LEARNING STATISTICAL MODELS FROM RELATIONAL DATA

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Lise Getoor
June 2002

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Daphne Koller
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Nir Friedman
(Hebrew University)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Nils Nilsson

Approved for the University Committee on Graduate Studies:

To my father, for his love and support.

Acknowledgments

First, I would like to thank my adviser Daphne Koller for her support throughout my years at Stanford. Daphne has taught me many things including tenacity, the importance of holding high-standards, and the art of choosing compelling research problems. I've been inspired by her brilliance, hard work, and dedication and have benefitted greatly from her guidance and example.

I would also like to thank my 'co-adviser', Nir Friedman. Nir's support and patience still astound me. In addition, his technical depth and attention to detail were instrumental in the development of this work. I was very lucky to have crossed paths with Nir early in my career and am grateful that he has continued to find the time to collaborate, despite being halfway around the world.

I would also like to thank Nils Nilsson for his warm and wise counsel throughout my years at Stanford. I've greatly appreciated Nil's enthusiastic support, from the time I was the TA for his machine learning class, through his final reading of my thesis.

The work in this thesis is the product of collaborations with a number of people. Both Daphne and Nir have not just supervised this work, but have been intimately involved in its development. In addition, I've had the good fortune of collaborating with two truly phenomenal colleagues. In many ways, my thesis builds upon the foundational work that Avi Pfeffer put forth in his thesis. Avi and I helped each other through those first years of grad school, including both the intellectual and personal challenges. He is not only an inspiring thinker—someone who is not afraid to tackle the 'big' problems in research—but a close friend as well. More recently, I have had the pleasure of collaborating with Benjamin Taskar. He has contributed

greatly to many of the ideas presented in this thesis. In addition to his keen intellect, he is a pleasure to work with. His unfailing good humor, amazing productivity and easy-going nature make him the ideal colleague. I would like to thank him for the many hours we've spent hacking together, working through theories together, and crunching before deadlines.

I'd also like to thank the members of DAGS (Daphne's Approximate Groups of Students) for their technical feedback and for all our great DAGS adventures. We've had a truly outstanding collection of individuals as members during my years in the group. I would like to thank Drago Anguelov, Xavier Boyen, Urszula Chajewska, Barbara Engalhart, Raya Fratkina, Carlos Guestrin, Manfred Jaeger, Alex Kozlov, Uri Lerner, Uri Nodelman, Dirk Ormoneit, Ron Parr, Mehran Sahami, Eran Segal, Ben Taskar and Simon Tong

An additional thanks goes to Uri Lerner— we truly created a monster when we created the software infrastructure for PHROG (Probabilistic Reasoning and Optimizing Gadgets). I hope our karma will not be forever damaged by the pain we inflicted on later members of DAGs. I'd like to thank Uri for his drive to make the system come into being, and all the hard work that he put in building the infrastructure from which so many different DAGs members have benefitted.

I would also like to thank Eran Segal, first, for his patience in wading through the PHROG PRM code I wrote and second, for the many extensions, tools and bug fixes he developed, which have been useful for my own work. Thanks to Eran too for being a great colleague and office-mate.

Another group of students that was very important to me during my stay at Stanford was the Quail study group: Eyal, Patrick, Pedrito, Avi and Urszula. The purpose of the group was to study for the qualifier. In the long hours that we studied, we not only learned the material, but learned a lot about each other's intellects and how we each approached problems. We also garnered a lot of emotional support and camaraderie from each other as well.

On a more personal note, I've had the support of a number of outstanding friends and colleagues throughout the years.

I'd like to begin foremost by thanking Amy Lansky. Her support and encouragement were instrumental in my decision to return to graduate school to pursue my PhD. Amy's example as an independent thinker and her role as a mentor when I worked for her at NASA have been an invaluable source of inspiration. Throughout my stay at Stanford, she has always been a great friend and cheerleader for me and I was very lucky to have someone like her rooting for me. More recently, in my last months of working on my thesis, Amy and her family have been been instrumental again, by giving me a space to work in and a fast Internet connection. I would like to thank them for their great generosity.

Rob Holte has been a great source of encouragement. His guidance and advice always proved useful, and I've particularly relied on it during these last few months and days of finishing my thesis. I would like to thank him for all his time and wise counsel.

Barney Pell strongly influenced my decision to return to pursue my PhD, both through his encouragement and enthusiastic example. Alon Levy has also been a great source of encouragement, as has Fausto Gunchiglia.

In addition, I'd like to thank a number of other NASA friends and colleagues. Peter Friedland, Steve Minton and Pandu Nayak were especially supportive of my decision to return to graduate school. I'd also like to thank all my NASA cronies whose example inspired my return: John Allen, Hamid Berenji, John Bresina, Wray Buntine, Peter Cheeseman, Jeremy Frank, Mark Friedman, Rich Levinson, Mike Lowry, Nicola Muscettola, and Andrew Philpot.

Other Stanford associates I'd like to thank include: Berthe Choueiry, Mike Genesereth, Illah Nourbakhsh, Ofer Matan, Sheila McIlraith, Ron Parr, Wanda Pratt, Jeanne Rhee, Tamara Munzner, and Diane Tang. Xerox PARC associates that I'd like to thank include Markus Fromherz and Greger Ottosson.

Going even further back, I'd like to thank Stuart Russell for his support while I did my master's with him at UC Berkeley. I'd also like to thank my long-time friend and colleague Marie desJardins and her husband John Park. We have seen each other through so many adventures and there will be more to come now that we are both relocating to Maryland.

Of course, I also want to thank my good friends who put up with me through this process: Ellen Anderson, Ellen Hatheway, Carolyn Helmke, Kathy Howard, Sandy Irani, Alane King, Mary Ann Leung, Cecilia Mills, Melicent Peck, and Robin Pugh. I'd also like to thank Hector Gonzalez-Banos for his support through the past year. And I'd like to thank Anne Westbrook for her enthusiastic support.

Finally, I'd like to thank my mom and dad, for their unwavering support throughout *everything*.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many large collections of structured information are stored in a relational format. Most often they are stored in a relational database, but other relational representations include object-oriented databases and semi-structured representations such as XML. These large collections of structured data are common: they occur in diverse areas including retail sales, telecommunications, insurance, medicine and scientific domains.

Recently there has been a growing interest in "mining" this data. In fact, this is the impetus for the newly emerging discipline of data mining. The goal in data mining is to discover unsuspected relationships and to summarize data in ways that are both understandable and useful to the owner of the data [Hand et al., 2001]. One way in which this goal can be accomplished is by building a statistical model describing the data.

The construction of statistical models is a task that has been well studied in both statistics and machine learning communities. There is a huge body of work on building both predictive models (models that are optimized for accurate prediction of a subset of the features) and descriptive models (models that describe the data or describe how the data is generated). Our focus here will be almost entirely on descriptive modeling, in particular density estimation. Many algorithms for density estimations exist. They vary in the types of models learned. Some algorithms construct parametric models, in that they assume a particular form for the distribution, and the task

is parameter estimation. Examples include estimating the parameters of Gaussian densities, Poisson distributions or multinomial distributions. Other algorithms, for example kernel estimators, use non-parametric models and no assumptions are made about the functional form. A third category of models are mixtures of parametric models; most commonly these are mixtures of Gaussians but any parametric model may be chosen.

However, few of these density estimation algorithms are capable of handling data in its relational form. The input to the algorithm is assumed to be in the form of a collection of instances all of which have the same set of attributes. Depending on the problem, this input may be considered a random sample of the population, or it may be the entire population. However, in both cases, the assumption is made that the structure of the items in the input is identical. If the input is coming from a relational database, then we see that the the algorithms essentially only work on relational databases with a single table. However, the majority of relational databases have more than one table!

This thesis describes our approach to learning statistical models from relational data. Our goal is to build statistical models that are able to capture the important inter-table correlations in the data and exploit the information available in relational structure of the data. Our hope is that these statistical models will more accurately capture the dependencies and correlations in the domain than previous approaches and prove useful for both data exploration and summarization in relational domains. Our contributions include a collection of statistical models applicable in relational settings and automated induction algorithms for the models. In addition to the theoretical descriptions of the models and learning algorithms, we demonstrate their application on real-world databases and provide some evidence that our hope has been met.

1.1 Motivation

Consider a simple relational database that describes customer transactions. The database may include a number of tables. Suppose we have a customer table containing customer information including demographic data such as income or education, an item table describing the items that may be purchased, and a table describing purchases, linking customers to the items that they have purchased. From this data, we may be interested in figuring out which items customers tend to buy together (basket analysis or affinity analysis), or the categories of customers that make certain types of purchases (customer profiling). The data is clearly relational: there is a many-many relationship between customers and the items they purchase.

Unfortunately, few inductive learning algorithms are capable of handling data in its relational form. Most are restricted to dealing with a flat set of instances, with a homogeneous set of attributes. To use these methods, one typically "flattens" the relational data, removing its richer structure, treating it as a collection of fixed-length vectors of attribute-value pairs stored in a single table. In our case, the data may be flattened into a table that records for each purchase, the characteristics of the customer and the characteristics of the item purchased. Alternatively, we may have a single row for each customer with an attribute for each potential purchase. A third alternative is to have a row for each item, and an attribute for each potential consumer. These last two approaches have the disadvantage that we must fix either the number of customers or the number of items in advance.

This flattening has several important adverse implications. From a database perspective, we will either need to materialize the flattened view, storing it as a single table, or perform the required join operations each time we query the database for statistics. Each of these has disadvantages. Converting the data to a single table is undesirable because it means we cannot mine our database directly. It introduces redundancy and potentially high consistency-maintenance overhead. Databases are typically organized to avoid redundancy; in order to save space and reduce maintenance overhead, tables are normalized to remove repeated information. Converting a set of tables into a single table can introduce duplication and we lose our compact

representation. Alternatively, executing the join each time we need to gather statistics from the database can be expensive and time consuming. In addition, we may need to build auxiliary indexes in order to perform the required joins repeatedly.

More importantly, however, this flattening process loses information which might be crucial in understanding the data. From a statistical perspective, storing the data in a single table can corrupt the integrity of our results. First there is the danger of introducing statistical skew when we flatten the data. In our transaction data, if we flatten the data into a vector of people and item attributes, then people who make a lot of purchases will be over-represented in this data. If we try to infer characteristics of people by computing statistics from this flattened data, we will get incorrect results.

Another form of information loss occurs if we lose the links between related objects; we may be able to infer important properties of an object based on the objects to which it is linked. For example, if two people live in the same household, they may make similar purchases. Obviously we cannot make these inferences if the links are not even modeled. A third form of information loss occurs when we repeatedly copy related objects without maintaining their shared identity. Suppose two purchases are made by the same person. If we simply make two copies of the person, we are making a false independence assumption. If we make inferences about unobserved attributes of the person, we have lost the requirement that these inferred attributes must be equal for each copy of the person. All of these drawbacks severely limit the ability of current statistical methods to mine relational databases.

1.2 Our Approach

In this thesis we provide the tools for constructing statistical models from relational data. Our goal is to learn structured probabilistic models, that represent statistical correlations both between the properties of an entity and between the properties of related entities. These statistical models can then be used for a variety of tasks including knowledge discovery, exploratory data analysis, data summarization and anomaly detection.

The starting point for our work is the structured representation of probabilistic

models, as exemplified in Bayesian networks (BNs) [Pearl, 1988]. A BN allows us to provide a compact representation of a complex probability distribution over some fixed set of *attributes* or *random variables*. The representation exploits the locality of influence that is present in many domains.

Building on this foundation, we introduce two statistical models for relational data: *Probabilistic Relational Models* and *Statistical Relational Models*. For each of these representations, we give the semantics and learning algorithms and demonstrate their applicability in a number real-world domains.

The *Probabilistic Relational Model* (PRM) knowledge representation framework is based on the work of Koller and Pfeffer [1998]. The model allows us to describe a template for a probability distribution. This, together with a set of domain objects, defines a distribution over the attributes of the objects. Such a model can then be used for reasoning about an entity using the entire rich structure of knowledge encoded by the relational representation. We can make inferences about an individual based on its particular relational context; these predictions will be different for a similar individual that is in a significantly different relational context.

Here we make several extensions to the basic PRM semantics provided by Koller and Pfeffer [1998] and Pfeffer [2000]. One extension allows us to represent new forms of link uncertainty, for example uncertainty about whether a customer purchased an item. We extend earlier definitions of structural uncertainty and give semantics for two forms of link uncertainty. These allow us to probabilistically model alternative relational contexts, extending the applicability of PRMs to domains in which the relational structure is not known a priori. In fact, this extension can improve the inductive performance of the model. In particular, making the relational structure part of the probabilistic model can improve predictive accuracy over models which base their prediction solely on the object's attributes and do not consider the relational context. For example, by learning the probability that certain categories of customers make certain types of purchases, we may be able to detect anomalies in our data (a customer of that category who did not make the predicted purchases) or improve our accuracy predicting attributes (predicting the customer category based on the purchases made by the customer).

We also extend the PRM semantics by providing further support for the use of class hierarchies. This extension allows us to use specialization and inheritance to learn more richly structured probabilistic models. For example we may learn that, while a single model for the distribution of customer education is sufficient to capture the patterns in the data, a specialized model for customer income is required. The income distribution depends may depend on whether the consumer lives in a rural or urban area. Thus we can specialize the class of customers into rural customer and urban customers. More interestingly, by allowing this specialization, we can allow the dependencies within the subclasses. For example, we can say that a rural person's purchase of a tractor accessories depends on the type of tractor they have purchased; a John Deer tractor buyer is less likely to buy Honda tractor parts. On the other hand, the city dweller may not know enough to match tractor parts, but perhaps their purchase of software depends on what hardware they have purchased; a Macintosh buyer is less likely to buy PC software. In addition, the class hierarchy mechanism allows us to model both class level dependencies and instance level dependencies.

We also introduce *statistical relational models* (SRMs), a new category of structured statistical models. While on the surface PRMs and SRMs share many similarities, an SRM has significantly different semantics from a PRM. Halpern [1990] identified two distinct semantics for first-order logics of probability, which we will call the *possible-worlds* approach and the *domain-frequency* approach. We will see that PRMs are an instance of the possible-worlds approach, while an SRM is an instance of the domain-frequency approach to first-order probabilistic logics.

In the possible-worlds approach, we put a distribution over possible worlds. The distribution can be used to answer degree of belief questions such as 'What is the probability that a particular person, Joe Smith, has purchased a particular tractor part, Widget Supreme?'. In this case, the semantics of the formula is defined by summing up the probability over all of the possible worlds in which Joe Smith has bought Widget Supremes. However, as many have noted [Bacchus, 1990, 1988], there are difficulties with making statistical assertions, such as '75% of the purchases where for widgets'.

The second approach, the domain-frequency approach, is appropriate for giving

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

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

The core contribution of this thesis is a suite of methods for automatically constructing structured statistical models (PRMs and SRMs) directly from relational data. Typically, statistical models are constructed by first choosing a space of candidate models, next providing a scoring function that measures the quality of a fitted model and then providing a search mechanism for searching over the possible models. We show that many of the techniques of Bayesian network learning can be extended to the task of learning these more complex models. We examine the problems of *parameter estimation* and *structure selection* for this class of models. We deal with some crucial technical issues that distinguish the problem of learning structured relational models from that of learning Bayesian networks. We provide a formulation of the likelihood function appropriate to this setting, and show how it interacts with the standard assumptions of BN learning.

As in BN learning, the dependencies are required to be acyclic. The search over coherent dependency structures is significantly more complex than in the case of learning BN structure and we introduce the necessary tools and concepts to do this effectively. Structure selection is made based on the score of the model and we introduce two scoring functions: one that uses the posterior probability of the model

(a Bayesian scoring function) and another that uses the posterior probability of the data (a maximum likelihood scoring function). The first is appropriate in the case where we are learning a model which we would like to use to generalize to other situations (the most common goal in statistics). The latter is appropriate in the case where we are constructing a compact model of a particular dataset, and we wish only to capture the current data as precisely as possible, given a limited amount of space.

In addition to providing the basic learning framework for PRMs and SRMs, we describe how the methods can be extended to handle structural uncertainty and make use of class hierarchies. Throughout, we provide experimental validation of our methods, including many examples of the application of our algorithms to real-world data.

1.3 Contributions

In this thesis, we describe

- The first algorithm for automatically inducing a Probabilistic Relational Model directly from a relational database. This includes:
  - A definition for the likelihood of a model;
  - A method for guaranteeing the coherence of a model;
  - Methods for estimating the parameters of a model from the database;
  - An algorithm for searching the space of candidate models.
- Extensions to the basic PRM semantics to handle:
  - Certain categories of potentially cyclic dependencies;
  - Uncertainty over link structure;
  - Class hierarchies.
- Statistical Relational Models, a new semantics for structured probabilistic relational models based on domain-frequencies.

  - A definition of the distribution induced by a database query;
  - The formal semantics relating an SRM to the distribution of values in the database;
  - Methods for automatically constructing an SRM from a database;
  - An evaluation of the use of SRMs to estimate the result size for select-join queries over multiple tables.

- Validation of the algorithms on several domains including:
  - A database describing Tuberculosis cases. The results have been submitted to a journal of epidemiology.
  - A database of scientific papers;
  - A database of web pages;
  - A database containing movies and peoples' rankings of movies.

1.4 Outline of Thesis

This thesis is organized as follows. Chapter 2 reviews the basic PRM semantics, including a brief review of Bayesian networks. Chapter 3 describes the algorithms for automatically constructing a PRM from an existing database, addressing both the task of choosing a model structure and estimating the parameters of the structure. It includes results on both synthetic and real-world data. Chapter 4 describes both the extension to the PRM semantics for handling link uncertainty and the necessary extensions to the learning algorithms for learning this category of models. Chapter 5 looks at ways that class-based models and instance-based models can be combined. This is done through the use of class-hierarchies. First we show how class hierarchies can expand the representational power of PRMs in interesting ways and then we show how they can either be used during learning or constructed as part of the learning algorithm. In Chapter 6 we turn to SRMs. We describe their semantics and present a learning algorithm, essentially the same as the learning algorithm for PRMs, but with a few important changes. We also examine in detail an application of SRMs to the task

of selectivity estimation of queries for a relational database. Chapter 7 describes some of the related work; in particular we describe similarities and differences between our work and work in Inductive Logic Programming (ILP) [Lavrac and Dzeroski, 1994, Muggleton, 1992], one of the only subfields of machine learning which focuses on learning from relational data.

Some of this material has appeared previously in workshop and conference papers and book chapters. The material in Chapter 3 first appeared in Friedman et al. [1999a]. It later appeared in Getoor et al. [2001a]. The material in Chapter 4 is based largely on Getoor et al. [2001b], which in turn developed from the work presented in Getoor et al. [2000b]. The material in Chapter 5 initially appeared in Getoor et al. [2000a]. Some of the material is Chapter 6 also appeared in Getoor et al. [2001c].

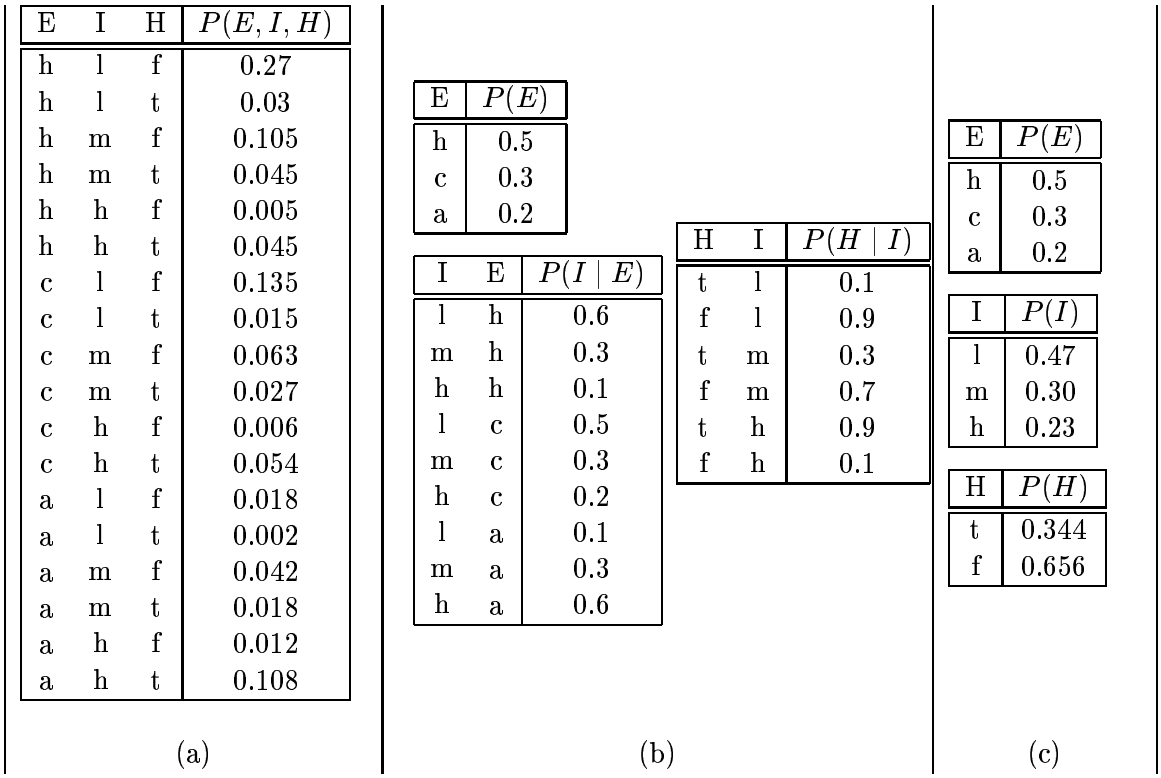
Chapter 2

Probabilistic Relational Models

In this chapter, we review the definition of probabilistic relational models. We introduce the basic relational model and the probabilistic model associated with it. The ideas are based on the recent work of Koller and Pfeffer [1998] and Pfeffer [2000], although we have generalized the terminology to be applicable to a wider range of relational systems. The probabilistic models that we study exploit conditional independence relationships that hold in many real-world distributions. We begin with a discussion of conditional independence. Then, since Bayesian networks provide the foundation for PRMs, we give a brief overview before introducing probabilistic relational models.

2.1 Conditional Independence

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



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

(a)

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

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

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

(b)

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

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

H	$P(H)$
t	0.344
f	0.656

(c)

Figure 2.1: (a) The joint probability distribution for a simple example. (b) A representation of the joint distribution that exploits conditional independence. (c) The single-attribute probability histograms.

Assume that the joint distribution of attribute values in a database is as shown in Figure 2.1(a). Using this joint distribution, we can compute the probability of any instantiation of E, I, and H, $P_{\mathcal{D}}(e, i, h)$. However, to explicitly represent the joint distribution we need to store 18 numbers, one for each possible combination of values for the attributes. (In fact, we can get away with 17 numbers because we know that the entries in the joint distribution must sum to 1.)

In many cases, however, our data will exhibit a certain structure that allows us to (perhaps approximately) represent the distribution using a much more compact form. The intuition is that some of the correlations between attributes might be indirect ones, mediated by other attributes. For example, the effect of education on owning a home might be mediated by income: a high-school dropout who owns a successful Internet startup is more likely to own a home than a highly educated beach bum —

the income is the dominant factor, not the education. This assertion is formalized by the statement that *Home-owner* is *conditionally independent* of *Education* given *Income*, i.e., for every combinations of values h, e, i, we have that:

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

This assumption holds for the distribution of Figure 2.1.

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

$$P(H, E, I) = P(H \mid I, E)P(I \mid E)P(E) = P(H \mid I)P(I \mid E)P(E)$$

where the last equality follows from the conditional independence of E and H given I. In our example, the joint distribution can be represented using the three tables shown in Figure 2.1(b). It is easy to verify that they do encode precisely the same joint distribution as in Figure 2.1(a).

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

Note that the conditional independence assumption is very different from assuming complete attribute independence. For example, the marginal distributions for the three attributes are shown in Figure 2.1(c). It is easy to see that the joint distribution that we would obtain from this strong attribute independence assumption in

this case is very different from the true underlying joint distribution. It is also important to note that our conditional independence assumption is compatible with the strong correlation that exists between *Home-owner* and *Education* in this distribution. Thus, conditional independence is a much weaker and more flexible assumption than standard (marginal) independence.

2.2 Bayesian Networks

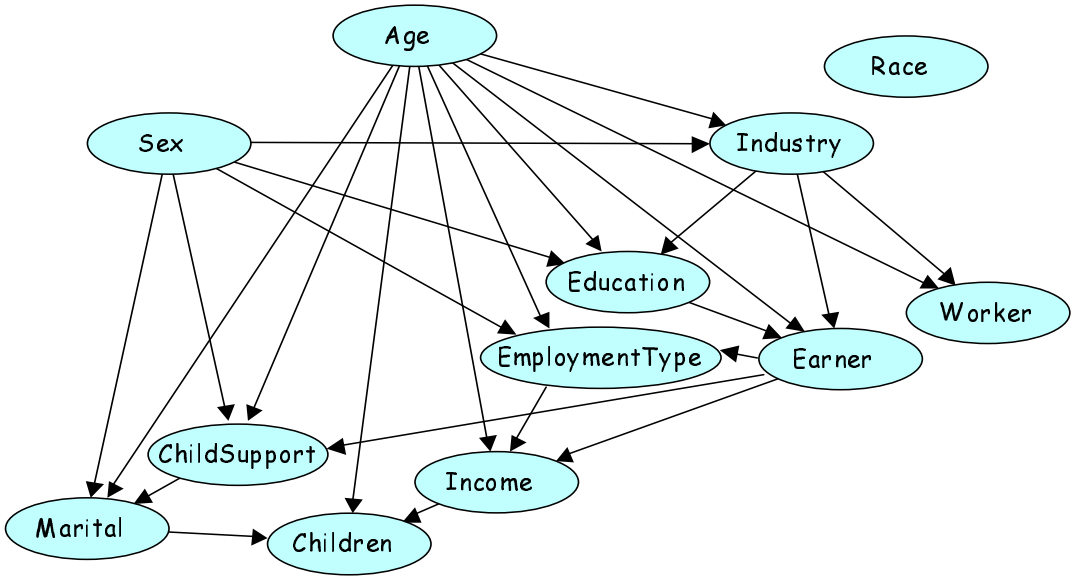


Figure 2.2: Bayesian network for the census domain.

Bayesian networks [Pearl, 1988] are compact graphical representations for high-dimensional joint distributions. They exploit the the underlying conditional independences in the domain — the fact that only a few aspects of the domain affect each other directly. We define our probability space as the set of possible assignments to the set a random variables $A_1, \ldots, A_n$. BNs can compactly represent a joint distribution over $A_1, \ldots, A_n$ by utilizing a structure that captures conditional independences among attributes, thereby taking advantage of the "locality" of probabilistic influences.

