informative about the type of webpage [6], and the citation links between papers are very informative about the paper topics [5].

By making objects and links first-class citizens in the model, our language easily allows us to place a probabilistic model directly over them. In other words, we can extend our framework to define probability distributions over the presence of relational links between objects in our model. By introducing these aspects of the world into the model, and correlating them with other attributes, we can both predict the link structure and use the presence of links to reach conclusions about attribute values.

5.5 Probabilistic Model of Link Structure

In our discussion so far, all relations between attributes are determined by the relational skeleton σ_r; only the descriptive attributes are uncertain. The relational skeleton specifies the set of objects in all classes, as well as all the relationships that hold between them (in other words, it specifies the values for all of the reference slots). Consider the simple university domain of section 5.2 describing professors, courses, students, and registrations. The relational skeleton specifies the complete relational structure in the model: it specifies which professor teaches each course, and it specifies all of the registrations of students in courses. In our simple university example, the relational skeleton (shown in figure 5.2(a)) contains all of the information except for the values for the descriptive attributes.

There is one distinction we will add to our relational schema. It is 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. We

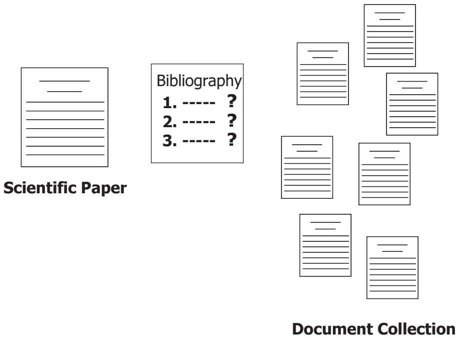


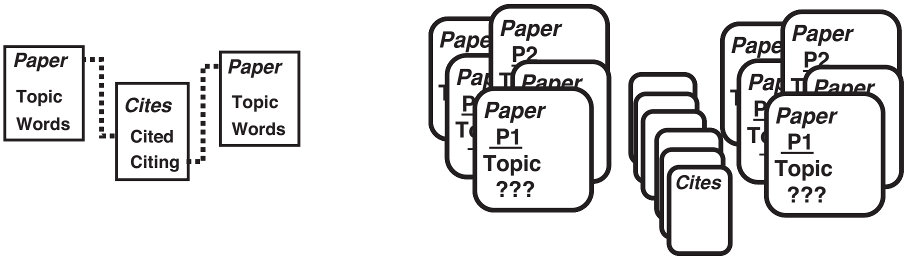
Figure 5.6 Reference uncertainty in a simple citation domain.

introduce $\mathcal{X}_{\mathcal{E}}$ 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.

5.5.1 Reference Uncertainty

Consider a simple citation domain illustrated in figure 5.6. Here we have a document collection. Each document has a bibliography that references some of the other documents in the collection. We may know the number of citations made by each document (i.e., it is outside the probabilistic model). By observing the citations that are made, we can use the links to reach conclusions about other attributes in the model. For example, by observing the number of citations to papers of various topics, we may be able to infer something about the topic of the citing paper.

figure 5.7(a) shows a simple schema for this domain. We have two classes, Paper and Cites. The Paper class has information about the topic of the paper and the words contained in the paper. For now, we simply have an attribute for each word that is *true* if the word occurs in the page and *false* otherwise. The Cites class represents the citation of one paper, the *Cited* paper, by another paper, the *Citing* paper. (In the figure, for readability, we show the Paper class twice.) In this model, we assume that the set of objects is prespecified, but relations among them, i.e., reference slots, are subject to probabilistic choices. Thus, rather than being given a full relational skeleton σ_r, we assume that we are given an *object skeleton* σ_o. The object skeleton specifies only the objects $\sigma_o(X)$ in each class $X \in \mathcal{X}$, but not the values of the reference slots. In our example, the object skeleton specifies the objects in class Paper and the objects in class Cites, but the reference slots of the Cites relation, Cites.*Cited* and Cites.*Citing* are unspecified. In other words, the probabilistic model does not provide a model of the total number of citation links, but only a distribution over their "endpoints." figure 5.7 shows an object skeleton for the citation domain.



(a) (b)

Figure 5.7 (a) A relational schema for the citation domain. (b) An object skeleton for the citation domain.

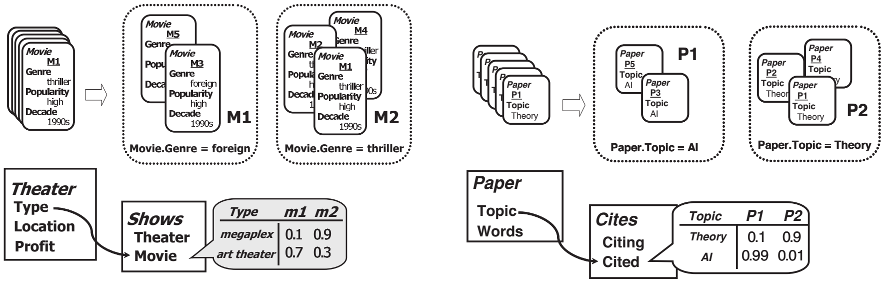
5.5.1.1 Probabilistic Model

In the case of reference uncertainty, we specify a probabilistic model for 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 the class Y to which $X.\rho$ refers. Thus, we need to specify a probability distribution over the set of all objects in Y. For example, for Cites.*Cited*, we must specify a distribution over the objects in class Paper.

A naive approach is to simply have the PRM specify a probability distribution directly over the objects $\sigma_o(Y)$ in Y. For example, for Cites.*Cited*, we would have to specify a distribution over the primary keys of Paper. This approach has two major flaws. Most obviously, this distribution would require a parameter for each object in Y, leading to a very large number of parameters. This is a problem both from a computational perspective — the model becomes very large — and from a statistical perspective — we often would not have enough data to make robust estimates for the parameters. 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.

In order to achieve a general and compact representation, we use the *attributes* of Y to define the probability distribution. In this model, we partition the class Y into subsets labeled $\psi_1, \ldots, \psi_m$ according to the values of some of its attributes, and specify a probability for choosing each partition, i.e., a distribution over the partitions. We then select an object within that partition uniformly.

For example, consider a description of movie theater showings as in figure 5.8(a). For the foreign key Shows.*Movie*, we can partition the class Movie by *Genre*, indicating that a movie theater first selects the genre of movie it wants to show, and then selects uniformly among the movies with the selected genre. For example, a movie theater may be much more likely to show a movie which is a thriller than a foreign movie. Having selected, for example, to show a thriller, the theater then selects the actual movie to show uniformly from within the set of thrillers. In addition, just as in the case of descriptive attributes, the partition choice can



(a)

(b)

Figure 5.8 (a) An example of reference uncertainty for a movie theater's showings. (b) A simple example of reference uncertainty in the citation domain

depend on other attributes in our model. Thus, the selector attribute can have parents. As illustrated in the figure, the choice of movie genre might depend on the type of theater. Consider another example in our citation domain. As shown in figure 5.8(b), we can partition the class Paper by *Topic*, indicating that the topic of a citing paper determines the topics of the papers it cites; and then the cited paper is chosen uniformly among the papers with the selected topic.

We make this intuition precise by defining, for each slot ρ, a *partition function* Ψ_ρ. We place several restrictions on the partition function which are captured in the following definition:

Definition 5.12
Let $X.\rho$ be a reference slot with domain Y. Let $\Psi_\rho : Y \to \mathrm{Dom}[\Psi_\rho]$ be a function where $\mathrm{Dom}[\Psi_\rho]$ is a finite set of labels. We say that Ψ_ρ is a *partition function* for ρ if there is a subset of the attributes of Y, $\mathcal{P}[\rho] \subseteq \mathcal{A}(Y)$, such that for any $y \in Y$ and any $y' \in Y$, if the values of the attributes $\mathcal{P}[\rho]$ of y and y' are the same, i.e., for each $A \in \mathcal{P}[\rho]$, $y.A = y'.A$, then $\Psi_\rho(y) = \Psi_\rho(y')$. We refer to $\mathcal{P}[\rho]$ as the *partition attributes* for ρ. ∎

Thus, the values of the partition attributes are all that is required to determine the partition to which an object belongs.

In our first example, $\Psi_{\mathsf{Shows}.\mathit{Movie}} : \mathsf{Movie} \to \{\mathit{foreign}, \mathit{thriller}\}$ and the partition attributes are $\mathcal{P}[\mathsf{Shows}.\mathit{Movie}] = \{\mathit{Genre}\}$. In the second example, $\Psi_{\mathsf{Cites}.\mathit{Cited}} : \mathsf{Paper} \to \{\mathit{AI}, \mathit{Theory}\}$ and the partition attributes are $\mathcal{P}[\mathsf{Cites}.\mathit{Cited}] = \{\mathit{Topic}\}$.

There are a number of natural methods for specifying the partition function. It can be defined simply by having one partition for each possible combination of values of the partition attributes, i.e., one partition for each value in the cross product of the partition attribute values. Our examples above take this approach. In both cases, there is only a single partition attribute, so specifying the partition function in this manner is not too unwieldy, but for larger collections of partition attributes or for partition attributes with large domains, this method for defining the partitioning function may be problematic. A more flexible and scalable approach is to define the partition function using a decision tree built over the partition attributes. In this case, there is one partition for each of the leaves in the decision tree.

Each possible value ψ determines a subset of Y from which the value of ρ (the referent) will be selected. For a particular instantiation $\mathcal{I}$ of the database, we use $\mathcal{I}(Y_\psi)$ to represent the set of objects in $\mathcal{I}(Y)$ that fall into the partition ψ.