A Bayesian network $\mathcal{B}$ consists of two components. The first component, G, is

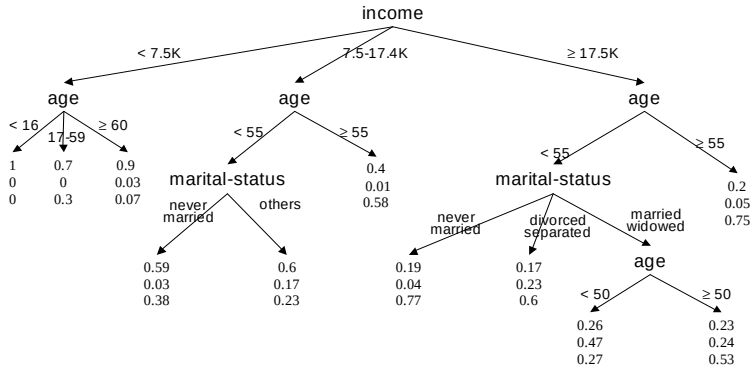


Figure 2.3: A tree-structured CPD for the *Children* node given its parents *Income*, *Age* and *Marital-Status*.

a directed acyclic graph whose nodes correspond to the attributes $A_1, \ldots, A_n$. The edges in the graph denote a direct dependence of an attribute A_i on its parents $\mathrm{Pa}(A_i)$. The graphical structure encodes a set of conditional independence assumptions: each node A_i is conditionally independent of its non-descendants given its parents.

Figure 2.2 shows a Bayesian network constructed (automatically, using a Bayesian network learning algorithm) from data obtained from the 1993 Current Population Survey of U.S. Census Bureau using their Data Extraction System [U.S., Census Bureau, 1992-93]. In this case, the table contains 12 attributes: *Age*, *Worker-Class*, *Education*, *Marital-Status*, *Industry*, *Race*, *Sex*, *Child-Support*, *Earner*, *Children*, *Income*, and *Employment-Type*. We see, for example, that the *Children* attribute (representing whether or not there are children in the household) depends on other attributes only via the attributes *Income*, *Age*, and *Marital-Status*. Thus, *Children* is conditionally independent of all other attributes given *Income*, *Age*, and *Marital-Status*.

The second component of a BN describes the statistical relationship between each node and its parents. It consists of a *conditional probability distribution (CPD)* $P_{\mathcal{B}}(A_i \mid \mathrm{Pa}(A_i))$ for each attribute, which specifies the distribution over the values of A_i given any possible assignment of values to its parents. Let $\mathcal{V}(A_i)$ denote the space

of possible values for A_i and $\mathcal{V}(\mathrm{Pa}(A_i))$ denote the space of possible values for the parents of A_i. The CPD is *legal* if all of the conditional probabilities are positive, and if, for any particular instantiation of A_i, $a_i \in \mathcal{V}(A_i)$, the sum over all possible instantiations of A_i, for any particalar instantition, $\boldsymbol{u}$, of A_i's parents is 1, i.e.,

$$\sum_{a_i \in \mathcal{V}(A_i)} P_{\mathcal{B}}(A_i = a_i \mid \mathrm{Pa}(A_i) = \boldsymbol{u}) = 1.$$

This CPD may be represented in a number of ways. It may be represented as a table, as in our earlier example. Alternatively, it can be represented as a tree [Boutilier et al., 1996], where the interior vertices represent splits on the value of some parent of A_i, and the leaves contain distributions over the values of A_i. In this representation, we find the conditional distribution over A_i given a particular choice of values $A_{k_1} = a_1, \ldots, A_{k_\ell} = a_\ell$ for its parents by following the appropriate path in the tree down to a leaf: When we encounter a split on some variable A_{k_j}, we go down the branch corresponding to the value of a_j; we then use the distribution stored at that leaf. The CPD tree for the *Children* attribute in the network of Figure 2.2 is shown in Figure 2.3. The possible values for this attribute are *N/A*, *Yes* and *No*. We can see, for example, that the distribution over *Children* given *Income* $\geq 17.5K$, *Age* < 55, and *Marital-Status* = *never-married* is $(0.19, 0.04, 0.77)$; the distribution given *Income* $\geq 17.5K$, *Age* < 50, and *Marital-Status* = *married* is $(0.26, 0.47, 0.27)$, as is the distribution given *Income* $\geq 17.5K$, *Age* < 50, and *Marital-Status* = *widowed*: the two instantiations lead to the same induced path down the tree.

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

$$P_{\mathcal{B}}(A_1, \ldots, A_n) = \prod_{i=1}^{n} P_{\mathcal{B}}(A_i \mid \mathrm{Pa}(A_i)). \tag{2.1}$$

Thus, from our compact model, we can recover the joint distribution; we do not need to represent it explicitly. In our example above, the number of entries in the full joint

distribution is approximately 7 billion, while the number of parameters in our BN is 951—a significant reduction!

2.3 Probabilistic Relational Models

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 pre-specified 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) [Koller and Pfeffer, 1998, Pfeffer, 2000] 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.

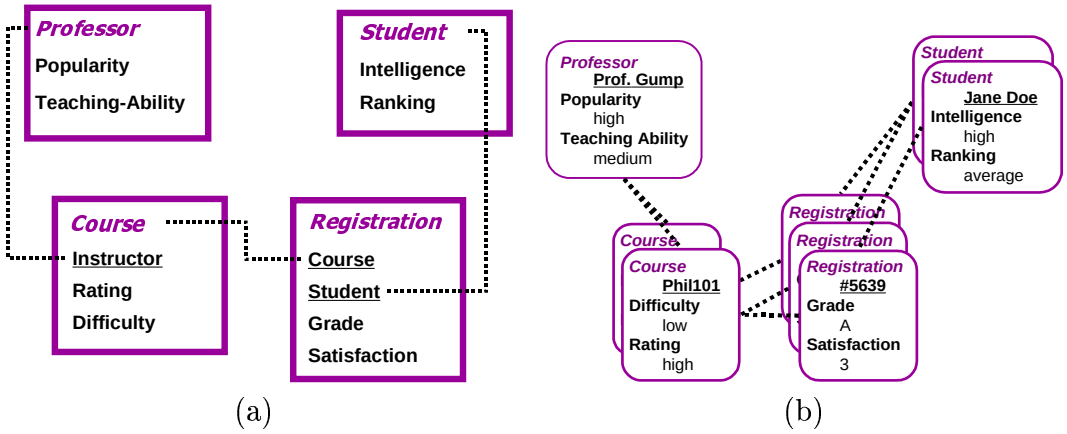


Figure 2.4: (a) A relational schema for a simple university domain. The underlined attributes are reference slots of the class and the dashed lines indicate the types of objects referenced. (b) An example instance of this schema. Here we do not show the reference slots, we use dashed lines to indicate the relationships that hold between objects.

2.3.1 Relational language

The relational language allows us to describe the kinds of objects in our domain. For example, Figure 2.4(a) shows the schema for a simple domain that we will be using as our running example in this chapter. 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 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. We achieve this effect 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 2.4(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, $\mathrm{Range}[\rho_i] = \mathrm{Dom}[\rho_{i+1}]$. 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.

[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.

2.3.2 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., $y \in \text{Range}[\rho]$. 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 2.4(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.

2.3.3 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 2.1: 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 2.5(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 each attribute of each object in the skeleton. A PRM then specifies a probability distribution over *completions* $\mathcal{I}$ of the skeleton.

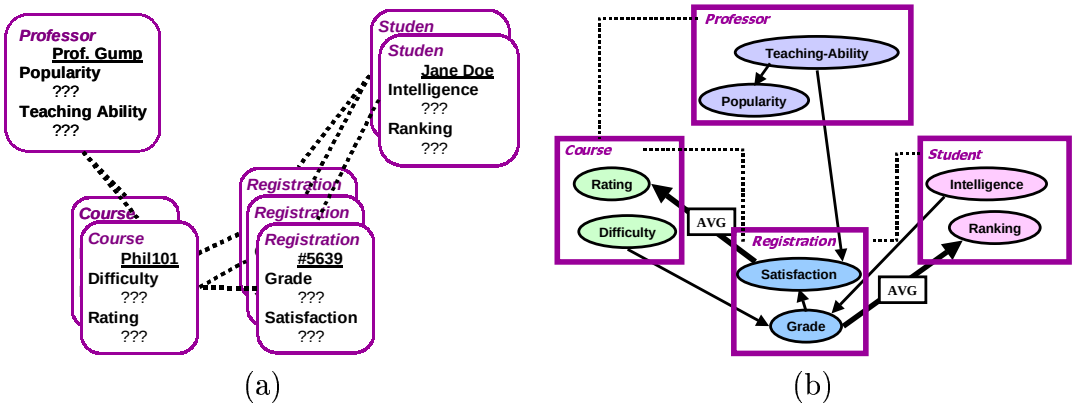


Figure 2.5: (a) The relational skeleton for the university domain. (b) The PRM dependency structure for our university example.

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 2.5(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 2.5(b), a professor's *Popularity* depends on her *Teaching-Ability*. The attribute $X.A$ can also depend on attributes of related objects $X.\tau.B$, where τ is a slot chain. In Figure 2.5(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 $x.\tau$ represents the *set* of objects that are τ-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 multi-set $\{y.B \ : \ y \in x.\tau\}$. For example,

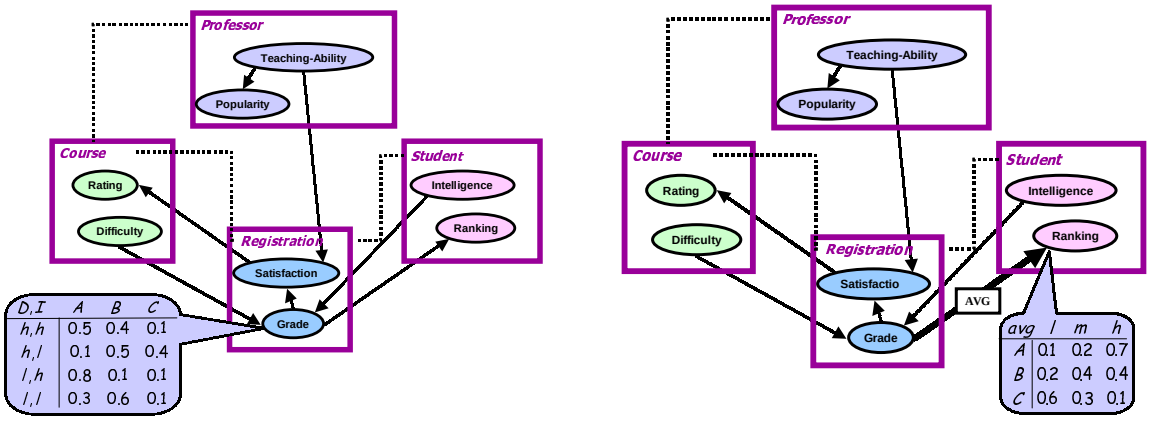
a student's rank depends on the grades in the courses in which they are registered. However each student may be enrolled in a different number of courses, and we will need a method of compactly representing these complex dependencies.

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 multi-set. 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); 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 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 multi-set of values of some ground type, and returns a summary of it. The type of the aggregate can be the same as that of its arguments. However, we allow other types as well, e.g., an aggregate that reports the size of the set. We allow $X.A$ to have as a parent $\gamma(X.\tau.B)$; the semantics is that for any $x \in X$, $x.A$ will depend on the value of $\gamma(x.\tau.B)$. 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 $\mathrm{Pa}(X.A)$ for $X.A$, we can define a local probability model for $X.A$. We associate $X.A$ with a CPD that specifies $P(X.A \mid \mathrm{Pa}(X.A))$. We require that the CPDs are legal. Figure 2.6 shows two CPDs. Let $\mathbf{U}$ be the set of parents of $X.A$, $\mathbf{U} = \mathrm{Pa}(X.A)$. Each of these parents U_i — whether a simple attribute in the same relation or an aggregate of a set of τ 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 2.2: 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) (b)

Figure 2.6: (a) The CPD for Registration.*Grade* (b) The CPD for an aggregate dependency of Student.*Ranking* on Student.*Registered-In.Grade*.

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

2.3.4 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 an *ground* Bayesian network over the random variables $x.A$.

Definition 2.3: 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 $x.\tau.B$. If τ is not single-valued, then the parent is the aggregate computed from the set of random variables $\{y \mid y \in x.\tau\}$, $\gamma(x.\tau.B)$.
- 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:

$$\begin{aligned} P(\mathcal{I} \mid \sigma_r, \mathcal{S}, \theta_{\mathcal{S}}) &= \prod_{x \in \sigma_r} \prod_{A \in \mathcal{A}(x)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \\ &= \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \sigma_r(X_i)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \end{aligned} \tag{2.2}$$

This expression is very similar to the chain rule for Bayesian networks Eq. (2.1). There are three primary differences. First, our random variables are the attributes of a set of objects. Second, the set of parents of a random variable can vary according to the relational context of the object — the set of objects to which it is related. Third, the parameters are shared; the parameters of the local probability models for attributes of objects in the same class are the identical.

2.3.5 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.

2.3.5.1 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 2.4: 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 $X.\tau.B$, and for each $y \in x.\tau$, 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 2.5: *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 Eq. (2.2).*

Proof: The probability of an instantiation $\mathcal{I}$ is the joint distribution over a set of random variables defined via the relational skeleton. We have one random variable $x.A$ for each $x \in \sigma_r(X)$ and each $A \in \mathcal{A}(X)$. Let $V_1, \ldots, V_N$ be the random variables defined above. Because the instance dependency graph is acyclic, we can construct a topological ordering $V_1, \ldots, V_N$ via the instance dependency graph; in other words, if $V_i = x.A$, then all ancestors of $x.A$ in the instance dependency graph appear before it in the ordering.

Our proof will use the following argument. Assume that we can construct a non-negative function $f(V_i, V_{i-1}, \ldots, V_1)$ which has the form of a conditional distribution $P(V_i \mid V_1, \ldots, V_{i-1})$, i.e., for any assignment $V_1, \ldots, V_{i-1}$ to $V_1, \ldots, V_{i-1}$, we have that

$$\sum_{V_i} f(V_i, V_{i-1}, \ldots, V_1) = 1. \tag{2.3}$$

Then, by the chain rule for probabilities, we can define

$$\begin{aligned} P(V_1, \ldots, V_N) &= \prod_{i=1}^{N} f(V_i, V_{i-1}, \ldots, V_1) \\ &= \prod_{i=1}^{N} P(V_i \mid V_1, \ldots, V_{i-1}). \end{aligned}$$

If f satisfies Eq. (2.3), then P is a well-defined joint distribution.

All that remains is to define the function f in a way that it satisfies Eq. (2.3). Specifically, the function f will be defined via Eq. (2.2).

Suppose that V_i is $x.A$, and consider any parent of $X.A$. There are two cases:

- If the parent is of the form $X.B$, then by the existence of type I edges, we have that $x.B$ precedes $x.A$ in G_{σ_r}. Hence, the variable $x.B$ precedes V_i.
- If the parent is of the form $X.\tau.B$, then by the existence of type II edges, for each $y \in x.\tau$ $y.B$ precedes $x.A$ in G_{σ_r}. Hence, each variable $y.B$ precedes V_i.

We can define

$$P(V_i \mid V_{i-1}, \ldots, V_1) = P(x.A \mid \mathrm{Pa}(x.A))$$

as in Eq. (2.2). As the right-hand-side is simply a CPD in the PRM, it specifies a well-defined conditional distribution, as required. And thus Eq. (2.2) is a well-defined joint distribution. ∎

∎

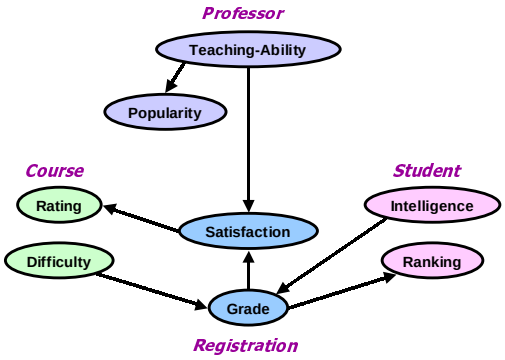


Figure 2.7: The class dependency graph for the school PRM.

2.3.5.2 Class Dependency Graph

The procedure 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 2.6: The *class dependency graph* G_Π for a PRM Π has a node for each descriptive attribute $X.A$, and the following edges:

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 $X.\tau.B$ we introduce an edge from $Y.B$ to $X.A$, where $Y = \text{Range}[X.\tau]$. ∎

Figure 2.7 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 2.7 shows, if the

class dependency graph is acyclic, we can never have that $x.A$ depends (directly or indirectly) on itself.

Theorem 2.7: *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.*

Proof: This is easy to show using proof by contradiction. Suppose there is a cycle $x_1.A_1 \rightarrow x_2.A_2 \cdots x_k.A_k \rightarrow x_1.A_1$. Then, because each of these object edges corresponds to an edge in the class dependency graph, we have the following cycle in the class graph:

$$X_1.A_1 \rightarrow X_2.A_2 \cdots X_k.A_k \rightarrow X_1.A_1$$

which contradicts our hypothesis that the class dependency graph is acyclic. ∎

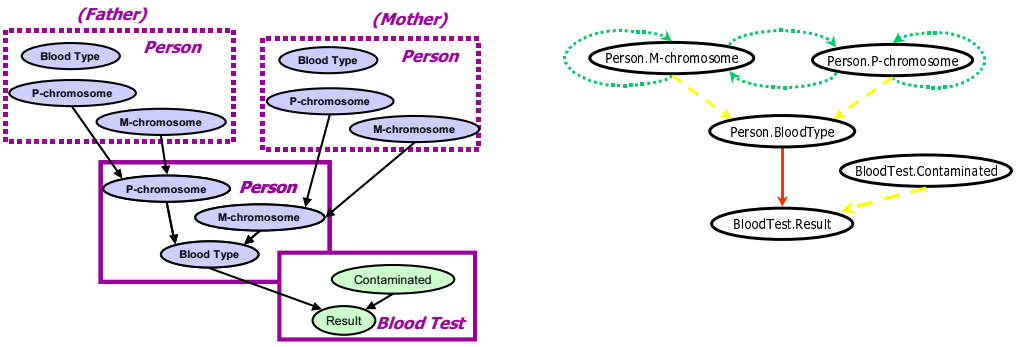
The following corollary follows immediately:

Corollary 2.8: *Let Π be a PRM whose class dependency structure $\mathcal{S}$ is acyclic. For for any relational skeleton σ_r, Π and σ_r define a coherent probability distribution over instantiations $\mathcal{I}$ that extend σ_r via Eq. (2.2).*

For example, if we examine the PRM of Figure 2.5(b), we can easily convince ourselves that we cannot create a cycle in any instance. Indeed, as we saw in Figure 2.7, 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.

Theorem 2.9: *If the class dependency graph G_Π of the PRM is cyclic then there exist relational skeletons for which the PRM would produce cyclic dependencies.*

Proof: It is straight forward to construct a relational skeleton that induces a cyclic instance dependency graph. The construction is as follows:



(a) (b)

Figure 2.8: (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.

- We have a single object x in each class X.
- For any slot $Y.\rho$ where $X = \text{Range}[\rho]$, we have $y.\rho = x$.

Now for any cycle in the class dependency graph induced by the PRM

$$X_1.A_1 \to X_2.A_2 \cdots X_k.A_k \to X_1.A_1,$$

the underlying instance dependency graph must have the following edges:

$$x_1.A_1 \to x_2.A_2 \cdots x_k.A_k \to x_1.A_1.$$

Thus, our PRM induces a cyclic dependency in for this simple skeleton. ∎

Thus our requirement that the class dependency graph is acyclic in order for the PRM to define a coherent distribution is, in a sense, complete.

2.3.5.3 Guaranteed Acyclic Relationships

In some cases, however, a cycle in the class dependency graph is not problematic. 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 2.8(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* 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 his parents; thus, at the class level, we have *Person.P-Chromosome* depending directly on *Person.P-Chromosome*. As we can see in Figure 2.8(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 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 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 2.8(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 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 τ 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 2.10: 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 $\gamma(X.\tau.B)$ is a parent of $X.A$, $Y = \text{Range}[X.\tau]$, and τ is guaranteed acyclic, we have a green edge $Y.B \rightarrow X.A$.
3. **Red edges:** If $\gamma(X.\tau.B)$ is a parent of $X.A$, $Y = \text{Range}[X.\tau]$, and τ 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 2.11: A (colored) dependency graph is *stratified* if every cycle in the graph contains at least one green edge and no red edges. ∎

Theorem 2.12: *If the colored class dependency graph is stratified for a PRM* Π*, then for* any *skeleton* σ_r*, the instance dependency graph is acyclic.*

Proof: Again we show this using proof by contradiction. Suppose there is a cycle $x_1.A_1 \rightarrow x_2.A_2 \cdots x_k.A_k \rightarrow x_1.A_1$ which contains at least one green edge and no red edges. In other words the cycle consists of some combination of green and yellow edges. From the cycle, we can reduce this to an ordering of the distinct objects

in the cycle, $x_{i_1} \ldots x_{i_m}$ by ignoring the yellow edges (since these are all within the same object). Consider the green edges: $x_{i_1}.A_1 \rightarrow x_{i_2}.B_1$, $x_{i_2}.A_2 \rightarrow x_{i_3}.B_2$, ..., $x_{i_m}.A_{i_m} \rightarrow x_{i_1}.B_{i_m}$. Each of these green edges uses some guaranteed acyclic slot chain τ_i. Because of the guaranteed acyclic ordering, $x_{i_j} \prec_{ga} x_{i_{j+1}}$. But, in addition, we have an edge from x_{i_m} to x_{i_1}. Thus we have both $x_{i_1} \prec_{ga} x_{i_m}$ and $x_{i_m} \prec_{ga} x_{i_1}$. This contradicts the assumption that we have a guaranteed acyclic ordering. Thus there can be no cycle of this type in the instance dependency graph. ∎

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.

2.4 The Difference between PRMs and BNs

The PRM specifies the 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 BN.

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 model, 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 BN is just a thought experiment; in many cases there is no need to actually construct this large underlying Bayesian network. For example, as we will see in the next chapter on learning, our construction algorithms all work at the class level, and exploit this compact representation.

2.5 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 a PRM supports many interesting patterns of reasoning. Often times we can view the inference as influence flowing between the inter-related 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 'A's, 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.

Although inference is an extremely important topic, in this thesis we do not describe new inference algorithms. Instead we make use of existing inference algorithms. In a few cases, we can do exact inference in the ground Bayes net. 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. In cases where the ground Bayesian network is very large and we cannot exploit regularities in its structure, we resort to approximate inference. Finally, for SRMs (e.g., in Chapter 6), we show how we can efficiently answer queries by doing inference in a much smaller network. We defer that discussion until Chapter 6. Here we briefly describe efficient exact inference and approximate inference briefly.

2.5.1 Exact Inference

We can always resort to exact inference on the ground BN, but the ground BN 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 [Koller and Pfeffer, 1997, Pfeffer et al., 1999, Pfeffer, 2000] 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. [1999] describes 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. [1999] to achieve impressive speedups; in a military battlespace domain the structured inference was orders of magnitudes faster than the standard BN exact inference algorithm.

2.5.2 Approximate Inference

Unfortunately the methods used in the inference algorithm above often are not applicable for PRMs we study. In the majority of cases, there are no generic objects that can be exploited. Unlike standard BN 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 our experiments in Chapter 4, 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 [1988]. The algorithm is guaranteed to converge to the correct marginal probabilities for each node only for singly connected Bayesian networks. However, empirical results [Murphy and Weiss, 1999] 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, referring to Weiss [2000], Murphy and Weiss [1999], MacKay et al. [1997] 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 BN,

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 belief propagation 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 BN. 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.

2.6 Conclusion

In this chapter, we have reviewed the definition of probabilistic relational models, originally introduced by Koller and Pfeffer [1998]. PRMs exploit both the compact representation of a probability distribution afforded by a Bayesian network with the expanded representational power provided by relational logic to define general yet compact representations for joint distributions over objects in structured domains. Here we have defined the major components of the model: the relational schema, the probabilistic dependency structure and parameterization, and the relational skeleton. We have shown the conditions under which these components together define a coherent probabilistic semantics for a domain. In later chapters we will describe extensions

to the basic model that will allow uncertainty over the link structure in the domain (Chapter 4), and will make use of class hierarchies (Chapter 5). However, first we turn to the important task of automatically constructing a PRM from an existing database.

Chapter 3

Learning Probabilistic Relational Models

In the previous chapter, we defined the PRM language and its semantics. We now move to the main contribution of this thesis: algorithms for learning a PRM from data. This chapter describes our first framework and algorithms which learn a PRM with attribute uncertainty. Following chapters examine extensions to the basic algorithms to handle additional forms of uncertainty.

3.1 Introduction

As we saw in the previous chapter, PRMs support complex reasoning patterns and can be used to make inferences about objects that are linked in many interesting ways. But we have not yet shown where a PRM comes from.

One option is to have a domain expert construct the model by hand. This can be a laborious and time-consuming process. It involves first determining the dependencies between the object attributes. In general this task is challenging, but for some well-understood domains the domain expert may be able to provide the model structure. The next challenge is specifying the parameters of the model. Even for experts, the elicitation of probabilities can be a very difficult task. Thus, this knowledge-engineering approach has limited applicability. It will only be successful in domains

in which there is an expert who thoroughly understands the domain.

But, as we stated at the outset of this thesis, we are particularly interested in domains where we have limited knowledge. We are interested in the task of knowledge discovery. So a method of constructing a PRM that does not rely on knowledge-engineering is needed. In this chapter, 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. Once we have learned a PRM, it can be used to make predictions and complex inferences in new situations.

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. Our training data consists of a fully specified instance of that schema. We assume that this 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 datasets.

There are two variants 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, there is no additional required input (although the user can, if available, provide prior knowledge about the structure, e.g., in the form of constraints). 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.

3.2 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:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \log P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} \log P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \right]. \end{aligned} \tag{3.1}$$

The key insight is that this equation is very similar to the log-likelihood of data given a Bayesian network [Heckerman, 1998]. In fact, it is the likelihood function of the Bayesian network induced by the 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 *tied*.

3.2.1 ML 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

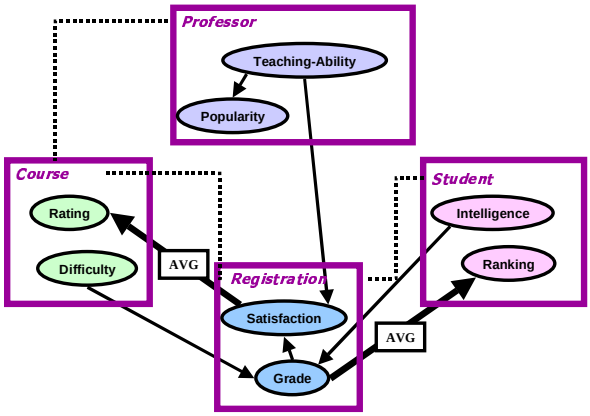


Figure 3.1: The dependency structure for the university PRM.

different classes:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} \log P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \right] \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \sum_{v \in \mathcal{V}(X.A)} \sum_{\mathbf{u} \in \mathcal{V}(\mathrm{Pa}X.A)} C_{X.A}[v, \mathbf{u}] \cdot \log \theta_{v|\mathbf{u}} \end{aligned} \tag{3.2}$$

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, maximal 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.