We now represent a probabilistic model over the values of ρ by specifying a distribution over possible partitions, which encodes how likely the reference value of ρ is to fall into one partition versus another. We formalize our intuition above by introducing a *selector attribute* S_ρ, whose domain is $\mathrm{Dom}[\Psi_\rho]$. The specification of the probabilistic model for the selector attribute S_ρ is the same as that of any other attribute: it has a set of parents and a CPD. In our earlier example, the CPD of $\mathsf{Show}.S_{Movie}$ might have as a parent $\mathsf{Theater}.\mathit{Type}$. For each instantiation of the parents, we have a distribution over $\mathrm{Dom}[S_\rho]$. The choice of value for S_ρ determines the partition Y_ψ from which the reference value of ρ is chosen; the choice of reference value for ρ is uniformly distributed within this set.

Definition 5.13

A *probabilistic relational model* Π *with reference uncertainty* over a relational schema $\mathcal{R}$ has the same components as in definition 5.2. In addition, for each reference slot $\rho \in \mathcal{R}(X)$ with $\mathrm{Range}[\rho] = Y$, we have:

- a partition function Ψ_ρ with a set of partition attributes $\mathcal{P}[\rho] \subseteq \mathcal{A}(Y)$;
- a new selector attribute S_ρ within X which takes on values in the range of Ψ_ρ;
- a set of parents and a CPD for S_ρ. ▮

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

$$P(\mathcal{I} \mid \sigma_o, \Pi) = \prod_{X \in \mathcal{X}} \prod_{x \in \sigma_o(X)} \prod_{A \in \mathcal{A}(X)} P(x.A \mid \mathrm{Pa}(x.A)) \prod_{\rho \in \mathcal{R}(X), y = x.\rho} \frac{P(x.S_\rho = \psi[y] \mid \mathrm{Pa}(x.S_\rho))}{|\mathcal{I}(Y_{\psi[y]})|}, \tag{5.2}$$

where $\psi[y]$ refers to $\Psi_\rho(y)$ — the partition that the partition function assigns y. Note that the last term in (5.2) depends on $\mathcal{I}$ in three ways: the interpretation of $x.\rho = y$, the values of the attributes $\mathcal{P}[\rho]$ within the object y, and the size of $Y_{\psi[y]}$. The above probability is not well-defined if there are no objects in a partition, so in that case we define it to be zero.

5.5.2 Coherence of the Probabilistic Model

As in the case of PRMs with attribute uncertainty, we must be careful to guarantee that our probability distribution is in fact coherent. In this case, the object skeleton does not specify which objects are related to which, and therefore the mapping of formal to actual parents depends on probabilistic choices made in the model. The associated ground Bayesian network will therefore be cumbersome and not particularly intuitive. We define our coherence constraints using an instance dependency graph, relative to our PRM and object skeleton.

Definition 5.14

The *instance dependency graph* for a PRM Π and an object skeleton σ_o is a graph G_{σ_o} with the nodes and edges described below. For each class X and each $x \in \sigma_o(X)$, we have the following nodes:

- a node $x.A$ for every descriptive attribute $X.A$;
- a node $x.\rho$ and a node $x.S_\rho$, for every reference slot $X.\rho$.

The dependency graph contains five types of edges:

- **Type I edges:** Consider any attribute (descriptive or selector) $X.A$ and formal parent $X.B$. We define an edge $x.B \rightarrow x.A$, for every $x \in \sigma_o(X)$.
- **Type II edges:** Consider any attribute (descriptive or selector) $X.A$ and formal parent $X.\boldsymbol{K}.B$ where $\text{Dom}[X.\boldsymbol{K}] = Y$. We define an edge $y.B \rightarrow x.A$, for every $x \in \sigma_o(X)$ and $y \in \sigma_o(Y)$.
- **Type III edges:** Consider any attribute $X.A$ and formal parent $X.\boldsymbol{K}.B$, where $\boldsymbol{K} = \rho_1, \ldots, \rho_k$, and $\text{Dom}[\rho_i] = X_i$. We define an edge $x.\rho_1 \rightarrow x.A$, for every $x \in \sigma_o(X)$. In addition, for each $i > 1$, we add an edge $x_i.\rho_i \rightarrow x.A$ for every $x_i \in \sigma_o(X_i)$ and for every $x \in \sigma_o(X)$.
- **Type IV edges:** Consider any slot $X.\rho$ and partition attribute $Y.B \in \mathcal{P}[\rho]$ for $Y = \text{Range}[\rho]$. We define an edge $y.B \rightarrow x.S_\rho$ for every $x \in \sigma_o(X)$ and $y \in \sigma_o(Y)$.
- **Type V edges:** Consider any slot $X.\rho$. We define an edge $x.S_\rho \rightarrow x.\rho$ for every $x \in \sigma_o(X)$.

We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to an object skeleton σ_o if the directed graph G_{σ_o} is acyclic. ▮

Intuitively, type I edges correspond to intra-object dependencies and type II edges to inter-object dependencies. These are the same edges that we had in the dependency graph for regular PRMs, except that they also apply to selector attributes. Moreover, there is an important difference in our treatment of type II edges. In this case, the skeleton does not specify the value of $x.\rho$, and hence we cannot determine from the skeleton on which object y the attribute $x.A$ actually depends. Therefore, our instance dependency graph must include an edge from every attribute $y.B$.

Type III edges represent the fact that the actual choice of parent for $x.A$ depends on the value of the slots used to define it. When the parent is defined via a slot chain, the actual choice depends on the values of all the slots along the chain. Since we cannot determine the particular object from the skeleton, we must include an edge from every slot $x_i.\rho_i$ potentially included in the chain.

Type V edges represent the dependency of a slot on the attributes defining the associated partition. To see why this dependence is required, we observe that our choice of reference value for $x.\rho$ depends on the values of the partition attributes

$\mathcal{P}[x.\rho]$ of all of the different objects y in Y. Thus, these attributes must be determined before $x.\rho$ is determined. Finally, type V edges represent the fact that the actual choice of parent for $x.A$ depends on the value of the selector attributes for the slots used to define it. In our example, as $\mathcal{P}[\mathsf{Shows}.\mathit{Movie}] = \{\mathsf{Movie}.\mathit{Genre}\}$, the genres of all movies must be determined before we can select the value of the reference slot $\mathsf{Shows}.\mathit{Movie}$.

Based on this definition, we can specify conditions under which (5.2) specifies a coherent probability distribution.

Theorem 5.15

Let Π be a PRM with reference uncertainty whose dependency structure $\mathcal{S}$ is acyclic relative to an object skeleton σ_o. Then Π and σ_o define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_o via (5.2).

This theorem is limited in that it is very specific to the constraints of a given object skeleton. As in the case of PRMs without relational uncertainty, we want to learn a model in one setting, and be assured that it will be acyclic for any skeleton we might encounter. We accomplish this goal by extending our definition of class dependency graph. We do so by extending the class dependency graph to contain edges that correspond to the edges we defined in the instance dependency graph.

Definition 5.16

The *class dependency graph* G_Π for a PRM with reference uncertainty Π has a node for each descriptive or selector attribute $X.A$ and each reference slot $X.\rho$, and the following edges:

- **Type I edges:** For any attribute $X.A$ and formal parent $X.B$, we have an edge $X.B \rightarrow X.A$.
- **Type II edges:** For any attribute $X.A$ and formal parent $X.\rho.B$ where $\text{Range}[\rho] = Y$, we have an edge $Y.B \rightarrow X.A$.
- **Type III edges:** For any attribute $X.A$ and formal parent $Y.\boldsymbol{K}.B$, where $\boldsymbol{K} = \rho_1, \ldots, \rho_k$, and $\text{Dom}[\rho_i] = X_i$, we define an edge $X.\rho_1 \rightarrow X.A$. In addition, for each $i > 1$, we add an edge $X.\rho_i \rightarrow X.A$.
- **Type IV edges:** For any slot $X.\rho$ and partition attribute $Y.B$ for $Y = \text{Range}[\rho]$, we have an edge $Y.B \rightarrow X.S_\rho$.
- **Type V edges:** For any slot $X.\rho$, we have an edge $X.S_\rho \rightarrow X.\rho$.

Figure 5.9 shows the class dependency graph for our extended movie example.

While the proof is a bit more complex than in the attribute uncertainty case, the following analogous theorem holds:

Theorem 5.17

Let Π be a PRM with reference uncertainty whose class dependency structure $\mathcal{S}$ is acyclic. For any object skeleton σ_o, Π and σ_o define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_o via (5.2).

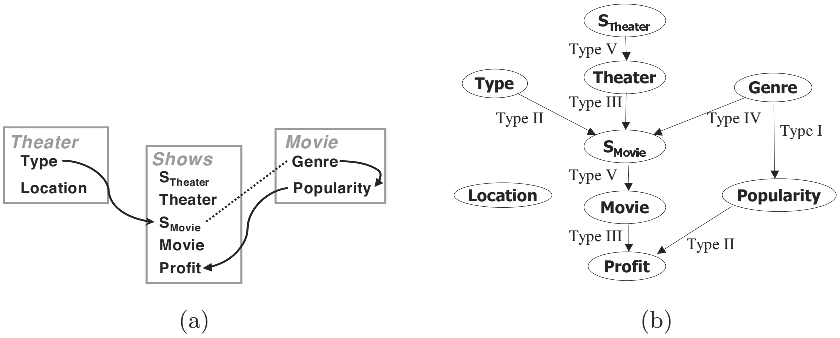


Figure 5.9 (a) A PRM for the movie theater example. The partition attributes are indicated using dashed lines. (b) The dependency graph for the movie theater example. The different edge types are labeled.

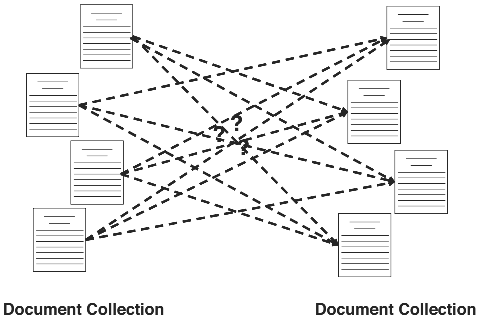
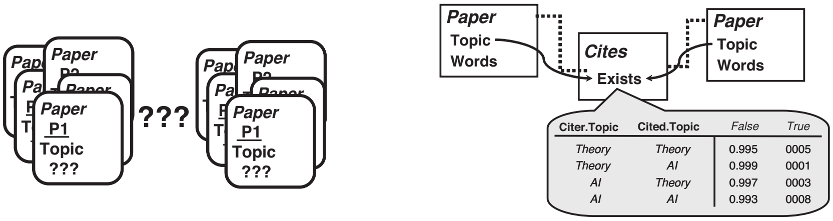


Figure 5.10 Existence uncertainty in a simple citation domain.

5.5.3 Existence Uncertainty

The second form of structural uncertainty we introduce is called *existence uncertainty*. In this case, we make no assumptions about the number of links that exist. The number of links that exist and the identity of the links are all part of the probabilistic model and can be used to make inferences about other attributes in our model. In our citation example above, we might assume that the set of papers is part of our background knowledge, but we want to provide an explicit model for the presence or absence of citations. Unlike the reference uncertainty model of the previous section, we do not assume that the total number of citations is fixed, but rather that each potential citation can be present or absent.



Citer.Topic	Cited.Topic	*False*	*True*
Theory	*Theory*	0.995	0005
Theory	*AI*	0.999	0001
AI	*Theory*	0.997	0003
AI	*AI*	0.993	0008

(a) (b)

Figure 5.11 (a) An entity skeleton for the citation domain. (b) A CPD for the *Exists* attribute of Cites.

5.5.3.1 Semantics of Relational Model

The object skeleton used for reference uncertainty assumes that the number of objects in each relation is known. Thus, if we consider a division of objects into entities and relations, the number of objects in classes of both types is fixed. Existence uncertainty assumes even less background information than specified by the object skeleton. Specifically, we assume that the number of relationship objects is not fixed in advance. This situation is illustrated in figure 5.10.

We assume that we are given only an *entity skeleton* σ_e, which specifies the set of objects in our domain only for the entity classes. Figure 5.11(a) shows an entity skeleton for the citation example. Our basic approach is to allow other objects within the model — those in the relationship classes — to be *undetermined*, i.e., their existence can be uncertain. In other words, we introduce into the model all of the objects that can *potentially* exist in it; with each of them, we associate a special binary variable that tells us whether the object actually exists or not. We call entity classes *determined* and relationship classes *undetermined*.

To specify the set of potential objects, 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 Cite class has the two reference slots, *Citing* and *Cited*. Thus the potential domain of the Cites class in a given instantiation $\mathcal{I}$ is $\mathcal{I}(\mathsf{Paper}) \times \mathcal{I}(\mathsf{Paper})$. Each "potential" object x in this class has the form $\mathsf{Cite}[y_1, y_2]$. Each such object is associated with a binary attribute $x.E$ that specifies whether paper y_1 did or did not cite paper y_2.

Definition 5.18
Consider a schema with determined and undetermined classes, and let σ_e be an entity skeleton over this schema. We define the *induced* relational skeleton, $\sigma_r[\sigma_e]$, to be the relational skeleton that contains the following objects:

- If X is a determined class, then $\sigma_r[\sigma_e](X) = \sigma_e(X)$.
- Let X be an undetermined class with reference slots $\rho_1, \ldots, \rho_k$ whose range types are $Y_1, \ldots, Y_k$ respectively. Then $\sigma_r[\sigma_e](X)$ contains an object $X[y_1, \ldots, y_k]$ for all tuples $\langle y_1, \ldots, y_k \rangle \in \sigma_r[\sigma_e](Y_1) \times \cdots \times \sigma_r[\sigma_e](Y_k)$.

The relations in $\sigma_r[\sigma_e]$ are defined in the obvious way: Slots of objects of determined classes are taken from the entity skeleton. Slots of objects of undetermined classes are induced from the object definition: $X[y_1, \ldots, y_k].\rho_i$ is y_i. ∎

To ensure that the semantics of schemata with undetermined classes is well-defined, we need a few tools. Specifically, we need to ensure that the set of potential objects is well-defined and finite. It is clear that if we allow cyclic references (e.g., an undetermined class with a reference to itself), then the set of potential objects is not finite. To avoid such situations, we need to put some requirements on the schema.

Definition 5.19
A set of classes $\mathcal{X}$ is *stratified* if there exists a partial ordering over the classes $\prec$ such that for any reference slot $X.\rho$ with range type Y, $Y \prec X$. ∎

Lemma 5.20
If the set of undetermined classes in a schema is stratified, then given any entity skeleton σ_e the number of potential objects in any undetermined class is finite.

As discussed, each undetermined X has a special *existence* attribute $X.E$ whose values are $\mathcal{V}(E) = \{true, false\}$. 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.

For a PRM with stratified undetermined classes, we define an instantiation to be an assignment of values to the attributes, including the *Exists* attribute, of *all* potential objects.

5.5.3.2 Probabilistic Model

We now specify the probabilistic model defined by the PRM. By treating the *Exists* attributes as standard descriptive attributes, we can essentially build our definition directly on top of the definition of standard PRMs.

Specifically, 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 has an associated CPD. figure 5.11(b) illustrates a CPD for the Cites.*Exists* attribute. In this example, the existence of a citation depends on the topic of the citing paper and the topic of the cited paper; e.g., it is more likely that citations will exist between papers with the same topic.

Using the induced relational skeleton and treating the existence events as descriptive attributes, we have set things up so that (5.1) applies with minor changes. There are two important changes to the definition of the distribution:

- We want to enforce that $x.E = false$ if $x.\rho.E = false$ for one of the slots ρ of X. Suppose that X has the slots $\rho_1, \ldots, \rho_k$, we define the *effective* CPD for $X.E$ as follows. Let $\mathrm{Pa}^*(X.E) = \mathrm{Pa}(X.E) \cup \{X.\rho_1.E, \ldots, X.\rho_k.E\}$, and define

$$P^*(X.E \mid \mathrm{Pa}^*(X.E)) = \begin{cases} P(X.E \mid \mathrm{Pa}(X.E)) & \text{if } X.\rho_i.E = \mathit{true}, \forall i = 1, \ldots, k, \\ 0 & \text{otherwise} \end{cases}$$

- We want to "decouple" the attributes of nonexistent objects from the rest of the PRM. Thus, if $X.A$ is a descriptive attribute, we define $\mathrm{Pa}^*(X.A) = \mathrm{Pa}(X.A) \cup \{X.E\}$, and

$$P^*(X.A \mid \mathrm{Pa}^*(X.A)) = \begin{cases} P(X.A \mid \mathrm{Pa}(X.A)) & \text{if } X.E = \mathit{true}, \\ \frac{1}{|\mathcal{V}(X.A)|} & \text{otherwise} \end{cases}$$

It is easy to verify that in both cases $P^*(X.A \mid \mathrm{Pa}^*(X.A))$ is a legal conditional distribution.

In effect, these constraints specify a new PRM Π^*, in which we treat $X.E$ as a standard descriptive attribute. For each attribute (including the *Exists* attribute), we define the parents of $X.A$ in Π^* to be $\mathrm{Pa}^*(X.A)$ and the associated CPD to be $P^*(X.A \mid \mathrm{Pa}^*(X.A))$.

Given an entity skeleton σ_e, a PRM with exists uncertainty Π specifies a distribution over a set of instantiations $\mathcal{I}$ consistent with $\sigma_r[\sigma_e]$:

$$P(\mathcal{I} \mid \sigma_e, \Pi) = P(\mathcal{I} \mid \sigma_r[\sigma_e], \Pi^*) = \prod_{X \in \mathcal{X}} \prod_{x \in \sigma_r[\sigma_e](X)} \prod_{A \in \mathcal{A}(x)} P^*(x.A \mid \mathrm{Pa}^*(x.A)) \tag{5.3}$$

We can similarly define the the class dependency graph for a PRM Π with exists uncertainty using the corresponding notions for the standard PRM Π^*. As there, we require that the class dependency graph G_{Π^*} is acyclic. One immediate consequence of this requirement is that the schema is stratified.

Lemma 5.21
If the class dependency graph G_{Π^*} is acyclic, then there is a stratification of the undetermined classes.

Based on this definition, we can prove the following result:

Theorem 5.22
Let Π be a PRM with existence uncertainty and an acyclic class dependency graph. Let σ_e be an entity skeleton. Then (5.3) defines a coherent distribution on all instantiations $\mathcal{I}$ of the induced relational skeleton $\sigma_r[\sigma_e]$.

5.6 PRMs with Class Hierarchies

Next we propose methods for discovering useful refinements of a PRM's dependency model. We begin by introducing *probabilistic relational models with class hierarchies* (PRMs-CH). PRMs-CH extend PRMs by including class hierarchies over the objects. Subclasses allow us to specialize the probabilistic model for some instances of a class. For example, if we have a class movie in our relational schema, we might consider subclasses of movies, such as documentaries, action movies, British comedies, etc. The popularity of an action movie (a subclass of movies) may depend on its budget, whereas the popularity of a documentary (another subclass of movies) may depend on the reputation of the director. Subclassing allows us to model probabilistic dependencies at the appropriate level of detail. For example, we can have the parents of the popularity attribute in the action movie subclass be different than the parents of the same attribute in the documentary subclass. In addition, subclassing allows additional dependency paths to be represented in the model that would not be allowed in a PRM that does not support subclasses. For example, whether a person enjoys action movies may depend on whether she enjoys documentaries. PRMs-CH provide a general mechanism that allow us to define a rich set of dependencies.