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

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

Proof: From Eq. (3.2) we see that we would like to maximize the multinomial distribution:

$$\sum_{v \in \mathcal{V}(X.A)} \sum_{\mathbf{u} \in \mathcal{V}(\mathrm{Pa}Class.A)} C_{X.A}[v, \mathbf{u}] \cdot \log \theta_{v|\mathbf{u}}$$

It is straightforward to set the derivative of this function to zero and solve for the parameters. It is well-known that the values of the parameters that maximize this function are simply their frequencies in the data. ∎

As a consequence of this proposition, parameter learning in PRMs is reduced to *counting*; the counts are the sufficient statistics. We need to count one vector of sufficient statistics for each CPD.

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

Consider our university example shown again in Figure 3.1. To compute the maximum likelihood parameters for Registration.*Grade*, we need to have counts for Registration.*Grade*, Registration.*Student.Intelligence* and Registration.*Course.Difficulty*.

3.2.2 Computing sufficient statistics from a DB

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. 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 construct a view that can be used to compute the sufficient statistics. This is useful 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.

3.2.3 Bayesian Parameter Estimation

In many cases, maximum likelihood parameter estimation is not robust, as it overfits the training data. The Bayesian approach uses a prior distribution over the parameters to smooth the irregularities in the training data, and is therefore significantly more robust. As we will see in Section 3.3.2, 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 distribution $\theta_{X.A|\mathbf{u}}$ 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 [Heckerman, 1998], 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 [DeGroot, 1970].) 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 3.2: *Assuming multinomial CPDs, prior independence, and Dirichlet priors, with hyperparameters* $\alpha_{X.A}[v, \mathbf{u}]$*, we have that:*

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

This suggests that the Bayesian estimate for $\theta_{\mathcal{S}}$ should be estimated using this formula as well. Unfortunately, the expected parameters 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 dataset 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 3.2. We use this approximation in our computation of the estimates for the parameters.

3.3 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.

3.3.1 Legal structures

We saw in Section 2.3.5.2 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.

3.3.1.1 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 $O(|V| + |E|)$.

3.3.1.2 Maintaining a Stratified Colored Class Dependency Graph

In the case where we have a colored class dependency graph, the algorithm is only slightly more complicated. Essentially we must consider all paths from (v, u) and ensure that, together with the new edge (u, v), any cycle is legal. Recall from Section 2.3.5.3 a cycle in a colored path dependency graph is legal if it contains at least one green edge and no red edges.

We will use a modification of Dijkstra's single-source shortest paths algorithm to determine the *unsafe* paths from v to any other node in G. There are two categories of unsafe paths possible. A path is unsafe if it contains no green edges or if contains a red edge.

In order to check for paths with no green edges, we can simply ignore the green edges in the graph, and apply our earlier DFS algorithm to see if there is a path from v to u.

A similar algorithm can be used to determine is there is a path containing a red edge from v to u. It is a bit more complicated, because we have to consider all possible paths. In this case we maintain CRP$[v_i]$ which is true if there is a path from v to v_i which contains a red edge. We perform a breadth first search of the graph, starting from node v. When we consider edge (w, x), we set CRP(x) to TRUE if there is either a path with a red edge from v to w or if the edge from (w, x) is red. The running time for this algorithm is also $O(|V| + |E|)$.

With these two functions we can determine whether adding an edge (u, w) to G results in an illegal colored dependency graph. An edge (u, v) introduces an illegal cycle if (u, v) is not green and we can find a path from v to u using only red and yellow edges, if CRP(u) is *true* or if (u, v) is red and there is a path from v to u.

3.3.1.3 Incremental Algorithms

The algorithms we have described do not save any information from one call to another, although clearly there may be some benefit in keeping track of the paths that exists in the graph from one call to the next. There are numerous incremental graph maintenance algorithms and analysis [Ausiello et al., 1991, Italiano, 1988]. These may well be useful here (as they would be in Bayesian network learning). But, in general, the time spent checking acyclicity is quite small compared with the time required to score the models. This is particularly true in the case of PRMs, where the number of nodes in the class dependency graph is relatively small. Thus, it is not clear that an incremental approach is needed.

3.3.2 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.[1] 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 τ 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 [Heckerman et al., 1995].

Proposition 3.3: *If $\mathcal{I}$ is a complete assignment, and $P(\theta_{\mathcal{S}} \mid \mathcal{S})$ satisfies parameter independence and is Dirichlet with hyperparameters $\alpha_{X.A}[v, \mathbf{u}]$, then the marginal*

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

likelihood of $\mathcal{I}$ *given* $\mathcal{S}$ *is:*

$$P(\mathcal{I} \mid \mathcal{S}, \sigma) = \prod_i \prod_{A \in \mathcal{A}(X_i)} \left[\prod_{\mathbf{u} \in \mathcal{V}(()Pa(X_i.A))} DM(\{C_{X_i.A}[v, \mathbf{u}]\}, \{\alpha_{X_i.A}[v, \mathbf{u}]\}) \right] \tag{3.3}$$

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

Hence, the marginal likelihood is a product of simple terms, each of which corresponds to a distribution $P(X.A \mid \mathbf{u})$ where $\mathbf{u} \in \mathcal{V}(\mathrm{Pa}(X.A))$. 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 MDL score (see, e.g., [Heckerman, 1998]).

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

3.3.3 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

```
Procedure Randomized Structure Search

Inputs:   random_step_prob — initial random step probability
          RANDOM_STEP_RATE — rate at which we descrease the probability
              of taking a random step
          MAX_STEPS — maximum number of search steps taken
          MAX_LOCAL_STEPS — maximum number of search steps
              for each different random_step_prob

num_steps = 0
while num_steps < MAX_STEPS
   num_local_steps = 0
   repeat
      if Random() < random_step_prob
         next_step = GetRandomSearchStep()
      else
         next_step = GetBestSearchStep()
      Apply next_step
      num_local_steps++
   until num_local_steps > MAX_LOCAL_STEPS or peak reached
   random_step_prob *= RANDOM_STEP_RATE
   num_steps += num_local_steps
end
return current model
```

Figure 3.2: The randomized structure search (**RSS**).

a procedure for finding legal high-scoring structures. For Bayesian networks, we know that this task is NP-hard [Chickering, 1996]. As PRM learning is at least as hard as 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 [Heckerman, 1998], 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 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 [Chickering et al., 1997], 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 tabu-list, which keeps track of the most recent states visited, and allow 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 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. One variant of such an algorithm is shown in Figure 3.2. This algorithm is still relatively naive and a number of improvements could be made such as having the random edge choice be a function of the score of the step. The algorithms we use for our experiments are either the simple greedy algorithm or the randomized algorithm **RSS**.

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

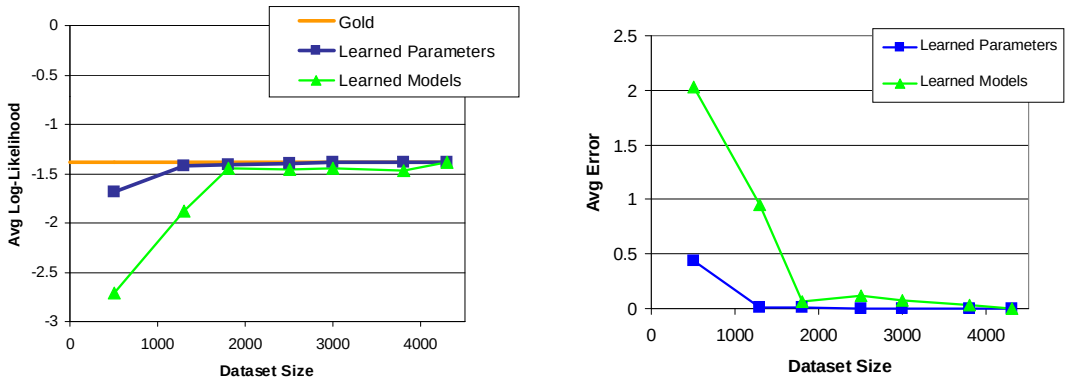
Eq. (3.3).) Now, if our search algorithm considers a modification to our current structure where the parent set of a single attribute $X.A$ is different, only the component of the score associated with $X.A$ will change. Thus, we need only reevaluate this particular component, leaving the others unchanged; this results in major computational savings.

However, there is 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 done very efficiently.

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

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

This scheme expands the set of potential parents at each iteration. It is the method



(a) (b)

Figure 3.3: (s) Learning curve showing the generalization performance of PRMs learned in the genetic domain. The x-axis shows the training set size; the y-axis shows the average log-likelihood on a test set of size 100,000. For each sample size, we learned models for ten different independent training sets of that size. The curve shows the general improvement in the average log-likelihood of the models as a function of the sample size. (b) Error (standard deviation) as a function of training set size.

that we have used in all of our results. 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. [1999b] 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 $\gamma(X.\tau.B)$ 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.

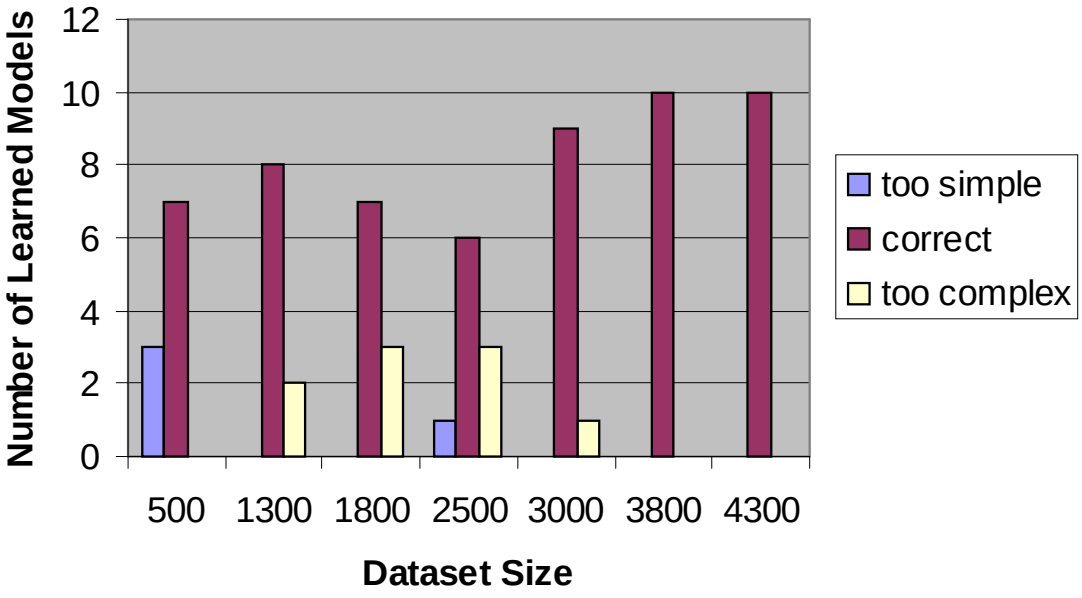


Figure 3.4: Results showing the percentage of models learned with the correct structure.

3.4 Experimental Evaluation on Synthetic Data

We begin by evaluating our algorithm on synthetic data. This allows us to evaluate the learned models against a 'gold' standard. We can then evaluate both the accuracy of our model and how well we recover the underlying structure.

We used the genetic domain described earlier in the chapter (Figure 2.8(a)) to generate the databases used in our experiments. Our sampling algorithm takes as input the size of the first generation, the distribution of genotypes in this generation and a birth rate and generates a family tree of size n. We generated various training databases, containing from 500 to 4300 individuals. For each size, we generated 10 different databases. For each training set, we test two different methods. In the first case, our algorithm has as input the correct structure and it simply estimates the parameters using the Bayesian parameter estimation methods described in Section 3.2.3. For our tests, we used a uniform Dirichlet parameter prior with equivalent

sample size $\alpha = 2$. In the second case, our algorithm learns both the structure and the parameters using the **RSS** algorithm described in the previous section. Again, we use an equivalent sample size of $\alpha = 2$ and a uniform structure prior.

We also generated an independent test database of size 100,000. We then tested how well the learned PRMs predict the test data. For measuring the predictive ability, we use the log-likelihood of the test data, the standard measure for evaluating density estimation procedures. Figure 3.3(a) shows the learning curve with results for the different training sets. The straight line at the top is the log-likelihood of the test data given the gold standard model, the 'true' model that was used to generate the data. Figure 3.3(b) shows the variance in log-likelihood (average error) for each of the methods as a function of training set size.

Even for the smallest training set size, the models learned have an error range that overlaps the average log-likelihood of the gold standard model, but the error ranges are still large. In the case where we are doing parameter estimation only, we quickly achieve a very accurate estimate—for training sets sizes as small as 1300 the model we learn is virtually indistinguishable from the true model. When we are learning structure as well, the required training set size is a little larger, but still for training sets larger than about 1800, we are able to learn models whose performance is hard to distinguish from the performance of the gold standard model.

Figure 3.4 shows the number of the learned models that have exactly the same structure as the gold standard model. An important caveat here is that models with a different structure from the gold standard model may have exactly the same score as the gold standard model. Many dependency structures are *score equivalent*, so that we cannot always distinguish between a model that has, for example, an edge going one direction versus a model that has the same edge directed in the opposite direction. However, in this case, there are no score equivalent models to the gold model, so we can make an exact structure comparison. We can also characterize the incorrect models as being either 'too simple', i.e., there are edges in the gold standard model that are missing in the learned model, or as being 'too complex', i.e., there are more edges in the learned model than required by the gold standard model.[2] In

[2]These overly complex models should be able to capture the underlying distribution: however,

this simple scenario, model structure could always be categorized as equivalent, 'too simple' or 'too complex'.

Our results show that as we increase the size of the training database, we generally learn more accurate structures. Examining the learned structures more closely, there is a qualitative difference in the incorrect models learned from small training sets versus large training sets. The incorrect models learned from small training sets have incorrect dependencies, such as a person's paternal genotype depending on a person's mother's paternal genotype, and missing dependencies. On the other hand, the incorrect models learned from the larger training sets capture the structure of the gold standard model, but are sometimes overly complex and include extraneous edges. While extra edges may indicate overfitting, this does not appear to be affecting us here as the scores of these models on test data are still quite high, and eventually we converge to the correct structure.

3.5 Application: Exploratory Data Analysis

We have also applied our PRM construction algorithm to various real-world domains. The goal in these domains is to gain a preliminary understanding of the the most important dependencies that hold in the domain. Once this exploratory phase is done, we may then want to focus on and analyze certain interesting correlations that we have discovered.

3.5.1 The TB Domain

One compelling domain that we have been fortunate to have the opportunity to work with is a database of epidemiological data for patients from the San Francisco tuberculosis (TB) clinic [Wilson et al., 1999].[3] The data collected describes TB patients and includes the strain of the disease with which they are inffected and information collected about people with whom the patients have been in contact.

in general a larger sample will be required to estimate the additional parameters accurately.

[3]We would like to thank Dr. Peter Small and Jeanne Rhee from the Stanford University Medical Center for providing us with the TB data and with their expertise on this topic.

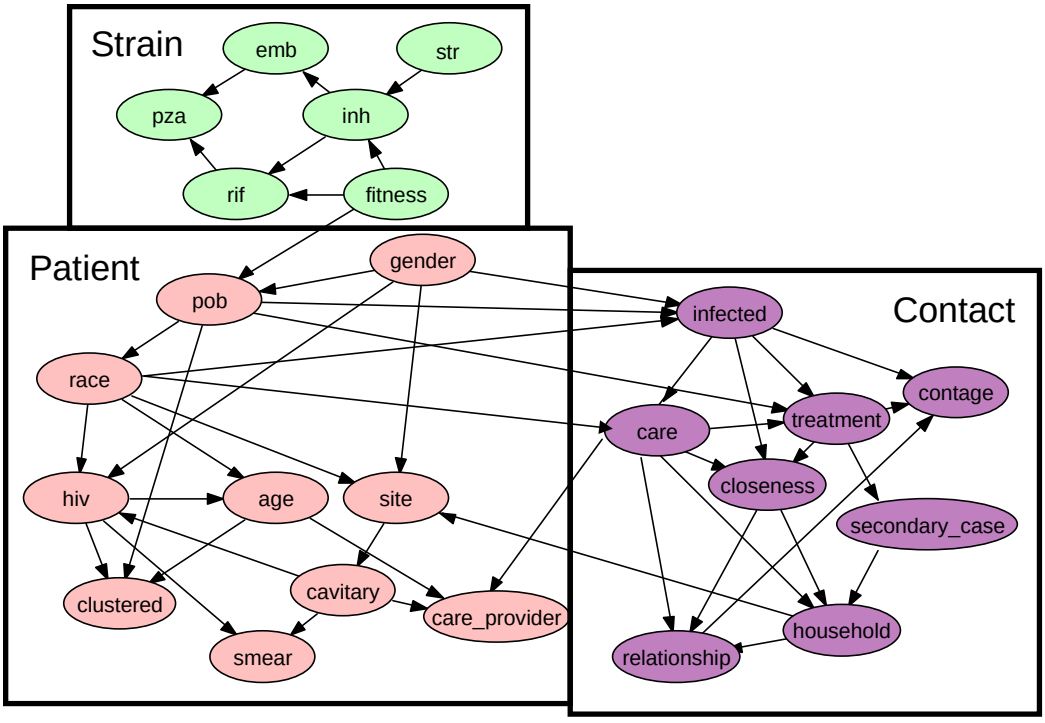


Figure 3.5: The PRM structure learned for the TB domain using tree-structured CPDs.

Of particular interest is whether a disease transmission chain between patients and contacts is indicated. The database contains three tables: Patient, Strain and Contact.

For the patient table, the schema contains demographic attributes such as *age*, *gender*, *race*, and place of birth (*pob*), as well as medical attributes such as HIV status, disease site (for TB), whether the X-ray results indicate cavitary infection, etc. The *clustered* attribute indicates whether this patient is a member of a cluster of TB outbreaks (a number of patients with the same TB strain) or whether the patient is infected with a unique strain of TB (often indicative of an independent reactivation of latent infection and not part of a disease transmission chain).

The strain table represents the different TB strains that have been identified among the patient population. The TB strains are identified using a genetic fingerprinting technique, in this case restriction fragment length polymorphism (RFLP), on

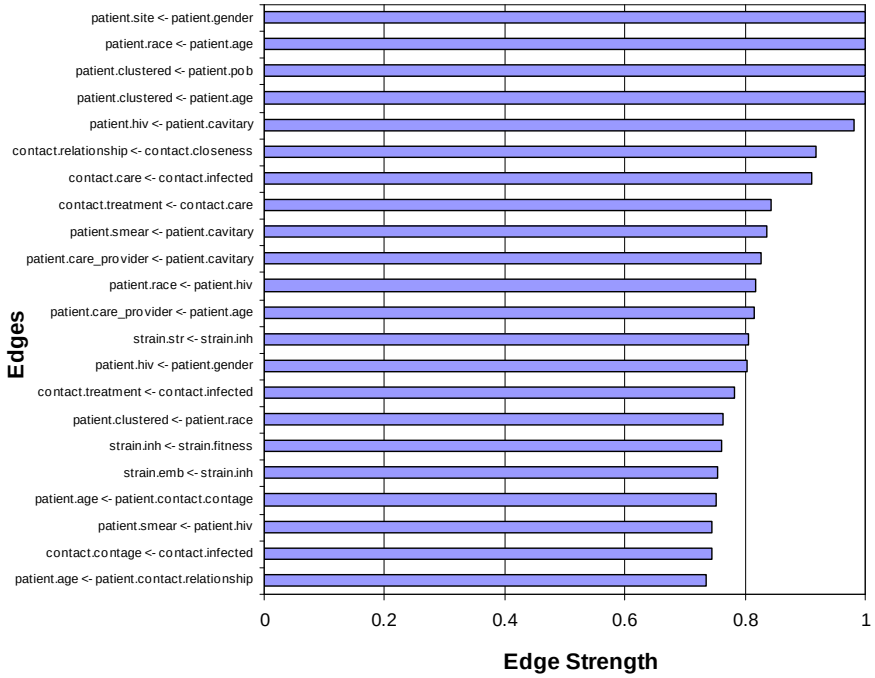


Figure 3.6: The edges with the highest strengths computed from 100 learned models. The strength is measured by weighing each edge occurrence by the score of the model it appears in.

bacterial isolates obtained from the patients. Epidemiologists assume that patients with isolates that have matching DNA fingerprint have been involved in disease transmission chains, while patients infected with a strain with a unique fingerprint have the disease due to a reactivation of a latent infection. The strain table has attributes for the isolate's drug susceptibility results for streptomycin (*str*), isoniazid (*inh*), rifampin (*rif*), ethambutol (*emb*), and pyrazinamide (*pza*). The *fitness* attribute reflects the relative prevalence of a particular strain of *M. tuberculosis* in the study population: strains identified only once were considered to have low fitness, strains isolated from 2–5 different patients were considered to have moderate fitness, and strains found in more than 5 individuals were considered to have high fitness.

Each patient is also asked for a list of people with whom he has been in contact; the contact table has attributes that specify the relationship of contact to the patient

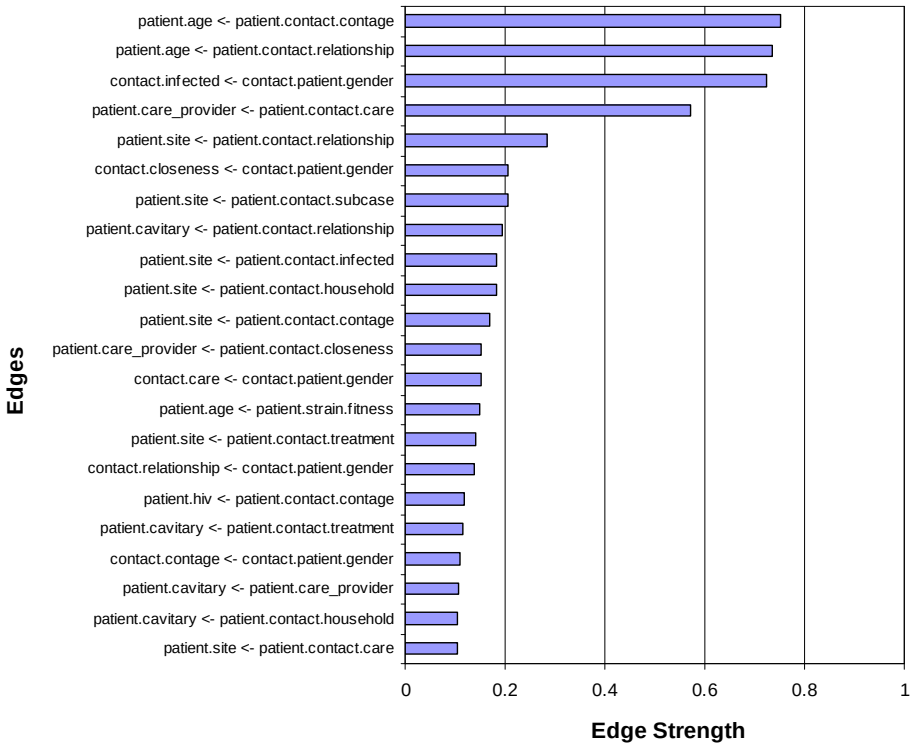


Figure 3.7: The strongest inter-relational edges in 100 learned models.

(sibling, coworker, etc.), contact age, whether the contact is a household member, etc.; in addition, the type of diagnostic procedure that the contact undergoes (*care*) and the result of the diagnosis (*infected*) are also reported. The *secondary_case* attribute indicates whether the contact later becomes a patient in the clinic.

The database contains information about 1879 patients. There are 1341 different strains and there is information on 15,336 patient contacts.

3.5.2 The Learned TB PRM

Figure 3.5 shows the PRM learned from this database. For all of the runs presented here, we learn a tree-structured CPD representation. We learn a rich dependency structure both within entities and between attributes in different entities. We reproduced many known dependencies, for example: the dependence of age at diagnosis on

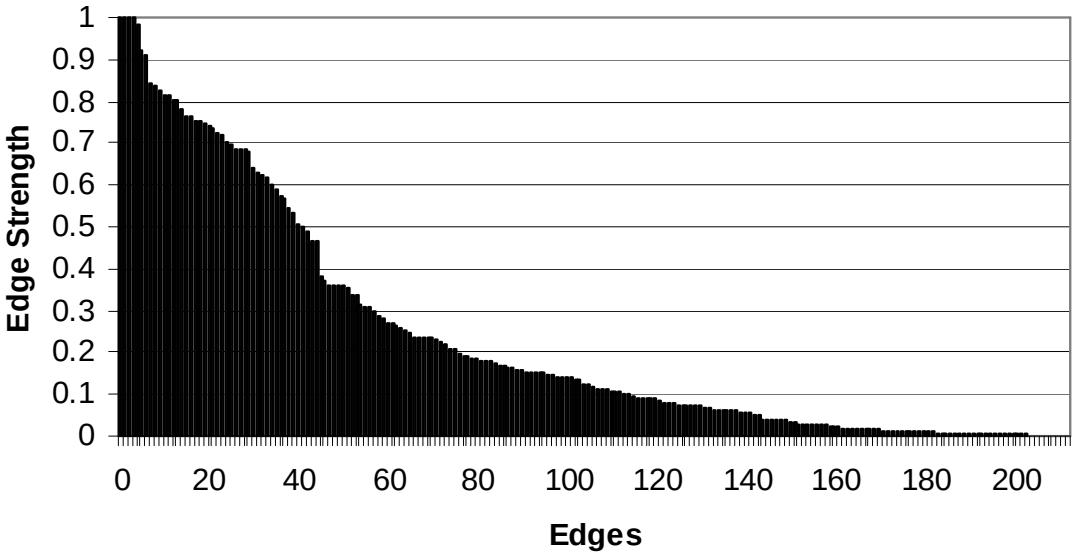


Figure 3.8: The distribution of edge strengths in 100 learned models (Tree CPDs).

HIV status — typically, HIV-positive patients are younger, and are infected with TB as a result of AIDS; the dependence of the contact's age on the relationship of contact with the patient — contacts who are coworkers are likely to be younger than contacts who are parents and older than those who are school friends; or the dependence of HIV status on race — Asian patients are rarely HIV positive whereas white patients are much more likely to be HIV positive.

We also discovered dependencies that are clearly relational, and that would have been difficult to detect using a non-relational learning algorithm. For example there is a correlation between the race of the patient and the fitness of the strain. Patients who are Asian are more likely to be infected with a strain which is unique in the population (low fitness), whereas other ethnicities are more likely to have strains that recur in several patients (high fitness). An explanation is that Asian patients are more often immigrants, who immigrate to the U.S. with a new strain of TB, whereas others are often infected locally. Another interesting relational correlation is between the place of birth of the patient and whether or not their contacts become secondary cases. US-born patients' contacts are more likely to become secondary cases than foreign

born patients. This phenomenon is also explained by the immigrant's likelihood of having a reactivated (non-infectious) TB case.

The model shown in Figure 3.5 shows the PRM learned using the greedy search algorithm, **Greedy**. In order to get a better understanding of the robustness of the learned dependencies, in the spirit of model averaging, we use **RSS** to learn a number of different models, and looked at the occurrences of edges in the collection of models. To compute the strength of an edge, we use the number of times it occurs in the models, weighted by the score of the model in which it occurs.

Figure 3.8 shows the distribution of occurrences of edges in 100 learned models. Of the possible 310 possible edges that can occur in our models, we see about two-thirds of them occurring at least once.[4] Figure 3.6 shows the edges with the highest strength. Looking at the edge strength supports our earlier discoveries. For example, the strength of an edge going from *hiv* to *age* is 62% and the strength of an edge going from *age* to *hiv* is 36%; together, these imply a strong coupling. The strength of an edge between contact relationship and contact age is 61.9% and there is a 38% strength of having the same edge in the reverse direction. There is a 82% strength of having an edge *hiv* to *race* and 16% strength of having the reversed edge.

Figure 3.7 shows the most probable edges between attributes in different relations. For the multi-relational dependencies, we see high confidence in edges between the ages of the patients and the contacts and between the contact relationship and the age of the patient. A more interesting highly probable edge is the edge between patient *gender* and contact *infected*. Female patients have a slightly higher probability of infecting their contacts (37% for women and 34% for men). The most common correlations between strain fitness and patient attributes are on the attributes *race*, *pob* and *age*, although each of these has strength less than 15%.

We note that multi-relational edges are generally less probable than intra-relation edges. However, this is not surprising. First, attributes of the same object or tuple are often more tightly coupled and thus more likely to have correlations. Second, by the nature of our phased search, we will uncover dependencies between attributes

[4] The models learned had tree structured CPDs; among other things this allows finer-grain dependencies to be learned. If we restrict attention to table CPDs, then in 100 runs only about a third of the possible edges occur.

within the same table before we consider dependencies between attributes in different tables: thus, we will only add the latter to our model in the case where the correlation has not already been explained.

There are several caveats to be made about these numbers. First, as mentioned earlier, many dependency structures are score equivalent, so that we cannot always distinguish between a model that has, for example, an edge directed one way from a model that has the same edge directed the opposite way. Second, in making inferences, the path of influence between two variables is what really matters, not just whether they are directly connected. Also, a more principled randomized search algorithm should be used. Nevertheless, these experiments give us some additional insight into the domain.

3.5.3 Inference in the learned networks

Once we have learned a PRM, we can use it to make inferences about other, newly encountered, patients and contacts.

For example, we may have a new patient, *Patient-101002* who has named 3 contacts, *Contact-1* and *Contact-2* and *Contact-3*. Figure 3.9 shows a portion of the ground Bayesian network for this scenario (The figure shows the Hugin Expert [2001] interface), along with the a priori probabilities for a number of the attributes.

Suppose we have some information about the patient's first two contacts. Both of these contacts were close and became infected with the disease. Using this information alone, the probability that the patient's third contact is infected goes from 35.34 to 40.29, as shown in Figure 3.10. If we also find out that the contact between the patient's and the contact has been close, then the probability that the third contact is infected increases to 52.53, as shown in Figure 3.11. While a priori, we would conclude that the contact is most likely not infected, by making use of this relational information, we would change our conclusion and conclude that the patient is likely to be infected.

Many more complex patterns of inference can be made, but each is in the context of a particular patient and their set of contacts.

3.6 Conclusion

In this chapter we have described the first framework for learning a statistical model from relational data. We have presented a method for the automatic construction of a PRM with attribute uncertainty 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. Most importantly, we have shown results on both synthetic data and on an interesting real-world problem, and shown how our algorithms can in fact be used to discover novel new patterns the data.

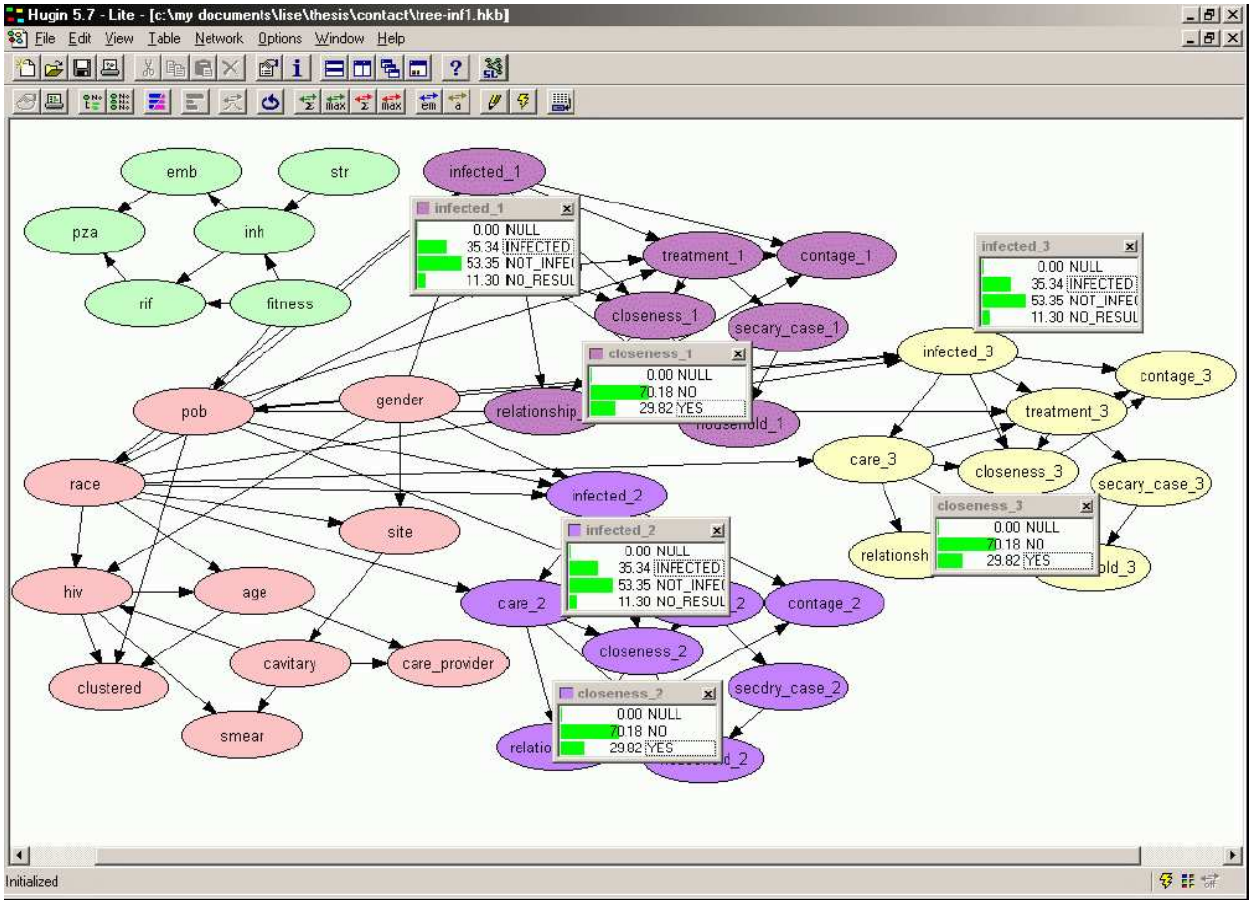


Figure 3.9: The ground Bayesian network for a patient and their three contacts. The figure shows the a priori probabilities of a number of attributes.

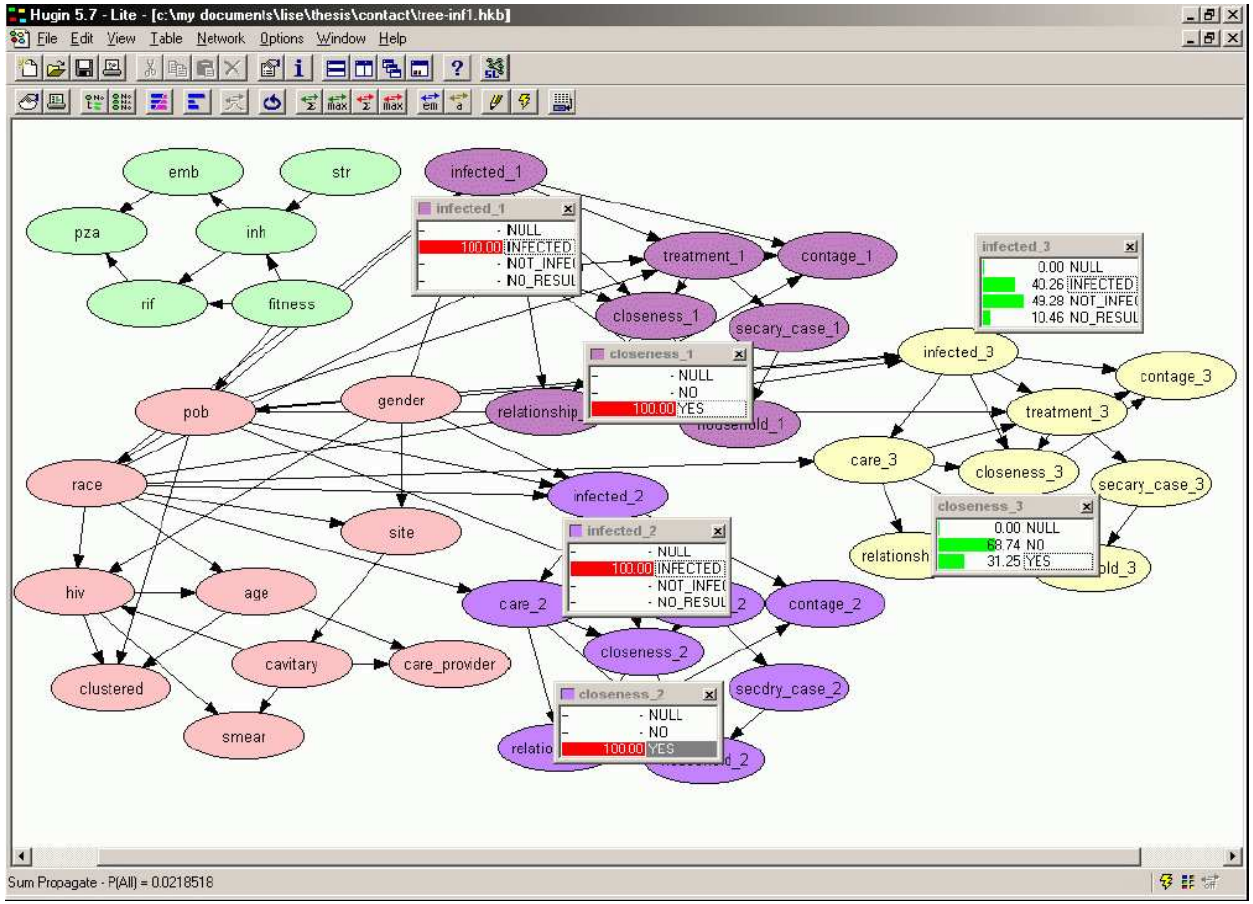


Figure 3.10: The updated probabilities after observing that two of the contacts were infected and close.

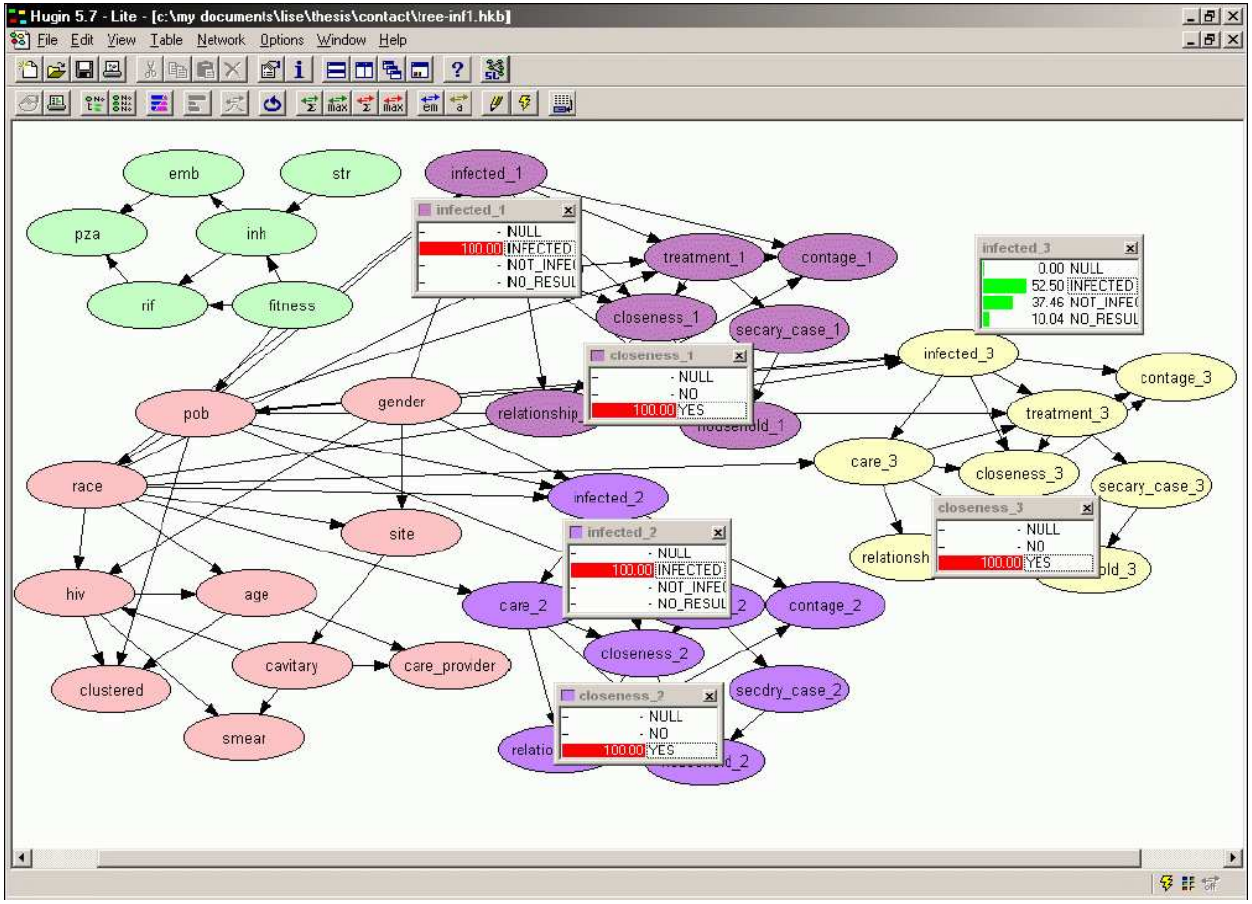


Figure 3.11: The probability of the third contact being infected, given the information about the first and second contact and the contact between the patient and the third contact is close.

Chapter 4

PRMs with Link Uncertainty

In this chapter, we propose probabilistic models not only for the attributes of the objects in a relational model, but also for the relational or link structure itself. In other words, we will model the probability that certain relationships hold between objects. We propose two mechanisms for modeling link uncertainty: *reference uncertainty* and *existence uncertainty*. We describe the appropriate conditions for using each model and learning algorithms for each model. We present experimental results showing that the learned models can be used to predict relational structure and, moreover, the observed relational structure can be used to provide better predictions for the attributes in the model.

4.1 Introduction

The PRM framework presented in Chapter 2 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 web page are very informative about the type of web page [Craven et al., 1998], and the citation links between papers are very informative

about the paper topics [Cohn and Hofmann, 2001].

As we show in this chapter, the PRM framework provides us with a very natural way of addressing this limitation. 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.

The concept of a probabilistic model over relational structure was introduced by Koller and Pfeffer [1998] under the name *structural uncertainty*. They defined several variants of structural uncertainty, and presented algorithms for doing probabilistic inference in models involving structural uncertainty. In this chapter, we show how a probabilistic model of relational structure can be learned directly from data. Specifically, we extend the *reference uncertainty* model of Koller and Pfeffer [1998] to make it suitable for a learning framework; we also introduce a new type of structural uncertainty, called *existence uncertainty*. We propose a method for learning such models from a relational database, and present empirical results on real-world data showing that these models can be used to predict relational structure, as well as use an observed relational structure to provide better predictions about attribute values.

4.2 Probabilistic Model of Link Structure

In the model described in the previous chapters, 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 Chapter 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 2.5(a)) contains all of the information except for the values for the descriptive attributes.

Thus, Eq. (2.2) determines the probabilistic model of the attributes of objects, but does not provide a model for the links between objects. In this chapter, we extend our probabilistic model to allow for *link uncertainty*. Here, we do not treat the relational structure as background knowledge, but choose to model it explicitly within the probabilistic framework. Clearly, there are many ways to represent a probability distribution over the relational structure. In this chapter, we explore two simple yet natural models: *Reference Uncertainty* and *Existence Uncertainty*.

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. Thus, a relationship such as Registration, which relates students to courses, is represented as a class, with reference slots to the class Student and the class Course. This approach, which blurs the distinction between entities and relationships, is common, and allows us to accommodate descriptive attributes that are associated with the relation, such as *Grade* and *Satisfaction*.

In this chapter we will find it useful to be able to make this distinction. We 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.

4.3 Reference Uncertainty

Consider a simple citation domain illustrated in Figure 4.1. 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

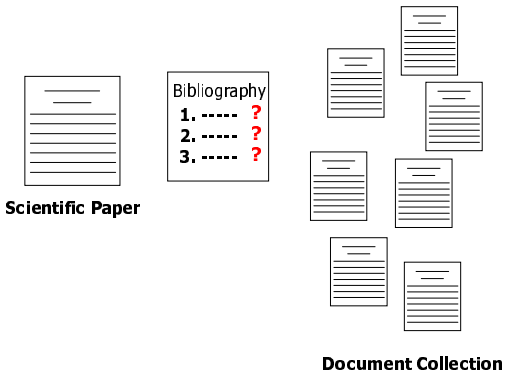


Figure 4.1: Reference uncertainty in a simple citation domain.

topics, we may be able to infer something about the topic of the citing paper.

Figure 4.2(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 pre-specified, 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 4.2 shows an object skeleton for the citation domain.

4.3.1 Probabilistic model

In the case of reference uncertainty, we must 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

(a) (b)

Figure 4.2: (a) A relational schema for the citation domain. (b) An object skeleton for the citation domain.

(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 4.3(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

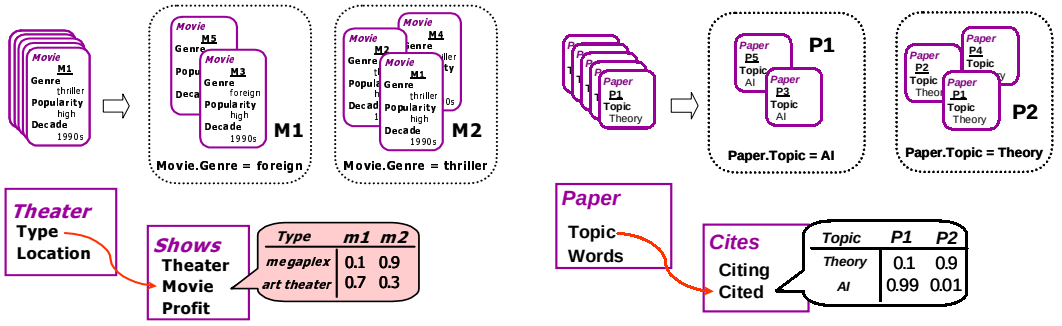


Figure 4.3: (a) An example of reference uncertainty for a movie theater's showings. (b) A simple example of reference uncertainty in the citation domain

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 in comparison to 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 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 4.3(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 4.1: Let $X.\rho$ be a reference slot with domain Y. Let $\Psi_\rho : Y \rightarrow \mathrm{Dom}[\Psi_\rho]$ 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, $\Psi_{\mathsf{Shows}.Movie} : \mathsf{Movie} \rightarrow \{Foreign, Thriller\}$ and the partition attributes ar $\mathcal{P}[\mathsf{Shows}.Movie] = \{Genre\}$. In the second example, $\Psi_{\mathsf{Cites}.Cited} : \mathsf{Paper} \rightarrow \{AI, Theory\}$ and the partition attributes are $\mathcal{P}[\mathsf{Cites}.Cited] = \{Topic\}$.

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 $\mathsf{Theater}.Type$. For each instantiation of the parents, we have a distribution over $\mathrm{Dom}[S_\rho]$.[1] The choice of value for S_ρ determines

[1] In the current work, we treat this distribution as a simple multinomial distribution over this value space. In general, however, we can represent such a distribution more compactly, e.g., using a Bayesian network. For example, the genre of movies shown by a movie theater might depend on its type (as above). However, the language of the movie can depend on the location of the theater. Thus, the partition will be defined by $\mathcal{P}(\mathsf{Show}.Movie) = \{\mathsf{Movie}.Genre, \mathsf{Movie}.Language\}$, and its parents would be $\mathsf{Theater}.Type$ and $\mathsf{Theater}.Location$. We can represent this conditional distribution more compactly by introducing a separate variable $S_{\mathsf{Movie}.Genre}$, with a parent $\mathsf{Theater}.Type$, and

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 4.2: A *probabilistic relational model* Π *with reference uncertainty* over a relational schema $\mathcal{R}$ has the same components as in Definition 2.2. In addition, for each reference slot $\rho \in \mathcal{R}(X)$ with $\text{Range}[\rho] = Y$, 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:

$$P(\mathcal{I} \mid \sigma_o, \Pi) = \prod_{X \in \mathcal{X}} \prod_{x \in \sigma_o(X)} \prod_{A \in \mathcal{A}(X)} P(x.A \mid \text{Pa}(x.A)) \prod_{\rho \in \mathcal{R}(X), y = x.\rho} \frac{P(x.S_\rho = \psi[y] \mid \text{Pa}(x.S_\rho))}{|\mathcal{I}(Y_{\psi[y]})|} \tag{4.1}$$

where $\psi[y]$ refers to $\Psi_\rho(y)$ — the partition that the partition function assigns y. Note that the last term in Eq. (4.1) 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 $Y_{\psi[y]}$.

There is one small problem with this definition: the probability is not well-defined if there are no objects in a partition, i.e., $|\mathcal{I}(Y_\psi)| = 0$. In this case, we will perform a renormalization of the distribution for the selector attribute, removing all probability mass from the empty partition. We then treat this term, $\frac{P(x.S_\rho = \psi[x.\rho] \mid \text{Pa}(x.S_\rho))}{|Y_\psi|}$, as 0 in the above product. In other words, an instantiation where $x.S_\rho = \psi$ and ψ is an empty partition necessarily has probability zero.[2]

another $S_{\text{Movie}.\textit{Language}}$, with a parent Theater.*Location*.

[2]While an important technical detail, in practice, for the majority of our experimental domains, we did not encounter empty partitions.

4.3.2 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 4.3: 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 $X.\tau.B$ where $\mathrm{Dom}[X.\tau] = Y$. 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 $X.\tau.B$, where $\tau = \rho_1, \ldots, \rho_k$, and $\mathrm{Dom}[\rho_i] = X_i$. We define an edge $x.\rho_1 \rightarrow x.A$, for every $x \in \sigma_o(X)$. In addition, for $i > 1$, we add an edge $x_i.\rho_i \rightarrow x.A$ for 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 $Y = \mathrm{Range}[\rho]$. 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 Eq. (4.1) specifies a coherent probability distribution.

Theorem 4.4: *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 Eq. (4.1).*

Proof: The probability of an instantiation $\mathcal{I}$ is the joint distribution over a set of random variables defined via the object skeleton. We have two types of variables:

- We have one random variable $x.A$ for each $x \in \sigma_o(X)$ and each $A \in \mathcal{A}(X)$. Note that this also includes variables of the form $x.S_\rho$ that correspond to selector variables.
- We have one random variable $x.\rho$ for each reference slot $\rho \in \mathcal{R}(X)$ and $x \in \sigma_o(X)$. This variable denotes the actual object in $\sigma_o(\text{Range}[\rho])$ that the slot points to.

Let $V_1, \ldots, V_N$ be the entire set of variables defined by the object skeleton. Clearly, there is a one-to-one mapping between joint assignments to these variables and instantiations $\mathcal{I}$ of the PRM that are consistent with σ_o. Because the instance dependency graph is acyclic, we can assume without loss of generality that the ordering $V_1, \ldots, V_N$ is a topological sort of the instance dependency graph; thus, if $V_i = x.A$, then all ancestors of V_i in the instance dependency graph appear before it in the ordering.

Our proof will use the following argument. Assume that we can construct a nonnegative function $f(V_i, V_{i-1}, \ldots, V_1)$ which has the form of a conditional distribution $P(V_i \mid V_1, \ldots, V_{i-1})$, i.e., for any assignment $V_1, \ldots, V_{i-1}$ to $V_1, \ldots, V_{i-1}$, we have that

$$\sum_{V_i} f(V_i, V_{i-1}, \ldots, V_1) = 1. \tag{4.2}$$

Then, by the chain rule for probabilities, we can define

$$\begin{aligned} P(V_1, \ldots, V_N) &= \prod_{i=1}^{N} f(V_i, V_{i-1}, \ldots, V_1) \\ &= \prod_{i=1}^{N} P(V_i \mid V_1, \ldots, V_{i-1}). \end{aligned}$$

If f satisfies Eq. (4.2), then P is a well-defined joint distribution.

All that remains is to define the function f in a way that it satisfies Eq. (4.2). Specifically, the function f will be defined via Eq. (4.1). We consider the two types of variables in our distribution.

Suppose that V_i is of the form $x.A$, and consider any parent of $X.A$. There are two cases:

- If the parent is of the form $X.B$, then by the existence of type I edges, we have that $x.B$ precedes $x.A$ in G_{σ_o}. Hence, the variable $x.B$ precedes V_i.
- If the parent is of the form $X.\tau.A$, then by the existence of type III edges, for each ρ_i along τ $x_i.\rho_i$ precedes $x.A$ in G_{σ_o} and hence, by the existence of type V edges all of the $x.S_{\rho_i}$ also precede $x.A$. Furthermore, by the existence of type II edges, for any $y \in x.\tau$, we have that $y.B$ precedes $x.A$.

In both cases, we can define

$$P(V_i \mid V_{i-1}, \ldots, V_1) = P(x.A \mid \mathrm{Pa}(x.A))$$

as in Eq. (4.1). As the right-hand-side is simply a CPD in the PRM, it specifies a well-defined conditional distribution, as required.

Now, suppose that V_i is of the form $x.\rho$. In this case, the conditional probability depends on $x.S_\rho$, and the value of the partition attributes of all objects in $\sigma_o(\mathrm{Range}[\rho])$. By the existence of type V edges, $x.S_\rho$ precedes $x.\rho$. Furthermore, by the existence of type IV edges, we have that $y.B$ for every $Y.B \in \mathcal{P}[\rho]$ and $y \in \sigma_o(\mathrm{Range}[\rho])$. Consequently, the assignment to $V_1, \ldots, V_{i-1}$ determines the number of objects in each partition of values of $\mathcal{P}[\rho]$ and hence the set $\mathcal{I}(Y_\psi)$ for every ψ.

Finally, we set

$$P(V_i = y \mid V_{i-1}, \ldots, V_1) = \begin{cases} \frac{1}{|\mathcal{I}(Y_{\psi[y]})|} & \text{if } \psi[y] = x.S_\rho \\ 0 & \text{otherwise} \end{cases}$$

which is a well defined distribution on the objects in $\sigma_o(Y)$. ∎

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.