To motivate our extensions, consider a simple PRM for the movie domain. Let us restrict attention to the three classes, Person, Movie, and Vote. We can have the attributes of Vote depending on attributes of the person voting (via the slot Vote.*Voter*) and on attributes of the movie (via the slot Vote.*Movie*). However, given the attributes of all the people and the movie in the model, the different votes are (conditionally) i.i.d.

5.6.1 Class Hierarchies

Our aim is to refine the notion of a class, such as Movie, into finer subclasses, such as action movies, comedy, documentaries, etc. Moreover, we want to allow recursive refinements of this structure, so that we might refine action movies into the subclasses spy movies, car chase movies, and kung-fu movies.

A class hierarchy for a class X defines an IS-A hierarchy for objects from class X. The root of the class hierarchy is simply class X itself. The subclasses of X are organized into an inheritance hierarchy. The leaves of the class hierarchy describe basic classes—these are the most specific characterization of objects that occur in the database. The interior nodes describe abstractions of the base-level classes. The intent is that the class hierarchy is designed to capture useful and meaningful abstractions in a particular domain.

More formally, a hierarchy $H[X]$ for a class X is a rooted directed acyclic graph defined by a subclass relation $\prec$ over a finite set of subclasses $\mathcal{C}[X]$. For $c, d \in \mathcal{C}[X]$, if $c \prec d$, we say that X_c is a *direct subclass* of X_d, and X_d is a *direct superclass* of X_c. The root of the tree is the class X. $Class_\top$ corresponds to the original class X. We define $\prec^*$ to be the transitive closure of $\prec$; if $c \prec^* d$, we say that X_c is a subclass of X_d. For example, figure 5.12 shows the simple class hierarchy for the Movie class.

We denote the sublcasses of the hierarchy by $\mathcal{C}[(]H[X])$. We achieve subclassing for a class X by requiring that there be an additional subclass indicator attribute $X.Class$ that determines the subclass to which an object belongs. Thus, if c is a

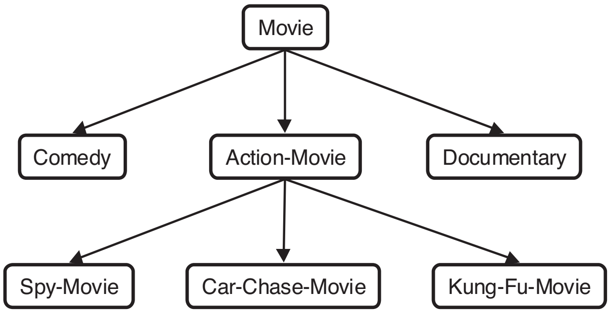


Figure 5.12 A simple class hierarchy for Movie.

subclass, then $\mathcal{I}(X_c)$ contains all objects $x \in X$ for which $x.\mathit{Class} \prec^* c$, i.e., all objects that are in some class which is a subclass of c. In our example, Movie has a subclass indicator variable Movie.*Class* with possible values

$$\{\mathit{Comedy}, \mathit{Action\text{-}Movie}, \mathit{Documentary}, \mathit{Spy\text{-}Movie}, \mathit{Car\text{-}Chase\text{-}Movie}, \mathit{Kung\text{-}Fu\text{-}Movie}\}$$

.

Subclasses allow us to make finer distinctions when constructing a probabilistic model. In particular, they allow us to *specialize* CPDs for different subclasses in the hierarchy.

Definition 5.23
A *probabilistic relational model with subclass hierarchy* is defined as follows. For each class $X \in \mathcal{X}$, we have

- a class hierarchy $H[X] = (\mathcal{C}[X], \prec)$;
- a subclass indicator attribute $X.\mathit{Class}$ such that $\mathcal{V}(X.\mathit{Class}) = \mathcal{C}[(]H[X])$;
- a CPD for $X.\mathit{Class}$;
- for each subclass $c \in \mathcal{C}[X]$ and attribute $A \in \mathcal{A}(X)$ we have either
    - a set of parents $\mathrm{Pa}^c(X.A)$ and a CPD that describes $P(X.A \mid \mathrm{Pa}^c(X.A))$; or
    - an *inherited* indicator that specifies that the CPD for $X.A$ in c is inherited from its direct superclass. The root of the hierarchy cannot have the inherited indicator. ▮

With the introduction of subclass hierarchies, we can refine our probabilistic dependencies. Before each attribute $X.A$ had an associated CPD. Now, if we like, we can specialize the CPD for an attribute within a particular subclass. We can associate a different CPD with the attributes of different subclasses. For example, the attribute Action-Movie.*Popularity* may have a different conditional distribution from the attribute Documentary.*Popularity*. Further, the distribution for each of the attributes may depend on a completely different set of parents. Continuing our earlier example, if the popularity of an action movie depends on its budget, then Action-Movie.*Popularity* would have as parents Action-Movie.*Budget*. However, for documentaries, the popularity depends on the reputation of the director; then Documentary.*Popularity* would have the parent Documentary.*Director*.*Reputation*.

We define $P(X.A \mid \mathrm{Pa}^c(X.A))$ to be the CPD associated with A in X_d, where d is the most specialized superclass of c (which may be c itself) such that the CPD of $X.A$ in d is not marked with the inherited indicator.

5.6.2 Refined Slot References

At first glance, the increase in representational power provided by supporting subclasses is deceptively small. It seems that little more than an extra constructed type variable has been added, and that the structure that is exploited by the new subclassed CPDs could just as easily have been provided using structured CPDs, such as the tree-structured CPDs or decision graphs [1, 4]. For example, the root node in the tree-structured CPD for attribute $X.A$ can split on the class attribute, $X.Class$, and then the subtrees can define the appropriate specializations of the CPD. In reality, it is not quite so simple; now $X.A$ would need to have as parents the union of all of the parents of its subclasses. However, the representational power is quite similar.

However, the representational power has been extended in a very important way. Certain dependency structures that would have been disallowed in the original framework are now allowed. These dependencies appear circular when examined only at the class level; however, when refined and modeled at the subclass level, they are no longer cyclic. One way of understanding this phenomenon is that, once we have refined the class, the subclass information allows us to disentangle and order the dependencies.

Returning to our earlier example, suppose that we have the classes Voter, Movie, and Vote. Vote has reference slots *Person* and *Movie* and an attribute *Ranking* that gives the score that a person has given for a movie. Suppose we want to model a correlation between a person's votes for documentaries and her votes for action movies. (This correlation might be a negative one.) In the unrefined model, we do not have a way of referring to a person's votes for some particular subset of movies; we can only consider aggregates over a person's entire set of votes. Furthermore, even if we could introduce such a dependence, the dependency graph would show a dependence of Vote.*Rank* on itself.

When we create subclasses of movie, we can also create specializations of any classes that make reference to movies. For example Vote has a reference slot Vote.*Movie*. Suppose we create subclasses of Movie: Comedy, Action-Movie, and Documentary. Then we can create corresponding specializations of Vote: Comedy-Vote, Action-Vote, and Documentary-Vote. Each of these subclasses refers only to a particular category of votes.

The introduction of subclasses of votes provides us with a way of isolating a person's votes on some subset of movies. In particular, we can try to introduce a dependence of Documentary-Vote.*Rank* on Action-Vote.*Rank*. In order to allow this dependency, we need a mechanism for constructing slot chains that restrict the types of objects along the path to belong to specific subclasses. Recall that a reference slot ρ is a function from Dom[ρ] to Range[ρ], i.e. from X to Y. We can introduce *refinements* of a slot reference by restricting the types of the objects in the range.

Definition 5.24
Let ρ be a slot (reference or inverse) of X with range Y. Let d be a subclass of Y. A *refined slot reference* $\rho_{\langle d \rangle}$ for ρ to d is a relation between X and Y:

$$\text{For } x \in X, y \in Y,\ y \in x.\rho_{\langle d \rangle} \text{ if } x \in X \text{ and } y \in Y_d, \text{ then } y \in x.\rho. \ \blacksquare$$

Returning to our earlier example, suppose that we have subclasses of Movie: Comedy, Action-Movie, and Documentary. In addition, suppose we also have subclasses of Vote, Comedy-Vote and Action-Vote, and Documentary-Vote. To get from a person to her votes, we use the inverse of slot reference Person.*Votes*. Now we can construct refinements of Person.*Votes*, $Votes_{\langle \textsf{Comedy-Vote} \rangle}$, $Votes_{\langle \textsf{Action-Vote} \rangle}$, and $Votes_{\langle \textsf{Documentary-Vote} \rangle}$.

Let us name these slots *Comedy-Votes* and *Action-Votes*, and *Documentary-Votes*. To specify the dependency of a person's rankings for documentaries on their rankings for action movies we can say that Documentary-Vote.*Rank* has a parent which is the person's action movie rankings: γ(Documentary-Vote.*Person.Action-Votes.Rank*).