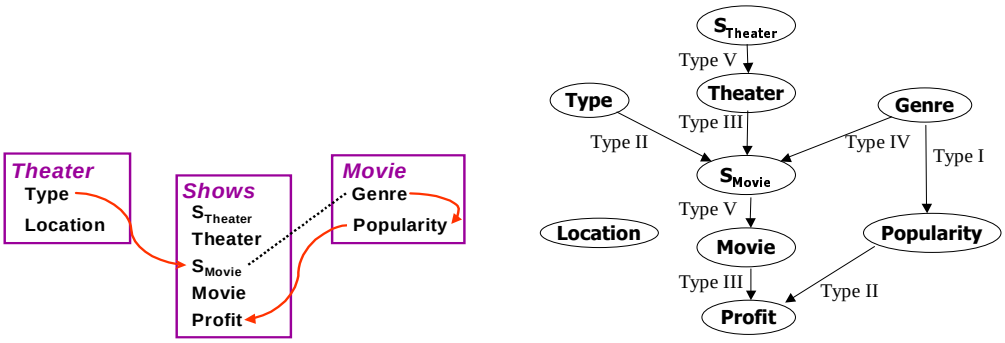


Figure 4.4: (a) A PRM for the movie theater example. The partition attributes are indicated using dashed lines. (b) The dependency graph for the movie theater example. The different edge types are labeled.

Definition 4.5: 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 $Y.\tau.B$, where $\tau = \rho_1, \ldots, \rho_k$, and $\text{Dom}[\rho_i] = X_i$, we define an edge $X.\rho_1 \rightarrow X.A$. In addition, 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 4.4 shows the class dependency graph for our extended movie example.

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

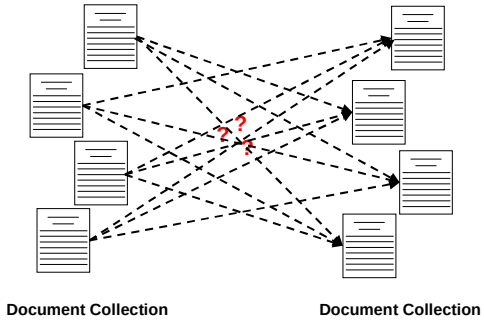


Figure 4.5: Existence uncertainty in a simple citation domain.

Lemma 4.6: *If the class dependency graph is acyclic for a PRM with reference uncertainty* Π*, then for* any *object skeleton* σ_o*, the instance dependency graph is acyclic.*

Proof: Assume by contradiction that there is a cycle

$$x_1.V_1 \rightarrow x_2.V_2 \cdots x_k.V_k \rightarrow x_1.V_1.$$

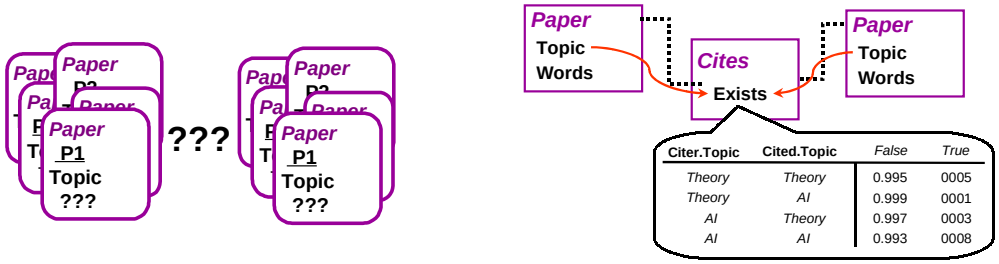
Then, because each of these object edges corresponds to an edge in the class dependency graph, we have the following cycle in the class dependency graph:

$$X_1.V_1 \rightarrow X_2.V_2 \cdots X_k.V_k \rightarrow X_1.V_1.$$

This contradicts our hypothesis that the class dependency graph is acyclic. ∎

The following corollary follows immediately:

Corollary 4.7: *Let* Π *be a PRM with reference uncertainty whose class dependency structure* $\mathcal{S}$ *is acyclic. For for any object skeleton* σ_o*,* Π *and* σ_o *define a coherent probability distribution over instantiations* $\mathcal{I}$ *that extend* σ_o *via Eq. (4.1).*



(a) (b)

Figure 4.6: (a) An entity skeleton for the citation domain. (b) A CPD for the *Exists* attribute of Cites.

4.4 Existence Uncertainty

The second form of link 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.

4.4.1 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 are 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 4.5.

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 4.6(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 4.8: 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 schemas 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 4.9: 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 4.10: *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.*

Proof: We prove this by induction on the stratification level of the class X. Let $\rho_1, \ldots, \rho_k$ be the set of reference slots of X, and let $Y_i = \text{Range}[\rho_i]$. Then $\mathcal{I}(X) = \mathcal{I}(Y_1) \times \cdots \mathcal{I}(Y_k)$. If X is the first level in the stratification, it cannot refer to any undetermined classes. Thus, the number of potential objects in $\mathcal{I}(X)$ is simply the product of the number of objects in each determined class Y_i as specified in the entity skeleton σ_e. Each term is finite, so the product of the terms will be finite.

Next, assume by induction that all of the classes at stratification levels less than i are finite. If X is at level i in the stratification, the constraints imposed by the stratification imply that the range type of all of its reference slots are at stratification levels $< i$. By our induction hypothesis, each of these classes is finite. Hence, the cross product of the classes of the reference slots is finite and the number of potential objects in X 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.

4.4.2 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 4.6(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.[3]

Using the induced relational skeleton and treating the existence events as descriptive attributes, we have set things up so that Eq. (2.2) applies with minor changes. There are two minor changes to the definition of the distribution:

- We want to enforce that $x.E = \mathit{false}$ if $x.\rho.E = \mathit{false}$ 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 non-existent 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

[3] This is similar to the 'encyclopedic' links discussed by [Ghani et al., 2001]. Our exists models can capture each of the types of links (encyclopedic, co-referencing, and partial co-referencing) defined in [Ghani et al., 2001]. Moreover our exists models are much more general, and can capture much richer patterns in the existence of links.

$P^*(X.A \mid \mathrm{Pa}^*(X.A))$.

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

$$P(\mathcal{I} \mid \sigma_e, \Pi) = P(\mathcal{I} \mid \sigma_r[\sigma_e], \Pi^*) = \prod_{X \in \mathcal{X}} \prod_{x \in \sigma_r[\sigma_e](X)} \prod_{A \in \mathcal{A}(x)} P^*(x.A \mid \mathrm{Pa}^*(x.A)) \quad (4.3)$$

We can similarly define the instance dependency graph and the class dependency graph for a PRM Π with existence 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 4.11: *If the class dependency graph G_{Π^*} is acyclic, then there is a stratification of the undetermined classes.*

Proof: The stratification is given by an ordering of *Exists* attributes consistent with the class dependency graph. Because the class dependency graph has an edge from $Y.E$ to $X.E$ for every slot $\rho \in \mathcal{R}(X)$ whose range type is Y, Y will precede X in the constructed ordering. Hence it is a stratification ordering. ∎

Furthermore, based on this definition, we can now easily prove the following result:

Theorem 4.12: *Let Π be a PRM with existence uncertainty and an acyclic class dependency graph. Let σ_e be an entity skeleton. Then Eq. (4.3) defines a coherent distribution on all instantiations $\mathcal{I}$ of the induced relational skeleton $\sigma_r[\sigma_e]$.*

Proof: By Lemma 4.11 and Lemma 4.10 we have that $\sigma_r[\sigma_e]$ is a well-defined relational skeleton. Using the assumption that G_{Π^*} is acyclic, we can apply Theorem 2.5 to Π^* and conclude that Π^* defines a coherent distribution over instances to σ_r, and hence so does Π. ∎

One potential shortcoming of our semantics is that it defines probabilities over instances that include assignment of descriptive attributes to non-existent objects. This potentially presents a problem. An actual instantiation (i.e., a database) will contain value assignments only for the descriptive attributes of existing objects. This

suggests that in order to compute the likelihood of a database we need to sum over all possible values of descriptive attributes of potential objects that do not exist. (As we shall see, such likelihood computations are an integral part of our learning procedure.) Fortunately, we can easily see that the definition of P^* ensures that if $x.E = 0$, then variables of the form $x.A$ are *independent* of all other variables in the instance. Thus, we can ignore such descriptive attributes when we compute the likelihood of a database.

The situation with *Exists* attribute is somewhat more complex. When we observe a database, we also observe that many potential objects do not exist. The non-existence of these objects can provide information about other attributes in the model, which is taken into consideration by the correlations between them and the *Exists* attributes in the PRM. At first glance, this idea presents computational difficulties, as there can be a very large number of non-existent objects. However, we note that the definition of P^* is such that we need to compute $P^*(x.E = \mathit{false} \mid \mathrm{Pa}^*(x.E))$ only for objects whose slots refer to existing objects, thereby bounding the number of non-existent objects we have to consider.

4.5 Example: Word models

Our two models of link uncertainty induce simple yet intuitive models for link existence. We illustrate this by showing a natural connection to the two most common models of word appearance in documents. Suppose our domain contains two entity classes: Document, representing the set of documents in our corpus, and Words, representing the words contained in our dictionary. Documents may have descriptive attributes such as *Topic*; dictionary entries have the attribute *Word*, which is the word itself, and may also have additional attributes such as the type of word. The relationship class Appearance represents the appearance of words in documents; it has two slots *InDoc* and *HasWord*. In this schema, structural uncertainty corresponds to a probabilistic model of the appearance of words in documents.

In existence uncertainty, the class Appearance is an undetermined class; the potential objects in this class correspond to document-word pairs (d, w), and the assertion

Appearance$(d, w).E = true$ means that the particular dictionary entry w appears in the particular document d. Now, suppose that Appearance.E has the parents Appearance.$InDoc.Topic$ and Appearance.$HasWord.Word$. This implies, that, for each word w and topic t, we have a parameter $p_{w,t}$ which is the probability that a word w appears in a document of topic t. Furthermore, the different events Appearance$(d, w).E$ are conditionally independent given the topic t. It is easy to see that this model is equivalent to the model often called *binary naive Bayes model* [McCallum and Nigam, 1998], where the class variable is the topic and the conditionally independent features are binary variables corresponding to the appearance of different dictionary entries in the document.

When using reference uncertainty, we can consider several modeling alternatives. The most straightforward model is to view a document as a bag of words. Now, Appearance also includes an attribute that designates the position of the word in the document. Thus, a document of n words has n related Appearance objects. We can provide a probabilistic model of word appearance by using reference uncertainty over the slot Appearance.$HasWord$. In particular, if we choose $\mathcal{P}$[Appearance.$HasWord$] = Words.$Word$, then we have a multinomial distribution over the words in the dictionary. If we set Appearance.$InDoc.Topic$ as the parent of the selector variable Appearance.$S_{HasWord}$, then we get a different multinomial distribution over words for each topic. The result is a model where a document is viewed as a sequence of independent samples from a multinomial distribution over the dictionary, where the sample distribution depends on the document topic. This document model is called the *multinomial Naive Bayesian model* [McCallum and Nigam, 1998].

Thus, for this simple PRM structure, the two forms of structural uncertainty lead to models that are well-studied within the statistical NLP community. However, the language of PRMs allows us to represent more complex structures: Both the existence and reference uncertainty can depend on properties of words rather than on the exact identity of the word; for example they can also depend on other attributes, such as the research area of the document's author.

4.6 Learning PRMs with Link Uncertainty

The previous sections described two variants of the PRM model with new power. Our aim is to *learn* such models from data: given a schema and an instance, construct a PRM that describes the dependencies between objects in the schema. As in the previous chapter, the PRM model is learned using the same type of training data: a complete instantiation that describes a set of objects, their attribute values and their reference slots. Here, however, we attempt to learn somewhat different models from this data. In the previous chapter, for PRMs with attribute uncertainty, we learn the probability of attributes given other attributes. 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.

As before, separate the learning problem into two tasks: evaluating the "goodness" of a candidate structure, and searching the space of legal candidate structures.

4.6.1 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 re-evaluated 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.

4.6.2 Learning with Existence Uncertainty

The extension of the Bayesian score to PRMs with existence uncertainty is straightforward; 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 non-existent entities. We perform this computation by counting, for each possible instantiation of $\mathrm{Pa}(X.E)$, 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 $\mathrm{Pa}(X.E)$. To compute $C_{X.E}[\mathit{true}, \mathbf{u}]$, we can use standard database query to compute how many objects $x \in \sigma(X)$ have $\mathrm{Pa}(x.E) = \mathbf{u}$. To compute $C_{X.E}[\mathit{false}, \mathbf{u}]$, we need to compute the number of *potential* entities. We can do this without explicitly considering each $(x_1, \ldots, x_k) \in \mathcal{I}(Y_1) \times \cdots \mathcal{I}(Y_k)$ by decomposing the computation as follows: Let ρ be a reference slot of X with $\mathrm{Range}[\rho] = Y$. Let $\mathrm{Pa}_\rho(X.E)$ be the subset of parents of $X.E$ along slot ρ and let $\mathbf{u}_\rho$ be the corresponding instantiation. We count the number of y consistent with $\mathbf{u}_\rho$. If $\mathrm{Pa}_\rho(X.E)$ is empty, this count is simply $|\mathcal{I}(Y)|$. 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

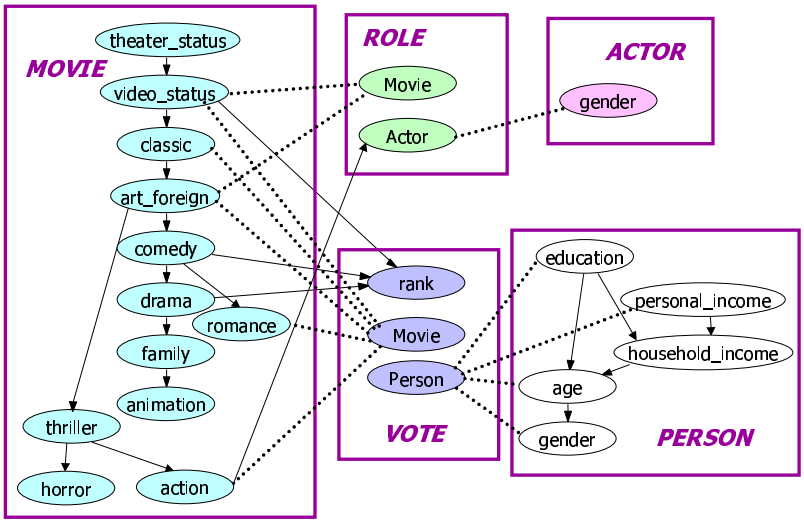


Figure 4.7: Π_{RU}, the PRM learned using reference uncertainty. The reference slots are Role.*Movie*, Role.*Actor*, Vote.*Movie*, and Vote.*Person*. Dashed lines indicate attributes used in defining the partition.

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 $X.A$.

4.7 Experimental Results

We evaluated our learning algorithms on several real-world data sets. In this section, we describe the PRM learned for a domain using both of our models for representing link uncertainty. In each case, we compare against a simple baseline model. Our experiments used the Bayesian score with a uniform Dirichlet parameter prior with equivalent sample size $\alpha = 2$, and a uniform distribution over structures. The partitions are defined using the cross-product of values of the partition attributes; they are not defined using decision trees.

4.7.1 Predictive ability

We first tested whether the additional expressive power allows us to better capture regularities in the domain. Toward this end, we evaluated the likelihood of test data given our learned model. Unfortunately, we cannot directly compare the likelihood of the EU and RU models, since the PRMs involve different sets of probabilistic events and condition on varying amounts of background knowledge. Thus to evaluate our models we compare the PRM with link uncertainty to a "baseline" model which incorporates link probabilities, but makes the "null" assumption that the link structure is uncorrelated with the descriptive attributes. For reference uncertainty, the baseline has $\mathcal{P}[\rho] = \emptyset$ for each slot. For existence uncertainty, it forces $x.E$ to have no parents in the model.

We evaluated these variants on a dataset that combines information about movies and actors from the Internet Movie Database[4] and information about people's ratings of movies from the Each Movie dataset,[5] where each person's demographic information was extended with census information for their zipcode. From these, we constructed five classes (with approximate sizes shown): Movie (1600), Actor (35,000); Role (50,000), Person (25,000), and Vote (300,000).

We modeled uncertainty about the link structure of the classes Role (relating actors to movies) and Vote (relating people to movies). For RU this was done by modeling the reference uncertainty of the slots of these objects. For EU this was done by modeling probability of the existence of such objects. We evaluated our methods using ten-fold cross validation. To do this, we partitioned our data into ten subsets. For each each subset, we trained on nine-tenths of the data (the data not included in the subset) and evaluated the log-likelihood of the held-out test subset. We then average the results. In both cases, the model using link uncertainty significantly outperformed its "baseline" counterpart. For RU, we obtained a log-likelihood of $-149,705$ as compared to $-152,280$ for the baseline model. For EU, we obtained a log-likelihood of $-210,044$ for the EU model, as compared to $-213,798$ for the baseline EU model. Thus, we see that the model where the relational structure is

[4] ©1990-2000 Internet Movie Database Limited.
[5] http://www.research.digital.com/SRC/EachMovie.

correlated with the attribute values is substantially more predictive than the baseline model that takes them to be independent: although any particular link is still a low-probability event, our link uncertainty models are much more predictive of its presence.

Figure 4.7 shows the RU model learned. In the RU model we partition each of the movie reference slots on genre attributes; we partition the actor reference slot on the actor's gender; and we partition the person reference of votes on age, gender and education. An examination of the model shows, for example, that younger voters are much more likely to have voted on action movies and that male action movies roles are more likely to exist than female roles. Furthermore, the actor reference slot has Movie.*Action* as a parent; the CPD encodes the fact that male actors are more likely to have roles in action movies than female actors.

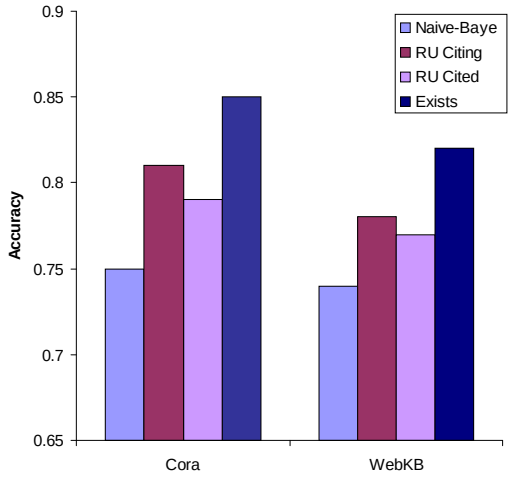
The EU model learned had an exist attribute for both vote and role. In the model we learned, the existence of a vote depended on the age of the voter and the movie genre, and the existence of a role depended on the gender of the actor and the movie genre.

4.7.2 Classification

While we cannot directly compare the likelihood of the EU and RU models, we can compare the predictive performance of each model. In this set of experiments, we evaluate the predictive accuracy of the different models on two datasets. It is quite interesting to note that we observed that both models of link uncertainty significantly increase our prediction accuracy.

We considered the conjecture that by modeling link structure we can improve the prediction of descriptive attributes. Here, we hide some attribute of a test-set object, and compute the probability over its possible values given the values of other attributes on the one hand, or the values of other attributes and the link structure on the other. We tested on two similar domains: Cora [McCallum et al., 2000] and WebKB [Craven et al., 1998]. The Cora dataset contains 4000 machine learning papers, each with a seven-valued *Topic* attribute, and 6000 citations. The WebKB dataset

Table 4.1: Prediction accuracy of topic/category attribute of documents in the Cora and WebKB datasets. Accuracies and reported standard deviations are based on a 10-fold cross validation.



	Cora	WebKB
baseline	75 ± 2.0	74 ± 2.5
RU Citing	81 ± 1.7	78 ± 2.3
RU Cited	79 ± 1.3	77 ± 1.5
EU	85 ± 0.9	82 ± 1.3

contains approximately 4000 pages from four Computer Science departments, with a five-valued attribute representing their "type", and 10,000 links between web pages. In both datasets we also have access to the content of the document (webpage/paper), which we summarize using a set of attributes that represent the presence of different words on the page (a binary Naive Bayes model). After stemming and removing stop words and rare words, the dictionary contains 1400 words in the Cora domain, and 800 words in the WebKB domain.

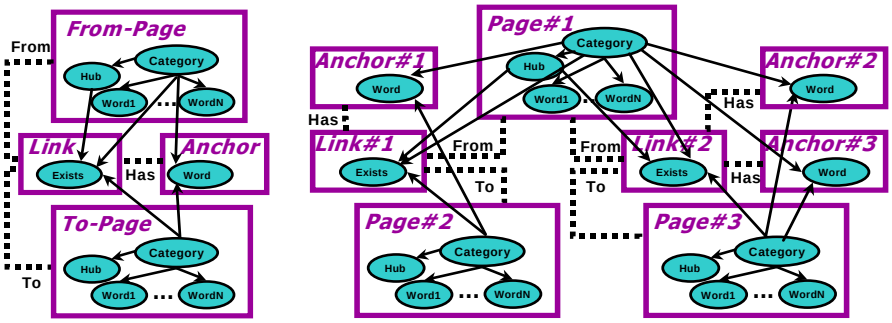
In both domains, we compared the performance of models that use only word appearance information to predict the category of the document with models that also used probabilistic information about the link from one document to another. We fixed the dependency structure of the models, using basically the same structure for both domains. In the Cora EU model, the existence of a citation depends on the topic of the citing paper and the cited paper. We evaluated two symmetrical RU models. In the first, we partition the citing paper by topic, inducing a distribution over the topic of Cites.*Citing*. The parent of the selector variable is Cites.*Cited*.*Topic*. The second model is symmetrical, using reference uncertainty over the cited paper.

Table 4.1 shows prediction accuracy on both data sets. We see that both models of link uncertainty significantly improve the accuracy scores, although existence uncertainty seems to be superior. Interestingly, the variant of the RU model that models reference uncertainty over the citing paper based on the topics of papers cited (or the from webpage based on the categories of pages to which it points) outperforms the cited variant. However, in all cases, the addition of citation/hyperlink information helps resolve ambiguous cases that are misclassified by the baseline model that considers words alone.

For example, paper #506 is a Probabilistic Methods paper, but is classified based on its words as a Genetic Algorithms paper (with probability 0.54). However, the paper cites two Probabilistic Methods papers, and is cited by three Probabilistic Methods papers, leading both the EU and RU models to classify it correctly. Paper #1272 contains words such as rule, theori, refin, induct, decis, and tree. The baseline model classifies it as a Rule Learning paper (probability 0.96). However, this paper cites one Neural Networks and one Reinforcement Learning paper, and is cited by seven Neural Networks, five Case-Based Reasoning, fourteen Rule Learning, three Genetic Algorithms, and seventeen Theory papers. The Cora EU model assigns it probability 0.99 of being a Theory paper, which is the correct topic. The first RU model assigns it a probability 0.56 of being Rule Learning paper, whereas the symmetric RU model classifies it correctly. In this case, an explanation of this phenomenon is that most of the information for this paper is in the topics of citing papers; it appears that RU models can make better use of information in the parents of the selector variable than in the partitioning variables.

4.7.3 Collective Classification

In the preceding experiments, the topics of all the linked papers or webpages were observed and we made a prediction for a single unobserved topic. In a more realistic setting, we will have a whole collection of unlabelled instances that are linked together. We can no longer make each prediction in isolation, since for example, the (predicted) topic of one paper influences the (predicted) topics of the papers it cites.



(a) (b)

Figure 4.8: (a) PRM Model for WebKB domain; (b) Fragment of unrolled network for WebKB model.

This task of collective classification requires us to reason about the entire collection of instances at once. In our framework, this translates into computing the posterior distribution over the unobserved variables given the data and assigning each unobserved variable its most likely value. This requires inference over the unrolled network defined by instantiating a PRM for a particular document collection. We cannot decompose this task into separate inference tasks over the objects in the model, as they are all correlated. In general, the unrolled network can be fairly complex, involving many documents that are linked in various ways. (In 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. Following Taskar et al. [2001], we use belief propagation for the task of inference in PRMs. An overview of the belief propagation algorithm was given in Section 2.5.2.

We evaluated several existence uncertainty models for the task of collective classification on the WebKB dataset [Craven et al., 1998]. Recall that the WebKB dataset consists of webpages in the Computer Science departments of four schools: Cornell, University of Texas at Austin, University of Washington, and University of Wisconsin.

Figure 4.8(a) shows a PRM for this domain. For clarity, the Page class is duplicated in the figure, once as From-Page and once as To-Page. Each page has a category attribute representing the type of web page which is one of {course, professor, student, project, other}. The text content of the web page is represented using a set of binary attributes that indicate the presence of different words on the page. After

stemming, removing stop words and rare words, the dictionary contains around 800 words.

In addition, each page was labelled with a "category hub" attribute, whose domain is {*course, professor, student, project, none*}. A page was labelled with a hub of a particular type (*course*, *professor*, *student*, or *project*) if it pointed to many pages of that category. The prevalence of such hubs (or directories) on the web was investigated by Kleinberg [1999], who noted that pages of the same topic or category are often linked to by hub pages. This insight was utilized in the classification algorithm FOIL-HUBS of [Slattery and Mitchell, 2000] for the WebKB dataset. Pages in the training set were hand-labeled with the hub attribute, but the attribute was hidden in the test data. Each school had one hub page of each category, except for Washington which does not have a project hub page and Wisconsin which does not have a faculty web page in the data set.

The data set also describes the links between the web pages within a given school. In addition, for each link between pages, the dataset specifies the words (one or more) on the anchor link. There are approximately 100 possible anchor words.

In these experiments, we included only pages that have at least one out link. The number of resulting pages for each school are: Cornell (318), Texas (319), Washington (420), and Wisconsin (465). The number of links for each school are: Cornell (923), Texas (1041), Washington (1534) and Wisconsin (1823).

Given a particular set of hyperlinked pages, the template is instantiated to produce an "unrolled" Bayesian network. Figure 4.8(b) shows a fragment of such a network for three webpages. The two existing links from page 1 to page 2 and 3 are shown while non-existing links are omitted for clarity (but still play a role in the inference). Also shown are the anchor word for link 1 and two anchor words for link 2. Note that during classification, existence of links and anchor words in the links are used as evidence to infer categories of the web pages. Hence, our unrolled Bayes net has active paths between categories of pages through the v-structures at Link.*Exists* and Anchor.*Word*. These active paths capture exactly the pattern of relational inference we set out to model.

We compared the performance of several models on predicting web page categories.

In each case, we learned a model from three schools, and tested the performance of the learned model on the remaining school. Our experiments used the Bayesian score with a uniform Dirichlet parameter prior with equivalent sample size $\alpha = 2$.