5.6.3 Support for Instance-Level Dependencies

The introduction of subclasses brings the benefit that we can now provide a smooth transition from the PRM, a class-based probabilistic model, to models that are more similar to Bayesian networks. To see this, suppose our subclass hierarchy for movies is very "deep" and starts with the general class and ends in the most refined levels with particular movie instances. Thus, at the most refined version of the model we can define the preferences of a person by either class-based dependency (the probability of enjoying documentary movies depends on whether the individual enjoys action movies) or instance-based dependency (the probability of enjoying *Terminator II* depends on whether the individual enjoys *The Hunt for Red October*). The latter model is essentially the same as the Bayesian network models learned by Breese et al. [2] in the context of collaborative filtering for TV programs.

In addition, the new flexibility in defining refined slot references allows us to make interesting combinations of these types of dependencies. For example, whether an individual enjoys a particular movie(e.g., *True Lies*) can be enough to predict whether she watches a whole other category of movies (e.g., James Bond movies).

5.6.4 Semantics

Using this definition, the semantics for PRM-CH are given by the following equation:

$$P(\mathcal{I} \mid \sigma_r, \Pi) = \prod_X \prod_{x \in \sigma_r(X)} P(x.\mathit{Class}) \prod_{A \in \mathcal{A}(X)} P(x.A \mid \mathrm{Pa}^{x.c}(x.A)). \tag{5.4}$$

As before, the probability of an instantiation of the database is the product of CPDs of the instance attributes; the key difference is that here, in addition to the skeleton determining the parents on an attribute, the subclass to which the object belongs determines which local probability model is used.

5.6.5 Coherence of Probabilistic Model

As in the case of PRMs with attribute uncertainty, we must be careful to guarantee that our probability distribution is in fact coherent. In this case, while the relational skeleton specifies which objects are related to which, it does not specify the subclass indicator for each object, so the mapping of formal to actual parents depends on the probabilistic choice for the subclass for the object. In addition, for refined slot references, the existence of the edge will depend on the subclass of the object. We will indicate edge existence by the coloring of an edge: a *black* edge exists in the graph, a *gray* edge may exist in the graph, and a *white* edge is invisible in the graph. As in previous sections, we define our coherence constraints using an instance dependency graph, relative to our PRM and relational skeleton.

Definition 5.25

The *colored instance dependency graph* for a CH-PRM Π_{CH} and a relational skeleton σ_r is a graph G_{σ_r}. The graph has the following nodes, for each class X and for each $x \in \sigma_r(X)$:

- A descriptive attribute node $x.A$, for every descriptive attribute $X.A \in \mathcal{A}(X)$;
- a subclass indicator node $x.\mathit{Class}$.

Let $\mathrm{Pa}^*(X.A) = \bigcup_{c \in \mathcal{C}[X]} \mathrm{Pa}^c(X.A)$. The dependency graph contains four types of edges. For each attribute $X.A$ (both descriptive attributes and the subclass indicator), we add the following edges:

- **Type I edges:** For every $x \in \sigma_r(X)$ and for each formal parent $X.B \in \mathrm{Pa}^*(X.A)$, we define an edge $x.B \rightarrow x.A$. This edge is black if the parents have not been specialized (which will be the case for the subclass indicator, $x.\mathit{Class}$, and possibly other attributes as well). All the other edges are colored gray.
- **Type II edges:** For every $x \in \sigma_r(X)$ and for each formal parent $X.\boldsymbol{K}.B \in \mathrm{Pa}^*(X.A)$, if $y \in x.\boldsymbol{K}$ in σ_r, we define an edge $y.B \rightarrow x.A$. If the CPD has been specialized, or if $\boldsymbol{K}$ contains any refined slot references this edge is colored *gray*; otherwise is is colored *black*. ▮

As before, type I edges correspond to intra-object dependencies and type II edges to inter-object dependencies. But since an object may be from any subclass, even though the relational skeleton specifies the objects to which it is related, until we know the subclass of an object, we do not know which of the local probability models applies. In addition, in the case where a parent of an object is defined via a refined slot reference, we also do not know the set of related objects until we know their subclasses. Thus, we add edges for every *possible* parent and color the edges used in defining parents gray. Type I and type II edges are gray when they are parents in a specialized CPD. In addition, type II edges may be gray if a refined slot reference is used in the definition of a parent.

At this point, the problem with our instance dependency graph is that there are some edges which are known to occur (the black edges) and some edges that may or may not exist (depending on the subclass of an object). How do we ensure our instance dependency graph is acyclic? In this case, we must ensure that the instance dependency graph is acyclic for any setting of the subclass indicators. Note that this is a probabilistic event. First, we extend our notion of acyclicity for our colored instance dependency graph.

Definition 5.26

A colored instance dependency graph is acyclic if, for any instantiation of the subclass indicators, there is an acyclic ordering of the nodes relative to the black edges in the graph. Given any a particular assignment of subclass indicators, we determine the black edges as follows:

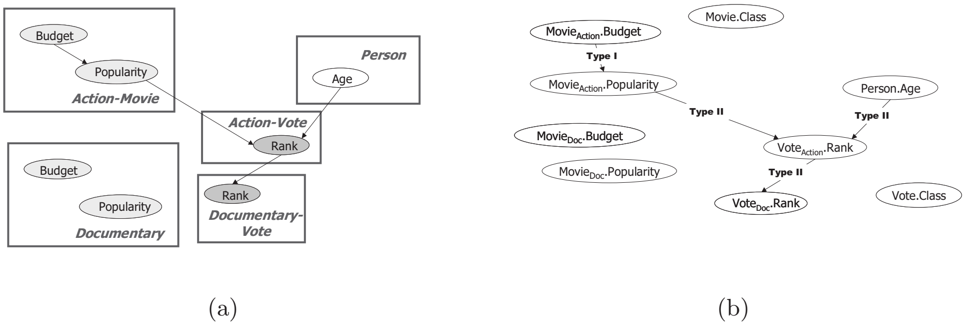
- Given a subclass assignment $y.Class$, all of the edges involving this object are colored either black or white. Let $y.Class = d$. The edges for any parent nodes are colored black if they are defined by the CPD $\text{Pa}^d(X.A)$, and white otherwise. In addition, the edges corresponding to any refined slot references, $\rho_{\langle d \rangle}(x, y)$, are set: If $y.Class = d$, the edge is colored black; otherwise it is painted white. ■

Based on this definition, we can specify conditions under which (5.4) specifies a coherent probability distribution.

Theorem 5.27

Let Π_{CH} be a PRM with class hierarchies whose colored dependency structure $\mathcal{S}$ is acyclic relative to a relational skeleton σ_r. Then Π_{CH} and σ_r define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_r via (5.4).

As in the previous case of PRMs with attribute uncertainty and PRMs with structural uncertainty, we want to learn a model in one setting, and be assured that it will be acyclic for any skeleton we might encounter. Again we achieve this goal through our definition of class dependency graph. We do so by extending the class dependency graph to contain edges that correspond to the edges we defined in the instance dependency graph.


Budget
Popularity
Action-Movie
Person
Age
Action-Vote
Rank
Budget
Popularity
Documentary
Rank
Documentary-Vote
Movie.Class
$\text{Movie}_{\text{Action}}$.Budget
Type I
$\text{Movie}_{\text{Action}}$.Popularity
Type II
Person.Age
Type II
$\text{Movie}_{\text{Doc}}$.Budget
$\text{Vote}_{\text{Action}}$.Rank
$\text{Movie}_{\text{Doc}}$.Popularity
Type II
Vote_{Doc}.Rank
Vote.Class

(a) (b)

Figure 5.13 (a) A simple PRM with class hierarchies for the movie domain. (b) The class dependency graph for this PRM.

Definition 5.28

The *class dependency graph* for a PRM with class hierarchy Π_{CH} has the following set of nodes for each $X \in \mathcal{X}$:

- for each subclass $c \in \mathcal{C}[X]$ and attribute $A \in \mathcal{A}(X)$, a node $X_c.A$;
- a node for the subclass indicator $X.Class$;

and the following edges:

- **Type I edges:** For any node $X_c.A$ and formal parent $X_c.B \in \text{Pa}^c(X_c.A)$ we have an edge $X_c.B \rightarrow X_c.A$.
- **Type II edges:** For any attribute $X_c.A$ and formal parent $X_c.\rho.B \in \text{Pa}^c(X_c.A)$, where $\text{Range}[\rho] = Y$, we have an edge $Y.B \rightarrow X_c.A$.
- **Type III edges:** For any attribute $X_c.A$, and for any direct superclass d, $c \prec d$, we add an edge $X_c.A \rightarrow X_d.A$. ▮

Figure 5.13 shows a simple class dependency graph for our movie example. The PRM-CH is given in figure 5.13(a) and the class dependency graph is shown in figure 5.13(b).

It is now easy to show that if this class dependency graph is acyclic, then the instance dependency graph is acyclic.

Lemma 5.29

If the class dependency graph is acyclic for a PRM with class hierarchies Π_{CH}, then for *any* relational skeleton σ_r, the colored instance dependency graph is acyclic.

And again we have the following corollary:

Corollary 5.30

Let Π_{CH} be a PRM with class hierarchies whose class dependency structure $\mathcal{S}$ is acyclic. For any relational skeleton σ_r, Π_{CH} and σ_r define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_r via (5.4).

5.7 Inference in PRMs

An important aspect of any probabilistic representation is the support for making inferences; having made some observations, how do we condition on these observations and update our probabilistic model? Inference in PRMs supports many interesting patterns of reasoning. Oftentimes we can view the inference as influence flowing between the interrelated objects. Consider a simple example of inference about a particular student in our school PRM. A priori we may believe a student is likely to be smart. We may observe his grades in several courses and see that for the most part he received As, but in one class he received a C. This may cause us to slightly reduce our belief that the student is smart, but it will not change it significantly. However, if we find that most of the other students that took the course received high grades, we then may believe that the course is an easy course. Since it is unlikely that a smart student got a low grade in an easy course, our probability for the student being smart now goes down substantially.

There are several potential approaches for performing inference effectively in PRMs. In a few cases, particularly when the skeleton is small, or it results in a network with low tree width, we can do exact inference in the ground Bayesian network. In other cases, when there are certain types of regularities in the ground Bayesian network, we can still perform exact inference by carefully exploiting and reusing computations. And in cases where the ground Bayesian network is very large and we cannot exploit regularities in its structure, we resort to approximate inference.

5.7.1 Exact Inference

We can always resort to exact inference on the ground Bayesian Network, but the ground Bayesian Network may be very large and thus this inference may prove intractable. Under certain circumstances, inference algorithms can exploit the model structure to make inference tractable. Previous work on inference in structured probabilistic models [14, 19, 18] shows how effective inference can be done for a number of different structured probabilistic models. The algorithms make use of the structure imposed by the class hierarchy to decompose the distribution and effectively reuse computation.

There are two ways in which aspects of the structure can be used to make inference more efficient. The first structural aspect is the natural encapsulation of objects that occurs in a well-designed class hierarchy. Ideally, the interactions between objects will occur via a small number of object attributes, and the majority of interactions between attributes will be encapsulated within the class. This can provide a natural decomposition of the model suitable for inference. The complexity of the inference will depend on the “width” of the connections between objects; if the width is small, we are guaranteed an efficient procedure.

The second structural aspect that is used to make inference efficient is the fact that similar objects occur many times in the model. Pfeffer et al. [19] describe a recursive inference algorithm that caches the computations that are done for fragments of the model; these computations then need only be performed once; we can reuse them for another object occurring in the same context. We can think of this object as a generic object, which occurs repeatedly in the model. Exploiting these structural aspects of the model allow Pfeffer et al. [19] to achieve impressive speedups; in a military battlespace domain the structured inference was orders of magnitudes faster than the standard Bayesian Network exact inference algorithm.

5.7.2 Approximate Inference

Unfortunately the methods used in the inference algorithm above often are not applicable for the PRMs we study. In the majority of cases, there are no generic objects that can be exploited. Unlike standard Bayesian Network inference, we cannot decompose this task into separate inference tasks over the objects in the model, as they are typically all correlated. Thus, inference in the PRM requires inference over the ground network defined by instantiating a PRM for a particular skeleton.

In general, the ground network can be fairly complex, involving many objects that are linked in various ways. (For example, in some of our experiments, the networks involve hundreds of thousands of nodes.) Exact inference over these networks is clearly impractical, so we must resort to approximate inference. We use belief propagation (BP), a local message-passing algorithm, introduced by Pearl [17]. The algorithm is guaranteed to converge to the correct marginal probabilities for each node only for singly connected Bayesian networks. However, empirical results [16] show that it often converges in general networks, and when it does the marginals are a good approximation to the correct posterior.

We provide a brief outline of one variant of BP, and refer the reader to [20, 16, 15] for more details. Consider a Bayesian network over some set of nodes (which in our case would be the variables $x.A$). We first convert the graph into a *family graph*, with a node F_i for each variable V_i in the Bayesian network, containing V_i and its parents. Two nodes are connected if they have some variable in common. The CPD of V_i is associated with F_i. Let ϕ_i represent the factor defined by the CPD; i.e., if F_i contains the variables $V, Y_1, \ldots, Y_k$, then ϕ_i is a function from the domains of these variables to $[0, 1]$. We also define ψ_i to be a factor over V_i that encompasses our evidence about V_i: $\psi_i(V_i) \equiv 1$ if V_i is not observed. If we observe $V_i = v$, we have that $\psi_i(v) = 1$ and 0 elsewhere. Our posterior distribution is then $\alpha \prod_i \phi_i \times \prod_i \psi_i$, where α is a normalizing constant.

The BP algorithm is now very simple. At each iteration, all the family nodes simultaneously send messages to all others, as follows:

$$m_{ij}(F_i \cap F_j) \leftarrow \alpha \sum_{F_i - F_j} \phi_i \cdot \psi_i \cdot \prod_{k \in N(i) - \{j\}} m_{ki},$$

where α is a (different) normalizing constant and $N(i)$ is the set of families that are neighbors of F_i in the family graph. This process is repeated until the beliefs converge. At any point in the algorithm, our marginal distribution about any family F_i is $b_i = \alpha \cdot \phi_i \cdot \psi_i \cdot \prod_{k \in N(i)} m_{ki}$. Each iteration is linear in the number of edges in the Bayesian network. While the algorithm is not guaranteed to converge, it typically converges after just a few iterations. After convergence, the b_i give us approximate marginal distributions over each of the families in the ground network.

5.8 Learning

Next, we turn our attention to learning a PRM. In the learning problem, our input contains a relational schema that describes the basic vocabulary in the domain — the set of classes, the attributes associated with the different classes, and the possible types of relations between objects in the different classes. For simplicity, in the description that follows, we assume the training data consists of a fully specified instance of that schema; if there are missing values, then an expectation maximization (EM) algorithm is needed as well. We begin by describing learning PRMs with attribute uncertainty, next describe the extensions to support learning PRMs with structural uncertainty, and then describe support for learning PRMs with class hierarchies.

We assume that the training instance is given in the form of a relational database. Although our approach would also work with other representations (e.g., a set of ground facts completed using the closed-world assumption), the efficient querying ability of relational databases is particularly helpful in our framework, and makes it possible to apply our algorithms to large data sets.

There are two components of the learning task: parameter estimation and structure learning. In the parameter estimation task, we assume that the qualitative dependency structure of the PRM is known; i.e., the input consists of the schema and training database (as above), as well as a qualitative dependency structure $\mathcal{S}$. The learning task is only to fill in the parameters that define the CPDs of the attributes. In the structure learning task, the dependency structure is not provided (although the user can, if available, provide prior knowledge about the structure, e.g., in the form of constraints) and the goal is to extract an entire PRM, structure as well as parameters, from the training database alone. We discuss each of these problems in turn.

5.8.1 Parameter Estimation

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. The likelihood of a parameter set is defined to be the probability of the data given the model. For a PRM, the likelihood of a parameter set $\theta_{\mathcal{S}}$ is: $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) = P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}})$. As usual, we typically work with the log of this function:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \log P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} \log P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \right]. \end{aligned} \tag{5.5}$$

The key insight is that this equation is very similar to the log-likelihood of data given a Bayesian network [11]. In fact, it is the likelihood function of the Bayesian network induced by the PRM 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—the parameters are *shared* or *tied*.

5.8.1.1 Maximum Likelihood Parameter Estimation

We can still 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 $\mathcal{I}$, σ 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:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} \log P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \right] \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \sum_{v \in \mathcal{V}(X.A)} \sum_{\mathbf{u} \in \mathcal{V}(\mathrm{Pa}X.A)} C_{X.A}[v, \mathbf{u}] \cdot \log \theta_{v \mid \mathbf{u}} \end{aligned} \tag{5.6}$$

where $C_{X.A}[v, \mathbf{u}]$ is the number of times we observe $\mathcal{I}_{x.A} = v$ and $\mathcal{I}_{\mathrm{Pa}(x.A)} = \mathbf{u}$ Each of the terms in the above sum can be maximized independently of the rest. Hence, maximum likelihood estimation reduces to independent maximization problems, one for each CPD.

For many parametric models, such as the exponential family, maximum likelihood estimation can be done via *sufficient statistics* that summarize the data. In the case of multinomial CPDs, these are just the counts we described above, $C_{X.A}[v, \mathbf{u}]$, the number of times we observe each of the different values $v, \mathbf{u}$ that the attribute $X.A$ and its parents can jointly take.

An important property of the database setting is that we can easily compute sufficient statistics. To compute $C_{X.A}[v, v_1, \ldots, v_k]$, we simply query over the class

X and its parents' classes, and project onto the appropriate set of attributes. For example, to learn the parameters for the grade CPD from our school example, we can compute the sufficient statistics with the following SQL query:

```
SELECT grade, intelligence, difficulty, count(*)
FROM from registration, student, course
GROUP BY grade, intelligence, difficulty
```

In some cases, it is useful to materialize a view that can be used to compute the sufficient statistics. This is beneficial when the relationship between the child attribute and the parent attribute is many-one rather than one-one or one-many. For example, consider the dependence of attributes of Student on attributes of Registration. In our example PRM, a student's ranking depends on the student's grades. In this case we would construct a view using the following SQL query:

```
CREATE VIEW v1
SELECT student.*, AVERAGE(grade) AS ave_grade,
      AVERAGE(satisfaction) as ave_satisfaction
FROM student s, registration r
WHERE s.student_id = r.student
```

To compute the statistics we would then project on the appropriate attributes from view v1:

```
SELECT ranking, ave_grade, COUNT(*)
FROM v1
GROUP BY ranking, ave_grade
```

Thus both the creation of the view and the process of counting occurrences can be computed using simple database queries, and can be executed efficiently. The view creation for each combination of classes is done once during the full learning algorithm (we will see exactly at which point this is done in the next section when we describe the search). If the tables being joined are indexed on the appropriate set of foreign keys, the construction of this view is efficient: the number of rows in the resulting table is the size of the child attribute's table; in our example this is |Student|. Computing the sufficient statistics can be done in one pass over the resulting table. The size of the resulting table is simply the number of unique combinations of attribute values. We are careful to cache sufficient statistics so they are only computed once. In some cases, we can compute new sufficient statistics from a previously cached set of sufficient statistics; we make use of this in our algorithm as well.