All models we compared can be viewed as a subset of the model in Figure 4.8(a). Our baseline is a standard binomial Naive Bayes model that uses only words on the page to predict the category of the page. We evaluated the following set of models:

1. **Naive-Bayes**: Our baseline model.
2. **Anchors:** This model uses both words on the page and anchor words on the links to predict the category.
3. **Exists**: This model adds uncertainty over the link relationship to the simple baseline model; the parents of Link.*Exists* are Link.*From-Page.Category* and Link.*To-Page.Category*.
4. **Ex+Hubs**: This model extends the **Exists** model with Hubs. In the model Link.*Exists* depends on *Link.From-Page.Hub* in addition to the categories of each of the pages.
5. **Ex+Anchors**: This model extends the **Exists** model with includes anchor words (but not hubs).
6. **Ex+Hubs+Anchors**: The final model includes existence uncertainty, hubs and anchor words.

Figure 4.9 compares the accuracy achieved by the different models on each of the schools. The final model, **Ex+Hubs+Anchors**, which incorporates existence uncertainty, hubs and anchor words, consistently outperforms the **Naive-Bayes** model by a significant amount. In addition, it outperforms any of the simpler variants.

Our algorithm was fairly successful at identifying the hubs in the test set: it recognized 8 out of 14 hubs correctly. However, precision was not very high: 38 pages were mislabeled as hubs. The pages mislabeled as hubs often pointed to many pages that had been labeled as Other web pages. However, on further inspection, these hub pages often *were* directories pointing to pages that were likely to be researcher home pages or course home pages and seemed to have been mislabeled in the training set

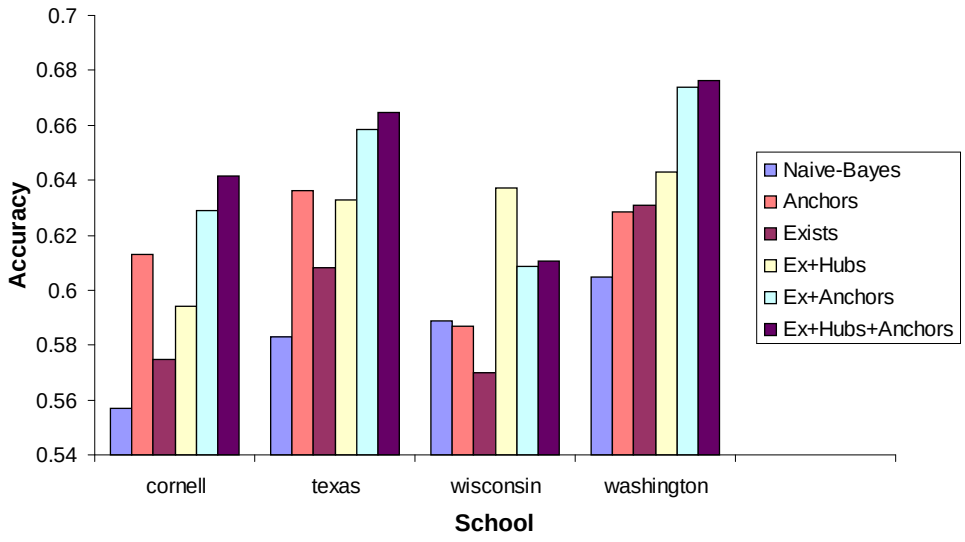


Figure 4.9: Comparison of accuracy of several models ranging from the simplest model, **Naive-Bayes** to the most complex model, **Ex+Hubs+Anchors**, which incorporates existence uncertainty, hubs and link anchor words. In each case, a model was learned for 3 schools and tested on the remaining school.

as Other. We investigated how much these misclassifications hurt the performance by revealing the labels of the hub attribute in the test data. The improvement in classification accuracy of **Ex+Hubs+Anchors** model was roughly 2%.

4.8 Conclusions

In this chapter, we proposed two representations for link uncertainty: reference uncertainty and existence uncertainty. Reference uncertainty models the process by which reference slots are selected from a given set. Existence uncertainty provides a model for whether a relation exists between two objects. We have shown how to integrate them within our learning framework, and presented results showing that they allow interesting patterns to be learned. The ability to learn probabilistic models of relational structure has many applications. It allows us to predict whether two objects with given properties are more likely to be related to each other. More surprisingly, the link structure also allows us to predict attribute values of interest. For example,

we can better predict the topic of a paper by using the fact that it cites certain types of papers. Both of the models we propose are relatively simple. They make certain independence assumptions that often will not hold. Thus the significance of our results is that even these simplistic models of link uncertainty provide us with increased predictive accuracy.

The ability to learn probabilistic models of relational structure is an exciting new direction for machine learning. Our treatment here only scratches the surface of this problem. In particular, although useful, neither of the representations proposed for structural uncertainty is entirely satisfying as a generative model. Furthermore, both models are restricted to considering the probabilistic model of a single relational "link" in isolation. These simple models can be seen as the naive Bayes of structural uncertainty; in practice, relational patterns involve multiple links, e.g., the concepts of hubs and authorities. In future work, we hope to provide a unified framework for representing and learning probabilistic models of relational "fingerprints" involving multiple entities and links.

Chapter 5

PRMs with Class Hierarchies

Here we examine the benefit that additional domain structure in the form of IS-A hierarchies provides during the construction of PRMs. We show how the introduction of subclasses allows us to use inheritance and specialization to refine our models so that they more accurately capture the dependencies in the data. We show how to learn PRMs with class hierarchies (PRM-CH) in two settings. In the first, the class hierarchy is provided, as part of the input, in the relational schema for the domain. In the second setting, in addition to learning the PRM, we must learn the class hierarchy. An important benefit provided by the class hierarchy mechanism is that PRM-CHs allow us to build models that can refer to both particular instances in our domain, and classes of objects in our domain. This capability allows us to bridge the gap between a class-based model and an attribute-value-based model.

5.1 Introduction

Consider the problem of collaborative filtering [Resnick et al., 1994, Shardanand et al., 1995, Breese et al., 1998], in which we utilize collected individual preferences to make a collaborative recommendation. Ideally, by finding individuals with similar tastes or preferences to a new user, we can use their combined preferences to make predictions for the new user's ratings for services or products. The canonical example is a movie recommender system, in which we have the rankings by large number of people on

a collection of movies. We then make recommendations by finding users who have similar taste to the new user (i.e., who have given similar rankings on some subset of the movies), and, for movies the new user has not yet seen, use the collective prediction of similar users to predict the new user's reaction.

One method for building a recommender system is to build a model of movie viewers and the movies that they enjoy. One approach, though certainly not the most common one, is to build a Bayesian network over the movies. The model represents the preferences of a single viewer, which has a random variable for each movie. Each person in the training set has a vector that represents the movies that they have watched, and their ratings for that movie. From this data set, we can learn a Bayesian network that represents the correlations between their preferences for the different films. Thus, we could learn that the user's rating of one movie, say "Forrest Gump", depends on her ratings for the movies that are its parents in the learned network, say "Big" and "Caddy Shack". Breese et al. [1998] compare this approach to other collaborative filtering approaches and show that it is superior in its ability to predict TV-show preferences.

This approach is limited in that it models only the relationships between instances of one class, the movies. We cannot model broad dependencies, such as whether a person enjoys British comedies depends on whether they like American slap-stick comedies. In addition, we cannot model relationships between people. For example, if my roommate recommends "Four Weddings and a Funeral", I may be more likely to take her advice and rent the movie (assuming that I think we have similar tastes; her recommendation may also have the opposite effect motivating me to avoid the movie entirely).

As we have seen, PRMs allow us to represent rich dependency structures, involving multiple entities and the relations between them; they allow the attributes of an entity to depend probabilistically on properties of related entities. However, PRMs model the domain at the *class* level; i.e., all instances in the same class share the same dependency model. This model is then instantiated for particular situations. For example, a person's ratings for a movie can depend both on the attributes of the person and the attributes of the movie. For a given situation, involving some set of

people and movies, this dependency model will be used several times. This allows us, for example, to use the properties and ratings of one person to reach conclusions about the properties of a movie (e.g., how funny it is), and thereby to reach conclusions about the chances that another viewer would like it.

In Chapter 3 we showed how to learn PRMs from relational data, and presented techniques for learning both parameters and the probabilistic dependency structure for the attributes in a relational model. This learning algorithm exploits the fact that the models are constructed at the class level. Thus, an observation concerning one person and one movie is used to refine the class model applied to all people and all movies, hence making much broader use of our data. This has the advantage that we can learn robust statistical models despite the fact that each individual may have seen only a few movies and each movie may have been seen by only a small number of people.

However, this class-based approach also has disadvantages: all elements of the same class must use the same model. For example, we cannot have the rating of a user for documentaries depend on one set of parents, and his ratings for comedies depend on another set of parents. Thus, we can not specialize the local probability models depending on the movie category or class. In addition, we cannot have the rating for documentaries depend on the rating for action movies (here for example, the correlation may be negative). The dependency model for these two ratings must be identical, and we cannot have the rating for a movie depend on itself. Finally, we cannot have the rating for a particular movie such as "Forest Gump" depend on the rating for another movie instance such as "Big": The dependency model for these two ratings must be identical, and we cannot have the rating for a movie depend on itself.

In this chapter, we propose methods for discovering useful refinements of a PRM's dependency model. We begin in Section 5.2 by defining *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, 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 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 I enjoy action movies may depend on whether I enjoy documentaries. PRMs-CH provide a general mechanism that allow us to define a rich set of dependencies. In fact, they provide the basic representational power that will allow us to model dependency models for individuals (as done in Bayesian Networks) and dependency models for categories of individuals (as done in PRMs).

In Section 4.6 we turn to some of the practical issues involved in learning PRMs-CH. First, we examine the case where the class hierarchy is given as input, as part of the relational schema. Our learning task is then simply to choose the appropriate level at which to model the probabilistic dependencies — at the class level, or specialized according to some subclass. We then turn to the case where the class hierarchy is not provided, and in addition to learning the probabilistic model, we must also discover the structure of the class hierarchy. In Section 5.4, we present some experimental results illustrating how we have expanded the space of probabilistic models considered by our learning algorithm, and how this allows us to learn more expressive and more accurate models.

As described in chapter Chapter 2, PRMs extend the representational power of (propositional) BNs to relational domains. In chapters Chapter 3 and Chapter 4 we examined how to learn richer probabilistic relational models. We described the semantics of the models and proposed learning algorithms for models with attribute uncertainty and models with structural uncertainty. In this chapter, we describe models that combine a class-based relational model with more traditional instance-based Bayesian networks. The tool we will use to provide this capability is a class inheritance hierarchy.

5.2 PRMs with Class Hierarchies

In this section, we describe refinements of our probabilistic model using class hierarchies. 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) independent and identically distributed. By contrast, in the BN model for this domain, each movie could have a different dependency model; we could even have one depend on the other.

5.2.1 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".

Formally, we introduce the notion of a probabilistic class hierarchy, similar to that introduced in Koller and Pfeffer [1997, 1998]. We assume that the original set of classes define, at the schema level, the structure of an object (attributes and slots associated with it). Unlike the subclass mechanism in Koller and Pfeffer [1997, 1998], subclasses do not change this object structure.

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 rooted directed acyclic graph defined by a subclass relation $\prec$ over a a finite set of subclasses $\mathcal{C}[X]$. For $c, d \in \mathcal{C}[X]$,

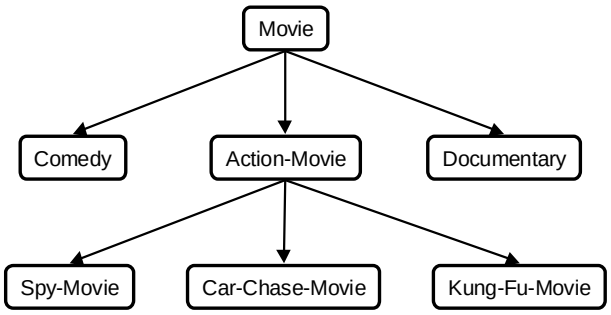


Figure 5.1: A simple class hierarchy for Movie.

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 we may have the class Movie and its direct subclasses Comedy, Action-Movie, and Documentary. The subclass Action-Movie might, in turn, have the direct subclasses Spy-Movie, Car-Chase-Movie, and Kung-Fu-Movie. We have that Spy-Movie is a direct subclass of Action-Movie, and a subclass (but not a direct one) of the root class Movie. Figure 5.1 shows the simple class hierarchy we have just described.

We define the leaves of the hierarchy to be the *basic subclasses*, denoted $basic(H[X])$. We achieve subclassing for a class X by requiring that there be an additional subclass indicator attribute $X.Class$ that determines the basic class to which an object belongs (in theory objects could be members of a non-basic class, however we have not examined that possibility here). Thus, if c is a subclass, then $\mathcal{I}(X_c)$ contains all objects $x \in X$ for which $x.Class \prec^* c$, i.e., all objects that are in some basic class which is a subclass of c. In our example, Movie has a subclass indicator variable Movie.*Class* with the five possible values

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

.

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 5.1: 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 $X.Class$ such that $\mathcal{V}(X.Class) = basic(H[X])$;
- a CPD for $X.Class$ (here we require that $X.Class$ has no parents.
- 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 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 discussion from the introduction, if the popularity of action movies 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.

5.2.2 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 [Boutilier et al., 1996, Chickering et al., 1997]. 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 defined 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 his 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 5.2:

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 their votes, we use the inverse of slot reference Person.*Votes*. Now we can construct refinements of Person.*Votes*, $Votes_{\langle \text{Comedy-Vote} \rangle}$, $Votes_{\langle \text{Action-Vote} \rangle}$ and $Votes_{\langle \text{Documentary-Vote} \rangle}$.

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*).

5.2.3 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 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. [1998] 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 Bonds Movies).

5.2.4 Semantics

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

$$P(\mathcal{I} \mid \sigma_r, \Pi) = \prod_{X} \prod_{x \in \sigma_r(X)} P(x.Class) \prod_{A \in \mathcal{A}(X)} P(x.A \mid \mathrm{Pa}^{x.c}(x.A)) \tag{5.1}$$

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.

5.2.5 Coherence of Probabilistic Model

At some level, the introduction of a class hierarchy introduces no substantial difficulties — the semantics of the model remain unchanged. Given a relational skeleton σ_r, and subclass information for each object, a PRM-CH Π_{CH} specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with σ_r which was given by

Eq. (5.1).

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 chapters, we define our coherence constraints using an instance dependency graph, relative to our PRM and relational skeleton.

Definition 5.3: 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 $x.Class$.

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, $x.Class$, 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 $X.\tau.B \in \mathrm{Pa}^*(X.A)$, if $y \in x.\tau$ in σ_r, we define an edge $y.B \rightarrow x.A$. If the CPD has been specialized, or if τ 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 it is related to, 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 grayed 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 5.4: 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 $\mathrm{Pa}^d(X.A)$, and white otherwise. In addition, the edges corresponding to any for 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 Eq. (5.1) specifies a coherent probability distribution.

Theorem 5.5: *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 Eq. (5.1).*

Proof: The probability of an instantiation $\mathcal{I}$ is the joint distribution over a set of random variables defined via the relational skeleton. We have the following variables:

- We have one random variable $x.A$ for each $x \in \sigma_r(X)$ and each $A \in \mathcal{A}(X)$.
- We have one random variable $x.Class$ for the class indicator variable for each $x \in \sigma_o(X)$.

Let $V_1, \ldots, V_N$ be the random variables defined above. We show that because the instance dependency graph is acyclic for any instantiation of the subclass indicators, we can construct an ordering $V_1, \ldots, V_N$ that is a topological sort of the instance dependency graph; *but* this ordering will be constructed dynamically, once we know the subclass information.

As in the proof of Theorem 2.5, our proof relies on the fact that the probability of any instantiation $\mathcal{I}$ is the product of legal conditional distributions, hence the product is a well-defined joint distribution. The distribution is defined via Eq. (5.1). We must ensure that the conditional distribution for each random variable is well-defined and can be determined by the time that it is required in the product. Because the CPDs come directly from the PRM-CH, the first requirement is satisfied trivially. So it remains to check that the variable ordering and choice of CPDs can be determined at the point at which they are needed.

The definition of acyclicity provides us with the necessary procedure for constructing the variable ordering. Because the subclass indicator variables do not have parents, we can place all of the subclass indicators, $x.Class$, for the objects at the start of our order. Once their values have been probabilistically chosen, we are in the position to color all of our edges either black or white. And once we know all of the black edges, because of the acyclicity, we know there is some topological ordering of the nodes that is consistent with the black edges. This provides us with our variable ordering.

Because the instance dependency graph is acyclic and each CPD in Eq. (5.1) can be determined and is well-defined, Eq. (5.1) is a well-defined joint distribution. ∎

As in the previous case of PRMs with attribute uncertainty and PRMs with link 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.

Definition 5.6: 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 $X.Class$.

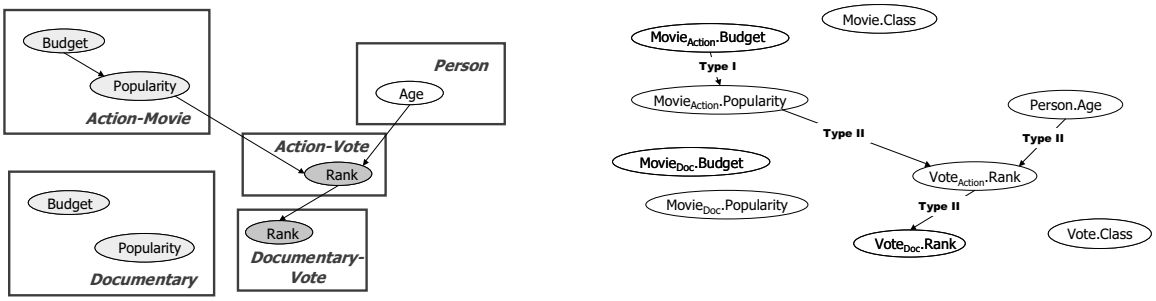
and the following edges:

- **Type I edges:** For any node $X_c.A$ and formal parent $X_c.B \in \mathrm{Pa}^c(X_c.A)$ we have an edge $X_c.B \rightarrow X_c.A$.
- **Type II edges:** For any attribute $X_c.A$ and formal parent $X_c.\rho.B \in \mathrm{Pa}^c(X_c.A)$, where $\mathrm{Range}[\rho] = Y$, 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 5.2 shows a simple class dependency graph for our movie example. The PRM-CH is given in Figure 5.2(a) and the class dependency graph is shown in Figure 5.2(b).

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

Lemma 5.7: *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.*



(a) (b)

Figure 5.2: (a) A simple PRM with class hierarchies for the movie domain (b) The class dependency graph for this PRM.

Proof: As before, we prove this by contradiction by showing that if there is a cycle in the instance dependence graph, then we can construct a cycle in the class dependency graph.

Suppose there is a cycle $x_1.A_1 \rightarrow x_2.A_2 \cdots x_k.A_k \rightarrow x_1.A_1$ in the instance dependency graph. Each of these object edges corresponds to an edge in the class dependency graph. Consider edge $x_i.A_i \rightarrow x_{i+1}.A_{i+1}$. Let $x_i.Class = c$. There are three cases:

- $x_i.A_i$ is the parent of $x_{i+1}.A_{i+1}$ and they are in the same object. Then the corresponding edge in the class dependency graph is the type I edge, $X_{c'}.A_i \rightarrow X_{c'}.A_{i+1}$, where c' may be a superclass of c (if this CPD is inherited).
- $x_i.A_i$ is the parent of $x_{i+1}.A_{i+1}$ and the parent is defined via a slot chain $x_i.\tau.A_{i+1}$. Let d be the subclass of x_{i+1}. As before there is some type II edge from $X_{c'}.A_i \rightarrow X_{d'}.A_{i+1}$, where c' is a superclass of c (in the case where the CPD was inherited from c'). Now however d' may be a superclass of d. So, we then follow type III edges from $X_d.A_{i+1}$ to $X_{d'}.A_{i+1}$.

.

In either case, we can find corresponding edges in our class dependency graph from which we can construct a cycle, which contradicts our hypothesis that the class dependency graph is acyclic. ∎

And again we have the following corollary:

Corollary 5.8: *Let Π_{CH} be a PRM with class hierarchies whose class dependency structure $\mathcal{S}$ is acyclic. For 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 Eq. (5.1).*

5.3 Learning PRM-CHs

As in previous chapters, we separate the learning problem into two basic questions: how to evaluate the "goodness" of a candidate structure, and how to search the space of legal candidate structures. We consider each question separately. 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.

5.3.1 Class Hierarchies Provided in Schema

We now turn to learning PRMs with class hierarchies. We begin with the simpler scenario, where we assume that the class hierarchy is given as part of input.

As in Chapter 3, 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 subclass. 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.

5.3.2 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 that 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 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 $\text{Dom}[\rho'] = D$ and $\text{Range}[\rho'] = X$.

5.3.3 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 $\mathsf{Split}(X, c, X.\tau.B)$, where c is a leaf in the current decision tree for $X.Class$ and $X.\tau.B$ 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 towards 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.

5.4 Experimental Results

5.4.1 Model Comparison

We begin with some preliminary results testing whether the increased representational power allowed in PRM-CHs in fact improves the quality of the models we learn. We compared the utility of models with subclasses (Π_{CH}) to our standard PRMs that do not support the refinement of class definitions (Π). Here we are given the structure, we do not allow inheritance of CPDs and we simply learn the parameters. We compare the log-likelihood of a test set for each model.

We present results for the Each Movie dataset[1]. Recall that this dataset contains

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

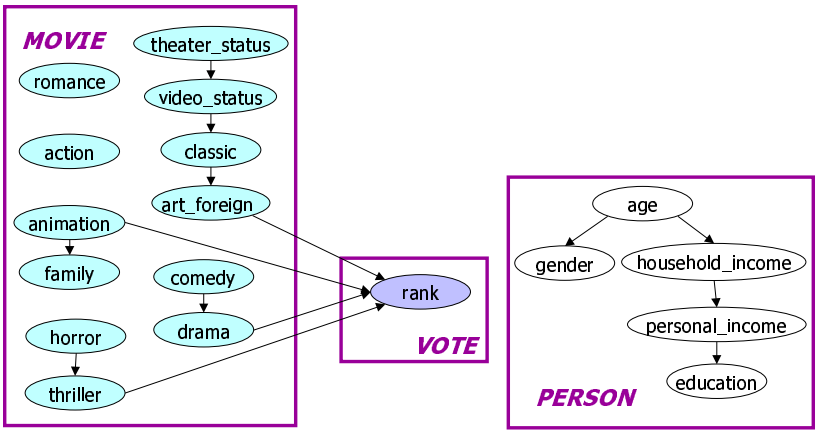


Figure 5.3: A PRM for the Movie domain.

information information about people's ratings of movies. We extended the demographic information we had for the people by including census information available for a person's zip-code. The three classes are Movie, Person, and Votes. The training set contained 1467 movies, 5210 people and 243,333 votes.

We defined a rather simple class hierarchy for Votes, based on the genre of the Movie, Action-Votes, Romance-Votes, Comedy-Votes and Other-Votes. We learned two different models, one that made use of the class hierarchy (Figure 5.4) and one that did not (Figure 5.3). We then evaluated the models on five different test sets. Note that, in relational data, different test sets have markedly different structure, so trying the model on different test sets might result in very different answers. Each test set had 1000 votes, and approximately 100 movies and 115 people. The average log-likelihood of the test set for Π was -12079 with a standard deviation of 475.68. The model with class hierarchies, Π_{CH}, performed much better, with average log-likelihood of -10558 and a standard deviation of 433.10. Using a standard t-test, we obtain that Π_{CH} is better than Π with well over 99% confidence interval.

Looking more closely at the qualitative difference in structure between the two models, we see that the PRM-CH is a much richer model. For example the dependency

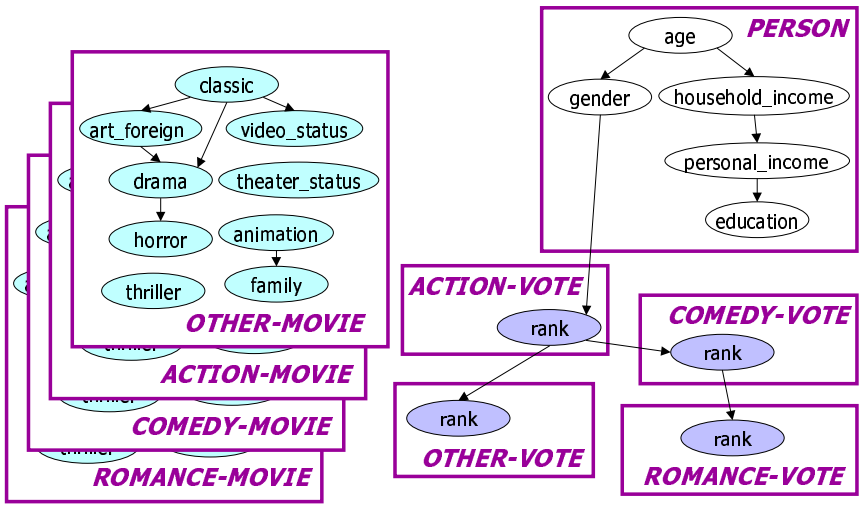


Figure 5.4: Π_{CH}. The links between vote rankings follows a slot chain, from a person's ranking on one class of movies to the person's ranking on another class of movies.

model for Vote.*Rank* cannot be represented without making use of the class hierarchy to both refine the attributes and refine the allowable slot chains. For example, we learn a dependence of $\mathsf{Vote}_{Romance}.Rank$ on $\mathsf{Vote}_{Romance}.Person.Comedy\text{-}Votes.Rank$, whereas $\mathsf{Vote}_{Action}.Rank$ depends on $\mathsf{Vote}_{Action}.Person.Gender$. Not only are these two dependency models different, but they would be cyclic if interpreted as a standard PRM. Note that in the PRM shown in Figure 5.3, there is no dependency between Vote.*Rank* and attributes of Person, so the PRM that uses class hierarchies allows us to discover dependencies on properties of Person that we could not uncover before.

5.5 Conclusion

In the chapter, we have proposed a method for making use of class hierarchies while learning PRMs. Class hierarchies give us additional leverage for refining our probabilistic models. They allow us to automatically disentangle our dependency model, allowing us to construct acyclic dependencies between elements within the same class.

They also allow us to span the spectrum between class level and instance level dependency models.

However, using class hierarchies significantly expands an already complex search algorithm. The search space for PRMs-CH is much larger. In this chapter, we describe a general search algorithm. However, a key to the success of the algorithm is the discovery of useful heuristics to guide the search. In future work, we intend to explore the space of possible heuristics, and to test empirically which heuristics work well on real-world problems.

Chapter 6

Statistical Relational Models

In this chapter, we describe a second type of probabilistic model for relational domains, a statistical relational model (SRM). While on the surface PRMs and SRMs share many similarities, a statistical relational model has significantly different semantics from a PRM and supports answering a different category of queries. An SRM is a statistical model of a particular database instantiation. The SRM captures the tuple frequencies in the database. It can be used to give (approximate) answers to queries on this database. In this chapter, we describe the semantics of SRMs and a learning algorithm for SRMs. In addition, we describe an application of SRMs to a fundamental problem in database theory: the task of estimating the result size of a database query (the number of tuples in the result). Our method provides an important new unified framework for the estimating the result size of complex queries over multiple tables.

6.1 Motivation

As we mentioned in the introduction, there are two distinct approaches to first-order logics of probability [Halpern, 1990]. The PRM model that we have been discussing up until this point is based on the possible-worlds approach; it defines a distribution over possible instantiations of the database. In this chapter, we introduce SRMs, which are based on an alternate semantics defined in terms of frequencies in the database.

We will see how these models are defined, how to make the connection between the database frequencies and the statistical model, and how to answer queries efficiently using these models.

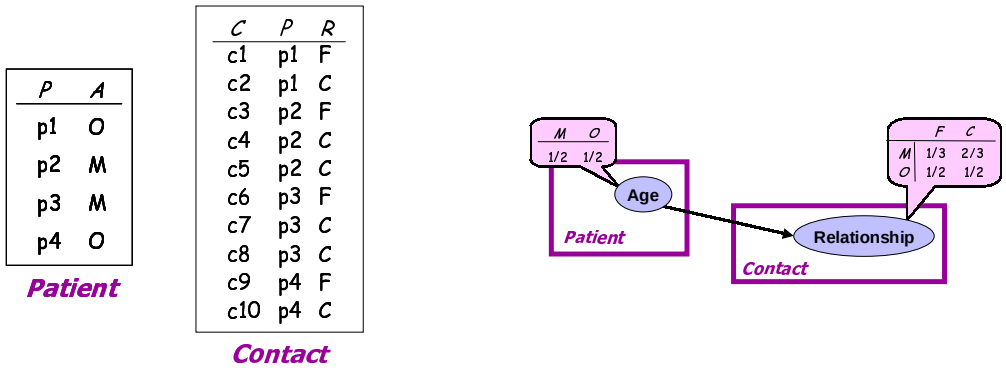
6.1.1 An Example

Consider two tables from the medical database of Section 3.5: Patient, containing information about tuberculosis (TB) patients, and Contact, describing people with whom a patient has had contact, and who may or may not be infected with the disease. Figure 6.1(a) shows a simplified version of the database, where the patient table has one attribute *Age*, which can take on values {*middle-aged*, *old*}, and the contact table has one attribute *Relationship* which can take on values {*family-member*, *coworker*} (In the figure, we abbreviate the variable and value names using their first letter.) Suppose we are interested in answering queries involving a join between these two tables. For example, we may be interested in the probability that the following query is satisfied:

SELECT *
FROM Patient p, Contact c
WHERE p.*age* = *old* **AND**
c.*Patient* = p.*Patient-ID* **AND**
c.*relationship* = *coworker*

I.e., the probability that a randomly chosen patient is elderly and has a contact who is a coworker. In our simple example, out of the ten Patient-Contact pairs, there are two in which the patient is old and the contact is a coworker, so computing the probability from the frequencies in the database, the probability that a randomly chosen patient is elderly and has a coworker contact is $\frac{1}{5}$.

A simple approach to this problem would proceed as follows: We begin by assuming referential integrity, in other words that each tuple in Contact joins with exactly one tuple in Patient. Then the size of the joined relation, prior to the selects, is |Contact|. We compute the probability p of Patient.*age* = *old* and the probability q of Contact.*relationship* = *coworker*, and estimate the probability of the resulting query being satisfied as $p \cdot q$, or $\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$.



(a) (b)

Figure 6.1: (a) A very simple TB database (b) A PRM for the DB

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

Now consider using a PRM to answer this query. Suppose, as shown in our example database, that there are two middle-aged patients and they each have three contacts, of which two are coworkers and one which is a family member, and that there are two older patients, each with two contacts, of which one is a coworker and the other is a family member. Suppose from this database, we learn the PRM shown in Figure 6.1(b), which has an attribute Patient.*Age* which has probability $\frac{1}{2}$ of having the value *middle-aged* and probability $\frac{1}{2}$ of having the value *old*, and

attribute Contact.*Relationship* which, for middle-aged patients, has probability $\frac{2}{3}$ of being a coworker, and probability $\frac{1}{3}$ of being a family member, and for older patients has probability $\frac{1}{2}$ of being a coworker, and probability $\frac{1}{2}$ of being a family member. Then if we computed the probability that a randomly chosen patient is older and has a contact that is a coworker, we might naively compute this by multiplying the probabilities from the PRM of a patient being old and the probability of the contact being a coworker given that the patient is old. This gives us $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Whereas the true probability is $\frac{1}{5}$!

6.1.2 Our Approach

We address these issues by providing a more accurate model of the joint frequency distribution of tuples from a database. In this chapter we provide the machinery, but before turning to development of a general theory, let us first illustrate our approach in a simple setting.

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

This sampling process induces a distribution

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

over the values of the join indicator J_F, and the descriptive attributes $\mathcal{A}(R) = \{A_1, \ldots, A_n\}$ and $\mathcal{A}(S) = \{B_1, \ldots, B_m\}$.

Now, consider any query Q over R and S of the form: $r.\mathbf{A} = \mathbf{a}, s.\mathbf{B} = \mathbf{b}, r.F = s.K$ (where we abbreviate a multidimensional select using vector notation). The probability that this query is satisfied by a randomly chosen tuple r from R and s

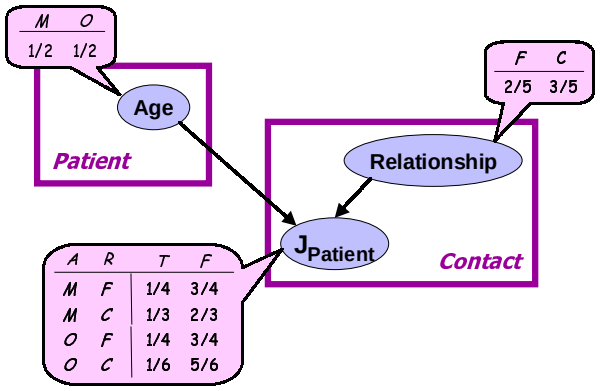


Figure 6.2: An SRM for the simple TB DB.

from S is simply:

$$P_{\mathcal{D}}(\mathbf{A} = \mathbf{a}, \mathbf{B} = \mathbf{b}, J_F = true).$$

In other words, we can estimate the result for any query of this form using the joint distribution $P_{\mathcal{D}}$ defined using our sampling process.

Consider our simple example from before. We introduce a join indicator variable $J_{Patient}$ which has parents $\mathsf{Contact}.Patient.Age$ and $\mathsf{Contact}.Relationship$. In this example, the CPD for the join indicator variable is computed using the frequencies in the database. For any particular value of Age and $Relationship$, (a, r), the probability that the join indicator is true is:

$$\frac{|p.Patient\text{-}ID = c.Patient \ \& \ p.Age = a \ \& \ c.Relationship = r|}{|p.Age = a| \cdot |c.Relationship = r|}$$

In other words, it is the proportion of tuples that join over the number of potential joins.

Then using our SRM to compute the probability that two randomly chosen tuples satisfy our query is simply:

$$P_{\mathcal{D}}(\mathsf{Patient}.Age = old \ \& \ \mathsf{Contact}.Type = coworker \ \& \ \mathsf{Contact}.J_{Patient} = true)$$

$$= \frac{1}{2} \cdot \frac{3}{5} \cdot \frac{1}{6} = \frac{1}{5}$$

which is the correct answer according to the frequencies in the database.

As we now show, an extension of the techniques in Chapter 3 allows us to estimate this joint distribution using a probabilistic graphical model.

6.2 The Relational Database

We begin by examining the distribution induced by the frequencies of various combinations of values occurring in the database. In the case where we have a single table, this task is relatively straightforward. The distribution is defined over a joint assignment of values of random variables, where we have one random variable for each attribute in the table, and the distribution is just the probability that the attributes of a randomly chosen tuple take on each particular joint assignment. In the case where we have more than one table, the definition becomes a bit trickier; we need to know something about the connection between the tuples from the different tables. In the case where tables can join in more than one way (i.e., in the case of foreign-key joins, when there is more than one foreign-key reference to a table), the problem is even more ill-defined.

In this section, we introduce the necessary machinery to construct a unique distribution from a relational schema and a database. The construction hinges on the notion of a 'universal foreign-key closure'. At an abstract level this defines the random variables in our distribution and the foreign-key join relationships among them. But, before we can get to the definition, we start with several of the necessary building blocks. In some cases, we are just reviewing definitions from earlier chapters, couching them in standard database terminology (for example we define foreign-keys, which are analogous to reference slots). However, in addition, we introduce several new concepts.

6.2.1 The Relational Schema

The relational schema defines the structure of our database. As in earlier chapters, it defines the classes (or in this case tables) and the attributes of the tables.

Definition 6.1: A *relation schema* $\mathcal{R}$ consists of a set of tables $\mathcal{R} = \{R_1, \ldots, R_m\}$. Each table R is associated with attributes of three types: a single primary key $R.K$,[1] a set $\mathcal{F}(R)$ of foreign keys, and a set $\mathcal{A}(R)$ of descriptive attributes. Each foreign key $R.F$ is associated with a table into which it points, $\text{Dom}[R.F] \in \mathcal{R}$. Each descriptive attribute $R.A$ is associated with a domain of possible values $\mathcal{V}(R.A)$. ∎

6.2.2 The Database

As before, a database is simply an instantiation of the schema.

Definition 6.2: A database $\mathcal{D}$ over $\mathcal{R}$ consists of a set of tuples $\mathcal{T}[R]$ for each table R. For each $t \in \mathcal{T}[R]$:

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

Note that the second bullet in our definition restricts attention to databases satisfying *referential integrity*: Let R be a table and let F be a foreign key in R that refers to some table S with primary key K; then for every tuple $r \in R$ there must be some tuple $s \in S$ such that $r.F = s.K$. We require referential integrity, but our model can easily be extended to accommodate cases where the foreign key takes the value *null*, indicating that there is no related tuple in $\text{Dom}[R.F]$.

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

6.2.3 The Distribution Induced by a Query

In a number of different places in this chapter, we will be interested in computing the answer to database queries. In some cases, we will limit the queries we consider, However, at the start, we will be considering general select-join queries of the following form: A query over a relational schema $\mathcal{R}$ is defined using tuple variables and the standard relational operators $\times$ (cross-product), σ (select), and $\bowtie$ (join). A tuple variable r is typed, i.e., $\text{Dom}[r] = R$, for some $R \in \mathcal{R}$, and refers to some tuple $x \in \mathcal{T}[R]$. In a number of places we will refer to a set of tuple variables $r_1, \ldots, r_k$, and their associated tables, $R_1, \ldots, R_k$, where $R_i = \text{Dom}[r_1]$.

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

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

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

Definition 6.3: For any query Q over database $\mathcal{D}$, the joins in Q induce a distribution $P_{\mathcal{D}}(Q)$ over the attributes of the tuple variables in Q, $R_1 \times \ldots \times R_k$. The probability that the query is satisfied is:

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

We can also view this joint distribution as generated by an imaginary process, where we independently sample a sequence of tuples $r_1, \ldots, r_k$ from $R_1, \ldots, R_k$, and then select as the values of $A_1, \ldots, A_n$ the values of $r.A_1, \ldots, r.A_n$. (Note that we are not suggesting that the sampling process used to define $P_{\mathcal{D}}$ be carried out in practice. We are merely using it as a way of defining $P_{\mathcal{D}}(Q)$.)

6.2.4 The Universal Foreign-Key Closure

While the expression in Definition 6.3 is computable for any select-join query, we will find it more useful if we can define a unique distribution induced by our database. This will allow us to make useful assertions about the distribution, for example asserting independence statements. To achieve this, we need to restrict our attention to a finite, acyclic collection of foreign-key joins. We begin by introducing an ordering over the tables in the schema:

Definition 6.4: Let $\prec$ be a partial ordering over the tables in our schema. We say that a foreign key $R.F$ with $\text{Dom}[R.F] = S$ is *consistent with* $\prec$ if $S \prec R$. Let $\mathbf{F}$ be a subset of the foreign keys in the database, $\mathbf{F} \subseteq \mathcal{F}(DB)$. A schema $\mathcal{R}$ is *(table) stratified* with respect to $\mathbf{F}$ if there exists a partial ordering $\prec$ such that for any $R.F \in \mathbf{F}$, $R.F$ is consistent with $\prec$. ∎

Often times, the set of foreign keys we consider is the entire set of foreign keys in the database. In this case, we will just say that the schema is table stratified, without referring to the set of foreign keys. We will call a table R a *leaf* in the ordering if there is no other table R' such that $R \prec R'$.

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

Definition 6.5: Let $\mathcal{D}$ be a database with relational schema $\mathcal{R}$ and let $\prec$ be a table stratification of $\mathcal{D}$ with respect to some subset $\mathbf{F}$ of $\mathcal{D}$'s foreign keys. The *universal foreign-key closure* of $\mathcal{D}$ with respect to $\mathbf{F}$ is defined by the query $\mathcal{U}$ we construct below. $\mathcal{T}(\mathcal{U})$ will be the set of tuples variables in our query. Initially, $\mathcal{T}(\mathcal{U})$ has one tuple variable for each of the tables that are leaves in $\prec$. Each tuple variable is initially marked unprocessed.

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

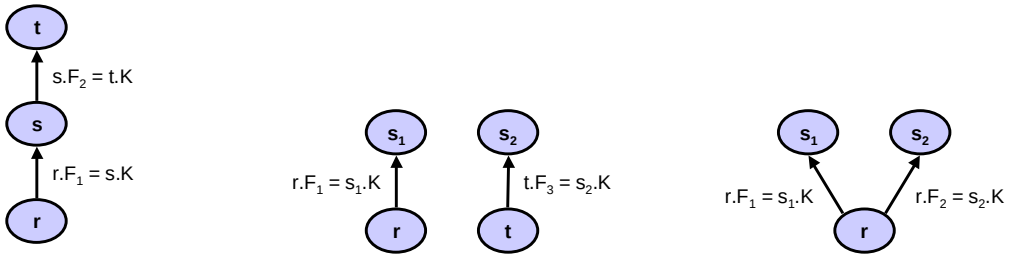
- Let r be an unprocessed tuple variable in $\mathcal{T}(\mathcal{U})$.
- Let $R = \text{Dom}[r]$. For each $F \in \mathcal{F}(R)$, where $R.F$ refers to S, add a new unique tuple variable s to $\mathcal{T}(\mathcal{U})$. This tuple variable is marked unprocessed. We say that s is the tuple variable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

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

As a simple example, suppose we have three tables, R, S and T, and $R.F_1$ is a foreign key into S and $S.F_2$ is a foreign key into T, and our table stratification is: $T \prec S \prec R$. Then the universal foreign-key closure is a query over $R \times S \times T$. Let the tuple variables of $\mathcal{U}$ be r, s and t then, in this case, $\bowtie_{\mathcal{U}} = \{(r.F_1 = s.K)\ \&\ (s.F_2 = t.K)\}$. One way of viewing the universal foreign-key closure is as defining a graph over the tuple variables. There is an edge between tuple variables that are (potentially) connected via a foreign-key join. For a foreign-key join $r.F = s.K$, we will draw the edge from r to s, $r \rightarrow s$. Figure 6.3(a) shows the graph for this example.

On the other hand, if in addition T has a foreign key $T.F_3$ into S, and we use a different set of foreign keys in our table stratification, where $S \prec T \prec R$ with respect to the foreign keys $\mathbf{F} = \{R.F_1, T.F_3\}$, then the universal foreign-key closure of the database is a query over $R \times S \times T \times S$ and the tuple variables are r, s_1, t and s_2; in this case we introduce two new tuple variables for S, one, s_1, associated with the foreign key $r.F_1$ and another, s_2 associated with the foreign key $t.F_3$. Here, $\bowtie_{\mathcal{U}} = \{(r.F_1 = s_1.K)\ \&\ (t.F_3 = s_2.K)\}$. Figure 6.3(b) shows the graph for this example.

As another example, suppose we have two tables R and S, and R has two foreign keys, $R.F_1$ and $R.F_2$, both into S. Our table stratification is $S \prec R$ and the universal



(a) (b) (c)

Figure 6.3: (a) A graph over the tuple variables for the universal foreign-key closure over tables R, S and T, with foreign keys $R.F_1$, $\text{Dom}[R.F_1] = S$ and $S.F_2$ $\text{Dom}[S.F_2] = T$, and table stratification $T \prec S \prec R$. (b) A graph over the tuple variables for the universal foreign-key closure over tables R, S and T, with foreign keys $R.F_1$, $\text{Dom}[R.F_1] = S$ and $T.F_3$ $\text{Dom}[T.F_3] = S$, and table stratification $S \prec T \prec R$. (c) A graph over the tuple variables for the universal foreign-key closure over tables R and S, with foreign keys $R.F_1$, $\text{Dom}[R.F_1] = S$ and $R.F_2$ $\text{Dom}[R.F_2] = S$, and table stratification $S \prec R$.

foreign-key closure is a query over $R \times S \times S$. Let r, s_1, and s_2 be our tuple variables. Then, in this case, $\bowtie_{\mathcal{U}} = \{(r.F_1 = s_1.K)\ \&\ (r.F_2 = s_2.K)\}$. Figure 6.3(c) shows the graph for this example.

Intuitively, the construction of $\mathcal{U}$ can be viewed as creating a collection of inverted trees over the tuple variables, where there is an edge between tuple variables that are (potentially) connected via a foreign-key join. Because a new tuple variable is introduced for each foreign key, and is associated with exactly one foreign key, each tuple variable has exactly one edge pointing to it (but may have many edges emanating from it). Because any graph in which each node has at most one incoming edge is collection of trees, our construction is a forest. We say that tuple t is above s in $\mathcal{U}$, $t > s$, if there is a path from s to t following the foreign-key joins in $\mathcal{U}$. In Figure 6.3(a), t is above s, s is above r. In Figure 6.3(b), s_1 is above r, however s_2 is not above r.

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

- Let $t_1, \ldots, t_j$ be the tuple variables $\mathcal{T}(\mathcal{U})$, and let $\mathcal{A}(\mathcal{U}) = \{A_1, \ldots, A_m\}$ be the attributes of the tuple variables.

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

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

6.3 Statistical Relational Model

Now that we have a well-defined notion of the probability distribution defined by a database, we turn to the task of representing the function in a compact manner. We will use Statistical Relational Models (SRMs) to represent the frequencies in the database. An SRM, like a PRM, has both a relational component to its description and a probabilistic component. The relational component is defined via the relational schema $\mathcal{R}$ of the database. We review the probabilistic component below, using database terminology rather than the object-oriented approach of Chapter 2. Some of the definitions are equivalent, but it will be useful to have the database definitions at hand later when we discuss queries.

SRMs, like PRMs, extend Bayesian networks to the relational setting. They allow us to model correlations not only between attributes of the same tuple, but also between attributes of related tuples in different tables. This is accomplished by allowing, as a parent of an attribute $R.A$, an attribute $S.B$ in another relation S such that R has a foreign key for S. As before, we can also allow dependencies on

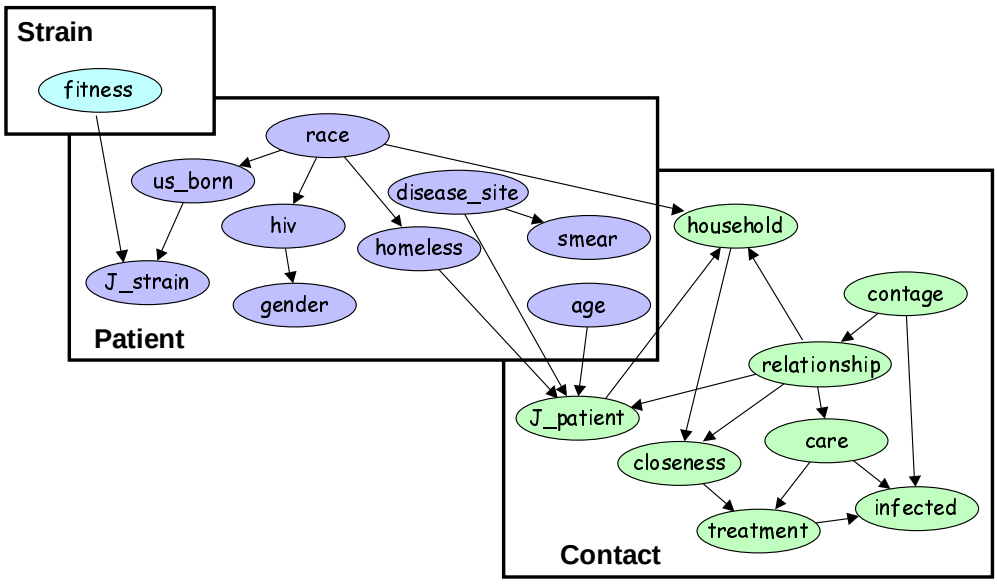


Figure 6.4: An SRM for the Tuberculosis domain.

attributes in relations that are related to R via a longer chain of joins; in this chapter to simplify the notation, we discuss only dependencies along a single pairwise join.

Definition 6.6: A *statistical relational model (SRM)* Ψ for a relational schema $\mathcal{R}$ is a pair $(\mathcal{S}, \theta)$, which specifies a local probability model for each of the following variables:

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

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

- $\mathcal{S}$ specifies a set of parents $\mathrm{Pa}(R.V)$, where each parent has the form $R.B$ or $R.F.B$ where F is a foreign key of R into some table S and B is an attribute of S. If $R.F.B$ is a parent of $R.A$, then $R.J_F$ must also be a parent of $R.A$.
- θ specifies a CPD $P(R.V \mid \mathrm{Pa}(R.V))$. We also require that the CPD of $R.V$ is only defined in cases where each join indicator parent $R.J_F$ is true, $R.J_F = \mathit{true}$.

∎

We refer to the join-indicator variables that are $R.V$'s parents as $\mathrm{Pa}_J(R.V)$ and the attributes that are $R.A$'s parents $\mathrm{Pa}_A(R.V)$. We use $\mathcal{J}(\Psi)$ to refer to all of the join indicator variables in Ψ. An SRM for our TB domain is shown in Figure 6.4. Here, for example, we have that whether the contact is a household member depends on the race of the patient.

This framework allows a probabilistic dependence of an attribute $r.A$ on an attribute $s.B$. This type of dependence only makes sense if s is related to r via some foreign key dependence. Our SRM models a distribution where r and s are chosen independently at random; there is no reason for their attributes to be correlated unless they are somehow related. Hence, we constrain the SRM model to allow $r.A$ to depend on $s.B$ only if $r.F = s.K$. The CPD of $R.A$ is only defined for cases where each $R.J_F \in \mathrm{Pa}_J(R.V)$ is true. In other words, in the CPD tree for $R.A$, the $R.J_F$ are at the top of the tree, and only the branch in which all $R.J_F = \mathit{true}$ is meaningful.

Note that the join indicator variable also has parents and a CPD. Consider the SRM for our TB domain. The join indicator variable $\mathsf{Patient}.J_{\mathit{Strain}}$ has the parents $\mathsf{Patient}.\mathit{US\_born}$ and $\mathsf{Strain}.\mathit{Fitness}$, which indicates whether the strain is unique in the population (and has low fitness) or has appeared in more than one patient (and has moderate to high fitness). There are essentially three cases: for a non-unique strain and a patient that was born outside the U.S., the probability that they will join is around 0.001; for a non-unique strain and a patient born in the U.S. the probability is 0.0029, nearly three times as large; for a unique strain, the probability is 0.0004, regardless of the patient's place of birth. Thus, we are much more likely to have U.S.-born patients joining to non-unique strains than foreign-born ones. (As we saw in Section 3.5, the explanation is that foreign-born patients often immigrate to the U.S. already infected with the disease; such patients typically have a unique strain indigenous to their region. U.S.-born patients, on the other hand, are much more likely to contract the disease by catching it from someone local, and therefore will appear in infection clusters.)

At this point, an SRM looks suspiciously like a PRM. And, at this syntactic level it is almost identical. Our descriptive attributes are allowed the same form of probabilistic dependencies as the descriptive attributes of PRMs. And the join

indicator variables are quite similar in form to the link uncertainty introduced in Chapter 4. However, the models are distinguished by their semantics. A PRM is a model over a particular skeleton and can be used to answer questions about a particular individual. Answering queries (performing inference) in a PRM can be quite expensive. A naive approach requires us to construct the entire unrolled Bayesian network. As we will see in the next section, SRMs answer a very different category of queries, and the algorithms for answering the queries are much more efficient.

6.4 SRM Semantics

We now wish to describe the relationship between an SRM and a database. An SRM describes a set of independence assumptions and the local distributions of attributes given their parents. Intuitively, we say that a database $\mathcal{D}$ is a model of an SRM Ψ if the conditional independence assumptions made in Ψ hold in $\mathcal{D}$ and if the local distributions match the frequencies in $\mathcal{D}$. In order to make this statement precise we will need to introduce several new concepts. The first set of definitions will allow us to define the set of independence assumptions that hold in an SRM.

6.4.1 The Path Dependency Graph

We would like to examine the dependencies that can be represented by an SRM. As in the case of PRMs, we cannot allow cycles in our probabilistic model. Recall that if there is a stratification of the class dependency graph, this guarantees that our instance dependency graph is acyclic. However, in the case of SRMs, we require a stronger form of stratification. We use $\mathbf{F}_\Psi$ to denote the set of foreign keys $R.F$ that are used to define parents in Ψ. We can use the notion of the table stratification of a schema to define the table stratification of an SRM:

Definition 6.7: An SRM Ψ with schema $\mathcal{R}$ is *(table) stratified* if there exists a table stratification $\prec_\Psi$ of $\mathcal{R}$ with respect to $\mathbf{F}_\Psi$. ∎

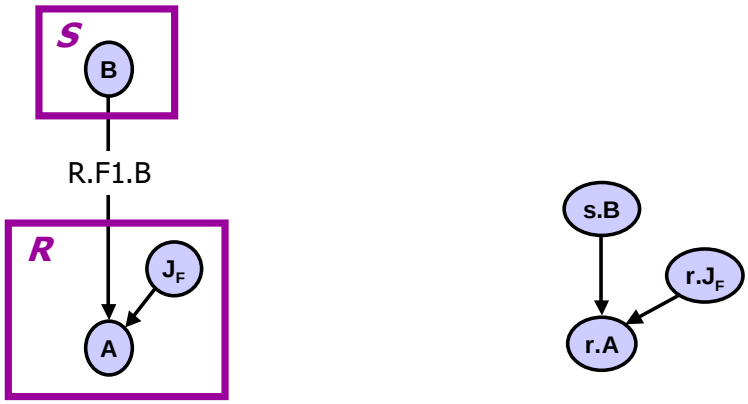
Note that this is a much stronger restriction than the stratification of the class dependency graph proposed for PRMs. The stratification required for PRMs is at the attribute level, whereas the stratification required for SRMs is at the table level. Clearly, if an SRM is table stratified, its class dependency graph is stratified.

Given this restriction, we can now define the set of independence assumptions made in an SRM. Recall that a BN defines the following set of independence assumptions: each random variable is conditionally independent of its non-descendants given its parents. We would like a similar definition of the set of independence assumptions defined by an SRM. In the case of SRMs the task is more difficult because the dependency graph is defined over attributes from different tables. The following set of definitions extend the notion of non-descendants to attributes in an SRM.