5.8.1.2 Bayesian Parameter Estimation

In many cases, maximum likelihood parameter estimation is not robust: 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 5.8.2, the Bayesian framework also gives us a good metric for evaluating the quality of different candidate structures.

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 distributions $\theta_{X.A|\mathbf{u}}$ 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 [11], we make two assumptions. First, we assume *parameter independence*: the priors over the parameters $\theta_{X.A|\mathbf{u}}$ for the different $X.A$ and $\mathbf{u}$ are independent. Second, we assume that the prior over $\theta_{X.A|\mathbf{u}}$ is a *Dirichlet* distribution. Briefly, a Dirichlet prior for a multinomial distribution of a variable V is specified by a set of *hyperparameters* $\{\alpha[v] : v \in \mathcal{V}(V))\}$. A distribution on the parameters of $P(V)$ is Dirichlet if

$$P(\theta_V) \propto \prod_v \theta_v^{\alpha[v]-1}.$$

(For more details see [7].) If $X.A$ can take on k values, then the prior is

$$P(\theta_{X.A|\mathbf{u}}) = \mathrm{Dir}(\theta_{X.A|\mathbf{u}} \mid \alpha_1, \ldots, \alpha_k).$$

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 $\theta_{X.A|\mathbf{u}}$. In other words,

$$P(\theta_{X.A|\mathbf{u}} \mid \mathcal{I}, \sigma, \mathcal{S}) = \mathrm{Dir}(\theta_{X.A|\mathbf{u}} | \alpha_{X.A}[v_1, \mathbf{u}] + C_{X.A}[v_1, \mathbf{u}], \ldots, \alpha_{X.A}[v_k, \mathbf{u}] + C_{X.A}[v_k, \mathbf{u}]).$$

Now that we have the posterior, we can compute the probability of new data. In the case where the new instance is conditionally independent of the old instances given the parameter values (which is always the case in Bayesian network models, but may not be true here), then the probability of the new data case can be conveniently rewritten using the expected parameters:

Proposition 5.31
Assuming multinomial CPDs, prior independence, and Dirichlet priors, with hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$, we have that

$$E_\theta[P(X.A = v \mid \mathrm{Pa}(X.A) = \mathbf{u}) \mid \mathcal{I}] = \frac{C_{X.A}[v, \mathbf{u}] + \alpha_{X.A}[v, \mathbf{u}]}{\sum_{i=1}^{k} C_{X.A}[v_i, \mathbf{u}] + \alpha_{X.A}[v_i, \mathbf{u}]}.$$

This suggests that the Bayesian estimate for $\theta_{\mathcal{S}}$ should be estimated using this formula as well. Unfortunately, the expected parameter is not the proper Bayesian solution for computing probability of new data in the case where the new data instance is not independent of previous data given the parameters. Suppose that we want to use the posterior to evaluate the probability of an instance $\mathcal{I}'$ of another skeleton σ'. If there are two instances x and x' of the class X such that $v^{\mathcal{I}'}(\mathrm{Pa}(x.A)) = v^{\mathcal{I}'}(\mathrm{Pa}(x'.A))$, then we will be relying on the same multinomial parameter vector twice. Using the chain rule, we see that the second probability depends on the posterior of the parameters after seeing the training data, *and* the first instance. In other words, the probability of a relational database given a distribution over parameter values is not identical to the probability of the data set when we have a point estimate of the parameters (i.e., when we act as though we know their values). However if the posterior is sharply peaked (i.e., we have a strong prior, or we have seen many training instances), we can approximate the solution using the expected parameters of proposition 5.31. We use this approximation in our computation of the estimates for the parameters.

5.8.1.3 Structure Learning

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.

5.8.1.4 Legal Structures

We saw in section 5.2.5.2 that we could construct a class dependency graph for a PRM, and the PRM defined a coherent probability distribution if the class dependency graph was stratified. During learning it is straightforward to maintain this structure, and consider only models whose dependency structure passes this test.

Maintaining a stratified class dependency graph Given a stratified class dependency graph $G(V, E)$, we can check whether local changes to the structure destroy the stratification. The operations we are concerned with are ones which add an edge (u, v) into the structure (clearly deleting an edge cannot introduce a cycle). We can check whether a new edge will introduce a cycle in time $O(|V| + |E|)$.

Let $G(V, E)$ be our stratified class dependency graph and let $G'(V, E \cup \{(u, v)\})$ be the class dependency graph with edge (u, v) added. Clearly if there is a cycle in G', it must contain (u, v).

We can check whether the new edge introduces a cycle by checking to see if, using this edge, there is a path $u, v, \ldots, u$. This reduces to checking to see if there is a path in the graph from v to u. We can do a a simple depth-first search to explore the graph to check for a path in $O(|V| + |E|)$.

5.8.2 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. We would like to find the MAP (maximum a posteriori) structure. 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 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.[2] 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 $\boldsymbol{K}$ 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 [12].

Proposition 5.32

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

[2] Although there are only a finite number that are reasonable to consider for a given skeleton.

given $\mathcal{S}$ is

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

$$\prod_i \prod_{A \in \mathcal{A}(X_i)} \left[\prod_{\mathbf{u} \in \mathcal{V}(()\mathrm{Pa}(X_i.A))} \mathrm{DM}(\{C_{X_i.A}[v, \mathbf{u}]\}, \{\alpha_{X_i.A}[v, \mathbf{u}]\}) \right], \qquad (5.7)$$

where $\mathrm{DM}(\{C[v]\}, \{\alpha[v]\}) = \frac{\Gamma(\sum_v \alpha[v])}{\Gamma(\sum_v (\alpha[v] + C[v])} \prod_v \frac{\Gamma(\alpha[v] + C[v])}{\Gamma(\alpha[v])}$, 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 $\mathbf{u} \in \mathcal{V}(\mathrm{Pa}(X.A))$. 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 $v \in \mathcal{V}(X.A)$.

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 seen explicitly in the asymptotic relation of the marginal likelihood to explicit penalization, such as the minimum description length (MDL) score (see, e.g., [11]).

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 [12]. 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.

5.8.3 Structure Search

Now that we have both 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 [3]. As PRM learning is at least as hard as Bayesian network learning (a Bayesian network 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.

As is standard in Bayesian network learning [11], we use a greedy local search procedure that maintains a "current" candidate structure and iteratively modifies it to increase the score. At each iteration, we consider a set of simple local transformations to the current structure, score all of them, and pick the one with the highest score. In the case where we are learning multinomial CPDs, the three operators we use are: *add edge*, *delete edge*, and *reverse edge*. In the case where we are learning tree CPDs, following [4], our operators consider only transformations to the CPD trees. The tree structure induces the dependency structure, as the parents of $X.A$ are simply those attributes that appear in its CPD tree. In this case, the two operators we use are: *split* — replaces a leaf in a CPD tree by an internal node with two leafs; and *trim* — replaces the subtree at an internal node by a single leaf.

The simplest heuristic search algorithm is a 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 the appropriate set of local transformations to that structure, score all of them, and pick the one with highest score.

We refer to this simple algorithm as the greedy algorithm. There are several common variants to improve the robustness of hill-climbing methods. One is is to make use of random restarts to deal with local maxima. In this algorithm, when we reach a local maximum, we take some fixed number of random steps, and then we restart our search process. Another common approach is to make use of a tabulist, which keeps track of the most recent states visited, and allows only steps which do not return to a recently visited state. A more sophisticated approach is to make use of a simulated annealing style of algorithm which uses the following procedure: in the early phases of the search we are likely to take random steps (rather than the best step), but as the search proceeds (i.e., the temperature cools) we are less likely to take random steps and more likely to take the best greedy step. The algorithms we have used are either the simple greedy algorithm or a simple randomized algorithm.

Regardless of the specific heuristic search algorithm used, an important component of the search is the scoring of candidate structures. 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. (This local score of an individual attribute is exactly the logarithm of the term in square brackets in (5.7).) 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.

However, there are still a very large number of possible structures to consider. We propose a heuristic search algorithm that addresses this issue. At a high level, the algorithm proceeds in phases. At each phase k, we have a set of potential parents $Pot_k(X.A)$ 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 $Pot_k(X.A)$. 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 be done very efficiently.

The success of this approach depends on the choice of the potential parents. Clearly, a bad initial choice can result to poor structures. Following [8], 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 attrributes may require aggregation; we may either specify the appropriate aggregator or search over the space of possible aggregators.

This scheme expands the set of potential parents at each iteration. In some cases, however, it may result in large set of potential parents. In such cases we may want to 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 by Friedman et al. [8] in the context of learning Bayesian networks. These 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 $\gamma(X.\boldsymbol{K}.B)$ to its current parents. We can then choose the highest scoring of these, as well as the current parents, to be the new set of potential parents. This approach would allow us to significantly reduce the size of the potential parent set, and thereby of the resulting view, while typically causing insignificant degradation in the quality of the learned model.

5.8.4 Learning PRMs with Structural Uncertainty

Next, we describe how to extend the basic PRM learning algorithm to deal with structural uncertainty. For PRMs with reference uncertainty, in addition we also attempt to learn the rules that govern the link models. For PRMs with existence uncertainty we learn the probability of existence of relationship objects.

5.8.4.1 Learning with Reference Uncertainty

The extension to scoring required to deal with reference uncertainty is not a difficult one. Once we fix the partitions defined by the attributes $\mathcal{P}[\rho]$, a CPD for S_ρ compactly defines a distribution over values of ρ. Thus, scoring the success in predicting the value of ρ can be done efficiently using standard Bayesian methods used for attribute uncertainty (e.g., using a standard Dirichlet prior over values of ρ).

The extension to search the model space for incorporating reference uncertainty involves expanding our search operators to allow the addition (and deletion) of attributes to partition definition for each reference slot. 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 $\mathcal{P}[\rho]$. We introduce two new operators, **refine** and **abstract**, which modify the partition by adding and deleting attributes from $\mathcal{P}[\rho]$. Initially, $\mathcal{P}[\rho]$ is empty for each ρ. The **refine** operator adds an attribute into $\mathcal{P}[\rho]$; the **abstract** operator deletes one. As mentioned earlier, we can define the partition simply by looking at the cross product of the values for each of the partition attributes, or using a decision tree. In the case of a decision tree, **refine** adds a split to one of the leaves and **abstract** removes a split. These newly introduced operators are treated by the search algorithm in exactly the same way as the standard edge-manipulation operators: 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 or partition set of an attribute that ω modifies. This is also true when we consider operators that modify the parent of selector attributes.

5.8.4.2 Learning with Existence Uncertainty

The extension of the Bayesian score to PRMs with existence uncertainty is straight forward; the exists attribute is simply a new descriptive attribute. The only new issue is how to compute sufficient statistics that include existence attributes $x.E$ without explicitly enumerating all the nonexistent entities. We perform this computation by counting, for each possible instantiation of $\text{Pa}(X.E)$, the number of potential objects with that instantiation, and subtracting the actual number of objects x with that parent instantiation.

Let $\mathbf{u}$ be a particular instantiation of $\text{Pa}(X.E)$. To compute $C_{X.E}[\mathit{true}, \mathbf{u}]$, we can use a standard database query to compute how many objects $x \in \sigma(X)$ have $\text{Pa}(x.E) = \mathbf{u}$. 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 $(x_1, \ldots, x_k) \in \mathcal{I}(Y_1) \times \cdots \mathcal{I}(Y_k)$ by decomposing the computation as follows: Let ρ be a reference slot of X with $\text{Range}[\rho] = Y$. Let $\text{Pa}_\rho(X.E)$ 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 $\text{Pa}_\rho(X.E)$ is empty, this count is simply $|\mathcal{I}(Y)|$. 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.

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. Our search algorithm now considers operators that add, delete, or reverse edges involving the exist attributes. As usual, we enforce coherence using the class dependency graph. In addition to having an edge from $Y.E$ to $X.E$ for every slot $\rho \in \mathcal{R}(X)$ whose range type is Y, when we add an edge from $Y.B$ to $X.A$, we add an edge from $Y.E$ to $X.E$ and an edge from $Y.E$ to $X.A$.

5.8.5 Learning PRM-CHs

We now turn to learning PRMs with class hierarchies. We examine two scenarios: in one case the class hierarchies are given as part of the input and in the other, in addition to learning the PRM, we also must learn the class hierarchy. The learning algorithms use the same criteria for scoring the models; however, the search space is significantly different.

5.8.6 Class Hierarchies Provided in the Input

We begin with the simpler learning with class hierarchies scenario, where we assume that the class hierarchy is given as part of the input. As in section 5.8, we restrict attention to fully observable data sets. Hence, we assume that, in our training set, the class of each object is given. Without this assumption, the subclass indicator attribute would play the role of a hidden variable, greatly complicating the learning algorithm.

As discussed above, we need a scoring function that allows us to evaluate different candidate structures, and a search procedure that searches over the space of possible structures. The scoring function remains largely unchanged. For each object x in each class X, we have the basic subclass c to which it belongs. For each attribute A of this object, the probabilistic model then specifies the subclass d of X from which c inherits the CPD of $X.A$. Then $x.A$ contributes only to the sufficient statistics for the CPD of $X_d.A$. With that recomputation of the sufficient statistics, the Bayesian score can now be computed unchanged.

Next we extend our search algorithm to make use of the subclass hierarchy. First, we extend our phased search to allow the introduction of new subclasses. Then, we introduce a new set of operators. The new operators allow us to refine and abstract the CPDs of attributes in our model, using our class hierarchy to guide us.

5.8.6.1 Introducing New Subclasses

New subclasses can be introduced at any point in the search. We may construct all the subclasses at the start of our search, or we may consider introducing them more gradually, perhaps at each phase of the search. Regardless of when the new subclasses are introduced, the search space is greatly expanded, and care must be taken to avoid the construction of an intractable search problem. Here we describe the mechanics of the introduction of the new subclasses.

For each new subclass introduced, each attribute for the subclass is associated with a CPD. A CPD can be marked as either "inherited" or "specialized." Initially, only the CPD for attributes of $X_\top$ are marked as specialized; all the other CPDs are inherited. Our original search operators — those that add and delete parents — can be applied to attributes at all levels of the class hierarchy. However, we only allow parents to be added and deleted from attributes whose CPDs have been specialized. Note that any change to the parents of an attribute is propagated to any descendents of the attribute whose CPDs are marked as inherited from this attribute.

Next, we introduce the operators Specialize and Inherit. If $X_c.A$ currently has an inherited CPD, we can apply Specialize(X_c.A). This has two effects. First, it recomputes the parameters of that CPD to utilize only the sufficient statistics of the subclass c. To understand this point, assume that $X_c.A$ was being inherited from X_d prior to the specialization. The CPD of $X_d.A$ was being computed using all objects in $\mathcal{I}(X_d)$. After the change, the CPD will be computed using just the objects in $\mathcal{I}(X_c)$. The second effect of the operator is that it makes the CPD modifiable, in that we can now add new parents or delete them. The Inherit operator has the opposite effect.

In addition, when a new subclass is introduced, we construct new refined slot references that make use of the subclass. Let D be a newly introduced subclass of Y. For each reference slot ρ of some class X with range Y, we introduce a new refined slot reference $\rho_{\langle D \rangle}$. In addition, we add each reference slot of Y to D; however, we refine the domain from Y to D. In other words, if we have the new reference slot ρ', where $\mathrm{Dom}[\rho'] = D$ and $\mathrm{Range}[\rho'] = X$.

5.8.6.2 Learning Subclass Hierarchies

We next examine the case where the subclass hierarchies are not given as part of the input. In this case, we will learn them at the same time we are learning the PRM.

As above, we wish to avoid the problem of learning from partially observable data. Hence, we need to assume that the basic subclasses are observed in the training set. At first glance, this requirement seems incompatible with our task definition: if the class hierarchy is not known, how can we observe subclasses in the training data? We resolve this problem by defining our class hierarchy based on the standard class attributes. For example, movies might be associated with an attribute specifying the genre — action, drama, or documentary. If our search algorithm decides that this attribute is a useful basis for forming subclasses, we would define subclasses based in a deterministic way on its values. Another attribute might be the reputation of the director. The algorithm might choose to refine the class hierarchy by partitioning sitcoms according to the values of this attribute. Note that, in this case, the class hierarchy depends on an attribute of a related class, not the class itself.

We implement this approach by requiring that the subclass indicator attribute be a deterministic function of its parents. These parents are the attributes used to define the subclass hierarchy. In our example, Movie.*Class* would have as parents Movie.*Genre* and Movie.*Director.Reputation*. Note that, as the function defining the subclass indicator variable is required to be deterministic, the subclass is effectively observed in the training data (due to the assumption that all other attributes are observed).

We restrict attention to decision-tree CPDs. The leaves in the decision tree represent the basic subclasses, and the attributes used for splitting the decision tree are the parents of the subclass indicator variable. We can allow binary splits that test whether an attribute has a particular value, or, if we find it necessary, we can allow a split on all possible values of an attribute.

The decision tree gives a simple algorithm for determining the subclass of an object. In order to build the decision tree during our search, we introduce a new operator $\mathsf{Split}(X, c, X.\boldsymbol{K}.B)$, where c is a leaf in the current decision tree for $X.\mathit{Class}$ and $X.\boldsymbol{K}.B$ is the attribute on which we will split that subclass.

Note that this step expands the space of models that can be considered, but in isolation does not change the score of the model. Thus, if we continue to use a purely greedy search, we would never take these steps. There are several approaches for addressing this problem. One is to use some lookahead for evaluating the quality of such a step. Another is to use various heuristics for guiding us toward worthwhile splits. For example, if an attribute is the common parent of many other attributes within X_c, it may be a good candidate on which to split.

The other operators, Specialize and Inherit, remain the same; they simply use the subclasses defined by the decision tree.

5.9 Conclusion

In this chapter we have described a comprehensive framework for learning a statistical model from relational data. We have presented a method for the automatic construction of a PRM from an existing database. Our method learns a structured statistical model directly from the relational database, without requiring the data to be flattened into a fixed attribute-value format. We have shown how to perform parameter estimation, developed a scoring criterion for use in structure selection, and defined the model search space. We have also provided algorithms for guaranteeing the coherence of the learned model.

References

[1] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1996.

[2] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1998.

[3] D. Chickering. Learning Bayesian networks is NP-complete. In *Artificial Intelligence and Statistics*, 1996.

[4] D. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1997.

[5] D. Cohn and T. Hofmann. The missing link—a probabilistic model of document content and hypertext connectivity. In *Proceedings of Neural Information Processing Systems*, 2001.

[6] 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 *Proceedings of the National Conference on Artificial Intelligence*, 1998.

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

[8] N. Friedman, I. Nachman, and D. Peér. Learning of Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1999.

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

[10] L. Getoor and J. Grant. PRL: A probabilistic relational language. *Machine Learning Journal*, 62(1-2):7–31, 2006.

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

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

[13] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1998.

[14] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1997.

[15] D. MacKay, R. McEliece, and J. Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communication*, 16(2):140–152, 1997.

[16] K. Murphy and Y. Weiss. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1999.

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

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

[19] A. Pfeffer, D. Koller, B. Milch, and K. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1999.

[20] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.