First, we will find it useful to introduce the *path dependency graph*. The path dependency graph is a dependency graph defined by the SRM over the attributes and join indicators of the tuples in $\mathcal{U}$.

Definition 6.8: The *path dependency graph* $\mathcal{G}$ for an SRM Ψ with dependency structure $\mathcal{S}$ is defined as follows: For each tuple variable $t \in \mathcal{T}(\mathcal{U})$ where t is a tuple variable for table R,

- For each attribute $R.A \in \mathcal{A}(R)$, and for each parent of $R.A \in \mathrm{Pa}(R.A)$
  - If the parent is of the form $R.B$, add the edge $t.B \rightarrow t.A$.
  - If the parent is of the form $R.F.B$, and s is the tuple variable associated with $R.F$ in $\mathcal{U}$, add the edge $s.B \rightarrow t.A$. Also add the edge for the corresponding join indicator, $t.J_F \rightarrow t.A$.
- Similarly for each join indicator $R.J_F$ of a foreign key of R, and for each parent of $R.A \in \mathrm{Pa}(R.J_F)$:
  - If the parent is of the form $R.B$, add the edge $t.B \rightarrow t.J_F$.
  - If the parent is of the form $R.F'.B$, and s is the tuple variable associated with $R.F'$ in $\mathcal{U}$, add the edge $s.B \rightarrow t.J_F$. If $t.J'_F \neq t.J_F$, also add the edge for the corresponding join indicator, $t.J'_F \rightarrow t.J_F$.



(a) (b)

Figure 6.5: (a) A simple example SRM for two relations, R and S. R has one descriptive attribute, $R.A$, and one foreign key, $R.F$, which refers to S. S has one descriptive attribute $S.B$. $R.A$ depends on $R.F.B$. (b) The path dependency graph for this example.

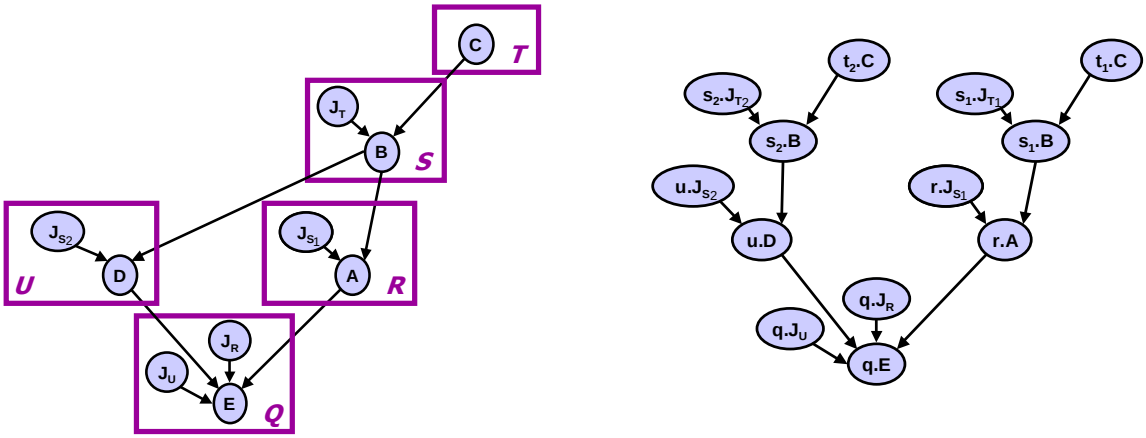
Let $\mathcal{V}(\mathcal{G}) = \{V_1, \ldots, V_p\}$ denote all of the variables in $\mathcal{G}$. ∎

To illustrate, suppose we have the simple SRM shown in Figure 6.5(a), with a schema containing two tables, R and S. R has one descriptive attribute and a foreign key $R.F$ which refers to S. S has one descriptive attribute, $S.B$. The value of $R.A$ depends on $R.F.B$. The path dependency graph is shown in Figure 6.5(b). It has nodes $r.A$, $r.J_F$, and $s.B$ where s is associated with foreign key $r.F$, and edges $s.B \rightarrow r.A$ and, because we use the join $r.F$ in defining this dependence, we have the edge $r.J_F \rightarrow r.A$.

A more complex SRM is shown in Figure 6.6(a). This SRM is over 5 tables: Q, U, R, S and T. The path dependency graph is shown in Figure 6.6(b). Here we see that, due to the construction of the universal foreign-key closure, there have been two tuple variables introduced for both table S and table T.

We introduce the notion of the upward joins of V, $J^*_{\mathcal{U}}(V)$, which are all the joins that occur "above" V in the path dependency graph $\mathcal{G}$. $J^*_{\mathcal{U}}(V)$ is defined as follows:

Definition 6.9: Let $\mathcal{G}$ be the universal foreign-key closure for a schema $\mathcal{R}$. Let $V \in \mathcal{V}(\mathcal{G})$. The upward joins of V are $J^*_{\mathcal{U}}(V) = \{J \mid J \text{ is an ancestor of V in } \mathcal{G}\}$. ∎



(a) (b)

Figure 6.6: (a) A more complex SRM for five relations, Q, R, S, T, and U. (b) The path dependency graph for this SRM. In this example, the nondescendants of $r.A$ are $T.C$, $S.J_{T_1}$, $Q.J_R$, $Q.J_U$ and $U.D$.

In Figure 6.6(b), the upward joins of $r.A$ are: $J^*_{\mathcal{U}}(r.A) = \{r.J_{S_1}, s_1.J_{T_1}\}$, and the upward joins of $q.E$ are: $J^*_{\mathcal{U}}(q.E) = \{q.J_R, q.J_U, r.J_{S_1}, s_1.J_{T_1}, u.J_{S_2}, s_2.J_{T_2}\}$.

We can extend our definition of upward joins to a set of variables, $\boldsymbol{V} \subseteq \mathcal{V}(\mathcal{G})$, in the obvious way: it is just the union of the upward joins of each $v \in \boldsymbol{V}$.

Next we introduce another definition defined via the path dependency graph, the notion of the *nondescendants* of a variable.

Definition 6.10: Let Ψ be an SRM with path dependency graph $\mathcal{G}$. Let $R.A$ be a node in $\mathcal{G}$ with parents $\mathrm{Pa}_{\mathcal{G}}(R.A)$. The non-descendants of $R.A$, $\mathrm{nondescendants}_{\mathcal{G}}(R.A)$, are the set of attributes that are not descendants of $R.A \in \mathcal{G}$, and are neither $R.A$ itself or among $\mathrm{Pa}(R.A)$:

$$\mathcal{V}(\mathcal{G}) - \{R.A\} - \mathrm{Pa}_{\mathcal{G}}(R.A) - \mathrm{descendants}_{\mathcal{G}}(R.A). \quad \blacksquare$$

In the earlier example shown in Figure 6.5(b), $r.A$ has no non-descendants and $r.J_F$ has non-descendant $s.B$. For the path dependency graph shown in Figure 6.6(b), the non-descendants of $r.A$ are $t_1.C$, $s_1.J_{T_1}$, $q.J_R$, $q.J_U$, $u.D$, $u.J_{S_2}$, $s_2.B$, $s_2.J_{T_2}$ and $t_2.C$.

$\boldsymbol{V} = \boldsymbol{v}$	legal?
$\{t_1.C = c\}$	yes
$\{s_1.B = b, s_1.J_{T_1} = \mathit{false}, t_1.C = c\}$	no
$\{s_1.J_{T_1} = \mathit{false}, t_1.C = c\}$	yes
$\{s_1.B = b, s_1.J_{T_1} = \mathit{true}, t_1.C = c\}$	yes
$\{r.J_{S_1} = \mathit{true}, s_1.J_{T_1} = \mathit{true}\}$	yes
$\{r.J_{S_1} = \mathit{true}, s_1.J_{T_1} = \mathit{false}\}$	no
$\{r.J_{S_1} = \mathit{false}, s_1.J_{T_1} = \mathit{true}\}$	yes
$\{q.E = e, q.J_R = \mathit{true}, q.J_U = \mathit{true}, r.J_{S_1} = \mathit{true}, s_1.J_{T_1} = \mathit{true}, u.J_{s_2} = \mathit{true}, s_2.J_{T_2} = \mathit{true}\}$	yes

Table 6.1: This table shows several events and along with their categorization as legal events for the path dependency graph of Figure 6.6(b).

Like $P_{\mathcal{U}}$, the SRM defines a probability distribution over the random variables $\mathcal{V}(\mathcal{G})$. However, we limit the events we consider to ones where an appropriate set of the join indicator variables are true. In order to make this more precise, we need to introduce a bit more machinery.

An event, which is assignment of values to some subset of the random variables, $\boldsymbol{V} \subseteq \mathcal{V}(\mathcal{G})$ is *legal* if, for any variable with a value assignment, all of the upward joins of the variable are true and included in $\boldsymbol{V}$.

Definition 6.11: Let $\mathcal{G}$ be the path dependecny graph for a schema $\mathcal{R}$. Let $\boldsymbol{V} \subseteq \mathcal{V}(\mathcal{G})$. An assignment of values to the random variables of $\boldsymbol{V} = \{v_1, \ldots, v_k\}$ is *legal* iff for each $V \in \boldsymbol{V}$, $J^*_{\mathcal{U}}(V) \subseteq \boldsymbol{V}$, and for $V_i \in J^*_{\mathcal{U}}(V)$, $v_i = \mathit{true}$. ∎

Intuitively, in a legal assignment, all of the upward joins of the variables must be included and *true*, however join indicators that are on the 'fringe', i.e., leaves of this set of variables, may be *false*.

Table 6.1 shows several examples of events for the SRM from Figure 6.6, and their categorization as either legal or illegal events.

Now we are ready to describe the distribution defined by an SRM:

Definition 6.12: Let $\Psi = (\mathcal{S}, \theta)$ be an SRM over $\mathcal{D}$, let $\mathcal{G}$ be the path dependency graph for Ψ. Let $\mathcal{V}(\mathcal{G}) = \{V_1, \ldots, V_p\}$ be the nodes in the path dependency graph.

Then Ψ defines the following distribution:

$$P_\Psi(V_1, \ldots, V_p) = \prod_{V_i \in \mathcal{V}(\mathcal{G})} P_\theta(V_i \mid \mathrm{Pa}_\mathcal{G}(V_i), J^*_\mathcal{U}(V_i) = \mathbf{T}) \blacksquare$$

Note that this distribution is defined *only* for legal events.

We begin by proving that this is in fact a coherent probability distribution.

Theorem 6.13: *Let Ψ be an SRM with a table-stratified schema R. Let $\mathcal{G}$ be the path dependency graph for Ψ. Then P_Ψ is a coherent distribution.*

Proof: In order for P_Ψ to define a coherent distribution, we show that it corresponds to a legal ground Bayesian network over legal events.[2] The ground Bayesian network is constructed by introducing a node for each attribute of the path dependency graph $\mathcal{G}$. The parents of each node are defined by $\mathcal{G}$, and the CPDs are taken from the SRM.

The ground Bayesian network is legal if it is acyclic and if each of the node CPDs are well-defined. Because the relational schema is table stratified, the path dependency graph will be acyclic. Hence the ground BN, which has the same structure as the path dependency graph, is acyclic. Assuming the CPDs in the SRMs are legal, then the CPDs in the BN will be legal. We condition on the upward joins of each attribute being *true*, hence P_Ψ defines a coherent distribution *for legal events.* $\blacksquare$

6.4.2 The Independencies in the Database

Now we return to the distribution defined by our database $P_\mathcal{U}(V_1, \ldots, V_m)$. Now that we have our definition of legal events, we are in the position to state the independence statements that hold in the database.

[2]Note, however, that this ground Bayesian network is quite different from the ground Bayesian networks we have defined in earlier chapters. In this case, the Bayesian network will be over attributes of the tuple variables; our earlier constructions had a node for each attribute of each tuple in each table as defined by the appropriate skeleton. Hence the Bayesian networks constructed in this chapter are *much* smaller than our earlier ground Bayesian networks.

Definition 6.14: Given a database $\mathcal{D}$, a set of foreign keys $\mathbf{F}$ and a table stratification $\prec$, the set of independence statements that hold in the database with respect to the a set of legal events E (which will be defined by the path dependency graph $\mathcal{G}$ for an SRM Ψ) are:

For any subsets $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ of $\mathcal{V}(\mathcal{U})$, if for all assignments of values $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ which are legal events:

$$P_{\mathcal{U}}(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = P_{\mathcal{U}}(\mathbf{x} \mid \mathbf{y})$$

then

$$I_{\mathcal{U}}(\mathbf{X}, \mathbf{Z} \mid \mathbf{Y}). \ \blacksquare$$

We see that our construction of the path dependency graph, which introduces a unique tuple variable for each foreign-key in the schema, places some restrictions on the dependencies we can model. In the example given in Figure 6.3(b), because we introduce a new tuple variable for the each reference to S, we cannot assert that r and t refer to the same s, and hence we cannot model dependence between r and t through s. Consider the case where we have a student and we have the grades for the student and the job offers a student receives. In other words, grades is a table with a foreign-key that refers to the student, and offers is a table with a foreign-key that refers to the student. While clearly there may be a correlation between a student's grades and the salaries of the offers she receives, our model does not allow us to assert the dependence, because in essence, we introduce a new tuple variable for the student who received the grades and a different tuple variable for the student who received the job offers. While we can if fact represent the correlation, for example through an attribute of student such as their intelligence. This is a limitation on the independence assertion that we can make. This is a design decision we have made, and certainly there are other possibilities. Note that in some cases one can sidestep the issue; for example, in the work on universal relations, it is assumed that there is only one foreign-key reference to each table.

6.4.3 Semantics: models

We are now in the position to make a connection between the independence assumptions made by an SRM and the independence statements that hold in our database. If the independence assumptions defined by the SRM hold for the database and the frequencies in the database match the CPDs of the SRM, we say that $\mathcal{D}$ is a *model* of Ψ.

Now we are ready to state the main definition of this section:

Definition 6.15: A database $\mathcal{D}$ with universal foreign-key closure $\mathcal{U}$ stratified with respect to $\prec_\Psi$ is a model of an SRM Ψ, $\mathcal{D} \models \Psi$, if they are both over the same table-stratified schema $\mathcal{R}$, and in the path dependency graph $\mathcal{G}$, for each node V in $\mathcal{V}(\mathcal{G})$:

- $I_\mathcal{U}(V, \text{nondescendants}_\mathcal{G}(V) \mid \text{Pa}_\mathcal{G}(V), J^*_\mathcal{U}(V) = \mathbf{T})$;
- $P_\mathcal{U}(V \mid \text{Pa}_\mathcal{G}(V), J^*_\mathcal{U}(V) = \mathbf{T}) = P_\Psi(V \mid \text{Pa}_\mathcal{G}(V), J^*_\mathcal{U}(V) = \mathbf{T})$. ∎

And, we are now ready to prove the main theorem of this chapter:

Theorem 6.16: *Let $\mathcal{D}$ be a database over schema $\mathcal{R}$ and Ψ be an SRM over the same schema with path dependency graph $\mathcal{G}$. Suppose $\mathcal{D} \models \Psi$. Then*

$$P_\mathcal{U}(V_1, \ldots, V_p) = P_\Psi(V_1, \ldots, V_p)$$

for legal events.

Proof: By definition

$$P_\Psi(V_1, \ldots, V_p) = \prod_{V_i \in \mathcal{V}\mathcal{G}} P_\theta(V_i \mid \text{Pa}_\mathcal{G}(V_i), J^*_\mathcal{U}(A_i) = \mathbf{T})$$

Let $V_1, \ldots, V_p$ be a topological ordering of the attributes and join indicators of $\mathcal{G}$. Consider the same ordering of the attributes of $\mathcal{U}$. We can use the chain rule to write

$P_{\mathcal{D}}(\mathcal{U})$ as follows:

$$P_{\mathcal{D}}(\mathcal{U}) = \prod_{i=1}^{p} P_{\mathcal{D}}(V_i \mid V_{i-1}, \ldots, V_1).$$

We can check that for any topological ordering of the attributes $V_1, \ldots, V_p$, if $v_1, \ldots, v_i$ is a legal event, then so is the prefix $v_1, \ldots, v_{i-1}$. Therefore in the above product, we are conditioning on legal events. Because $\mathcal{D} \models \Psi$, the same independences hold in $P_{\mathcal{U}}$ and P_{Ψ}, and we can rewrite the above as:

$$P_{\mathcal{U}}(V_1, \ldots, V_p) = \prod_{i=1}^{p} P_{\mathcal{D}}(V_i \mid \mathrm{Pa}_{\mathcal{G}}(V_i), J^*_{\mathcal{U}}(V_i) = \mathbf{T})$$

Because the parameters in the SRM match the frequencies in the database, $P_{\mathcal{D}}(V_i \mid \mathrm{Pa}_{\mathcal{G}}(V_i), J^*_{\mathcal{U}}(V_i) = \mathbf{T}) = P_{\theta}(V_i \mid \mathrm{Pa}_{\mathcal{G}}(V_i), J^*_{\mathcal{U}}(V_i) = \mathbf{T})$, therefore the probability functions $P_{\mathcal{D}}(\mathcal{U})$ and $P_{\Psi}(\mathcal{G})$ are equal. ∎

Also note that for any SRM Ψ, there will be an infinite number of databases that are models of Ψ. And, for any database, in general there will be more than one SRM that it models. However, in the case of functional dependencies, even this SRM is not unique.

This may all sound rather negative. In fact, even finding independencies that hold perfectly for all legal assignments of values to attributes and join indicators in the database is rather difficult. However, as we will see later when we learn an SRM from a database, instead we will be finding an SRM that is a good approximation for the independencies that hold in the database.

6.4.4 Inference: Answering Queries

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

6.4.4.1 Queries

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

Definition 6.17: Given a database $\mathcal{D}$ with universal foreign-key closure $\mathcal{U}$, let $T \subseteq \mathcal{T}(\mathcal{U})$. T is *closed* with respect to $\mathcal{U}$ if for all $t \in T$, if for each foreign key $t.F$ of T with associated tuple variable s, $s \in T$. Let $\mathcal{J}(T) = \{t.F \bowtie s.K \mid t \in T$ and s is the associated tuple variable for t.F$\}$. ∎

Definition 6.18: Given a database $\mathcal{D}$ with universal foreign-key closure $\mathcal{U}$, Q is a *inverted-tree foreign-key-join* query over tuple variables $T = \{t_1, \ldots, t_k\}$, if T is closed with respect to $\mathcal{U}$, $\bowtie_Q = \mathcal{J}(T)$, and we select on some subset of the attributes $\mathcal{A}(T)$, $\sigma_Q = \{A_i = a_i \mid A_i \in \mathcal{A}(T)\}$:

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

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

6.4.4.2 Query Evaluation Bayesian Network

Definition 6.19: Let $\Psi = (\mathcal{S}, \theta)$ be an SRM over $\mathcal{D}$, and let Q be a legal query. We define the *query-evaluation* Bayesian network $\Upsilon[Q]$ to be a BN as follows:

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

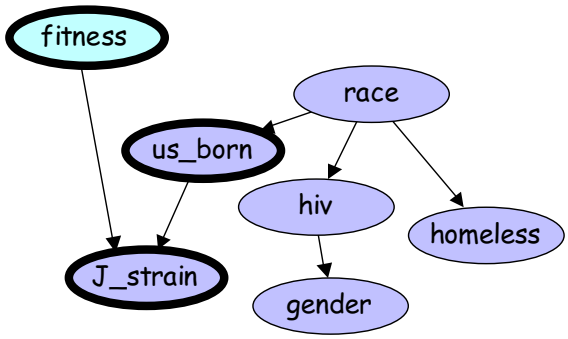


Figure 6.7: A query-evaluation BN for TB domain and the keyjoin query $s.Fitness = high$ & $p.Strain = s.Strain\text{-}ID$ & $p.US\_born = true$. The attributes mentioned in the query are shown with a heavy outline. (The disconnected nodes have been removed from the graph.)

In our simple example in Figure 6.5(a), the query evaluation BN is identical to the path dependency graph in Figure 6.5(b). As another example, Figure 6.7 shows the query evaluation BN for the following foreign-key closed keyjoin query over our TB SRM: $s.Fitness = high$ & $p.Strain = s.Strain\text{-}ID$ & $p.US\_born = true$.

The query evaluation Bayesian network defines the following distribution:

Definition 6.20: Let $\Psi = (\mathcal{S}, \theta)$ be an SRM over $\mathcal{D}$, let Q be a legal query and let $\Upsilon[Q]$ be the query evaluation Bayesian network for Q. Then $\Upsilon[Q]$ defines the following distribution:

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

where $\mathrm{Pa}(A_i)$ are defined by $\Upsilon[Q]$. ∎

Lemma 6.21: *Let Ψ be an SRM with a table-stratified schema $\mathcal{R}$. Let Q be a legal*

query. Then

$$P_\Psi(Q) = \Upsilon[Q]$$

Proof: As we saw, if Q is a legal query, it is an upwardly closed set of fragments of the universal foreign-key closure. Thus, $\Upsilon[Q]$ is a an upwardly closed fragment of the path dependency graph $\mathcal{G}$. We can construct $\Upsilon[Q]$ from the ground Bayesian network for $P_\Psi(Q)$ by simply applying barren node elimination to remove descendants that are not part of the query. ∎

Now we are ready to prove the main theorem of this section:

Theorem 6.22: *Let database $\mathcal{D} \models \Psi$. Let Q be a legal query and let $\Upsilon[Q]$ be the query evaluation Bayesian network for the query.*

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

Proof: This is a straightforward consequence of Theorem 6.16 and Lemma 6.21 ∎

6.4.5 More Expressive Queries

In the previous section, we showed how to compute the answer to queries that are inverted foreign-key queries. It is straightforward to extend the results beyond the universal foreign-key closure to products of unconnected inverted foreign-key queries over non-interacting copies of $\mathcal{U}$; the solution is just the cross product of the independent components.

More interestingly, we can also use our methods to answer queries that are not inverted foreign-key queries. A simple example of a query that is not of this form is $R.F = S.K$ & $T.F = S.K$. As we saw in Figure 6.3(b), our SRM makes the assumption that the tuple variable that R joins with is independent from the tuple variable that T joins with. But, we can still use our SRM to answer this query, we are just no longer guaranteed to compute the correct answer, we will only be computing

an approximation. In effect, we are making an incorrect independence assumption. While on the surface this may not seem terribly useful, the SRM is still likely to produce better estimates than other methods commonly used for estimating queries results, which make even stronger (potentially incorrect) independence assumptions (for example assuming independence of all attributes).

6.5 Learning SRMs

The previous sections showed how to answer queries once we have an SRM that captures the significant statistical correlations in the data distribution. This section addresses the question of how to construct such a model automatically from the relational database. The learning algorithm is essentially the same as the learning algorithm described in Chapter 3. Here we highlight the important differences in the algorithms.

As before, the input to the construction algorithm consists of two parts: a relational schema, that specifies the basic vocabulary in the domain — the set of tables, the attributes associated with each of the tables, and the possible keyjoins between tuples; and the database itself, which specifies the actual tuples contained in each table. Here, because our goal is summarization, we wish to ensure the the compactness of our model. We accomplish this by imposing a storage limitation on the models, so that our task is to find the best SRM that satisfies our space constraints.

In the construction algorithm, our goal is to find an SRM $(\mathcal{S}, \theta)$ that best represents the dependencies in the data. There is an important difference in this goal in comparison with our optimization criteria in Section 3.3.2. For a PRM, we were interested in constructing a model that trades off fit to data with model complexity. This tradeoff allows us to avoid fitting the training data too closely, which would reduce our ability to predict unseen data. However, in the case of an SRM, our goal is very different: We do not want to generalize to new data, but only to summarize the patterns in the existing data. This difference in focus is what motivates our choice of scoring function.

6.5.1 Scoring Criterion

To provide a formal definition of model quality, we make use of basic concepts from information theory [Cover and Thomas, 1991]. The quality of a model can be measured by the extent to which it summarizes the data. In other words, if we had the model, how many bits would be required, using an optimal encoding, to represent the data. The more informative the model, the fewer bits are required to encode the data.

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

$$l(\theta, \mathcal{S} \mid \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{S}, \theta) \tag{6.1}$$

As before, we can therefore formulate the model construction task as that of finding the model that has maximum log-likelihood given the data.

6.5.2 Parameter Estimation

As we discussed in Section 3.2.1, the highest likelihood parameterization for a given structure $\mathcal{S}$ is the one that precisely matches the frequencies in the data. We have seen how to do maximum likelihood parameter estimation for descriptive attributes, so here we discuss only the extensions required for join indicator variables.

The computation of the CPD for a join indicator variable J_F requires that we compute the probability that a random tuple r from R and a random tuple s from S will satisfy $r.F = s.K$. The probability of the join event can depend on values of attributes in r and s, e.g., on the value of $r.A$ and $s.B$. In our TB domain, the join indicator between Patient and Strain depends on *US_born* within the Patient table and on *Fitness* within the Strain table. To compute the sufficient statistics for $P(J_F \mid r.A, s.B)$, we need to compute the total number of cases where $r.A = a$, $s.B = b$, and then the number within those where $r.F = s.K$. Fortunately, this

computation is also easy. The first is simply $C_{\mathcal{D}}[R.A = a] \cdot C_{\mathcal{D}}[S.B = b]$. The latter is $C_{\mathcal{D}}[R.A = a, S.B = b, R.F = S.K]$, which can be computed by joining the two tables and then doing a count and group-by query. The cost of this operation (assuming an appropriate index structure) is linear in the number of tuples in R and in S.

6.5.3 Structure Selection

Our second task is the structure selection task: finding the dependency structure that achieves the highest log-likelihood score. Consider a particular structure $\mathcal{S}$. We saw in Section 3.2.1, the optimal choice (in terms of likelihood) for parameterizing $\mathcal{S}$. We use $\hat{\theta_{\mathcal{S}}}$ to denote this set of parameters. Let $P_{\mathcal{D}}$ be the distribution in the database, as above. We can reformulate the log-likelihood score in terms of mutual information:

$$l(\mathcal{S}, \hat{\theta_{\mathcal{S}}} \mid \mathcal{D}) = \left[\sum_i |R_i| \sum_{A \in \mathcal{A}(R_i)} \mathbb{M}I_{P_{\mathcal{D}}}(R_i.A; \mathrm{Pa}(R_i.A)) \right] + C \tag{6.2}$$

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

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

6.5.3.1 Search algorithm

Now we must provide a search algorithm for finding a high-scoring hypothesis in our space. As before, the search algorithm is greedy hill-climbing search, using random steps to escape local maxima. We maintain our current candidate structure $\mathcal{S}$ and iteratively improve it. At each iteration, we consider a set of simple local transformations to that structure. For each resulting candidate successor $\mathcal{S}'$, we check that it satisfies our constraints, and select the best one. We restrict attention to simple transformations such as adding, deleting or reversing an edge, and adding or deleting a split in a CPD tree. This process continues until none of the possible successor structures $\mathcal{S}'$ have a higher score than $\mathcal{S}$. At this point, the algorithm can take some number of random steps, and then resume the hill-climbing process. After some number of iterations of this form, the algorithm halts and outputs the best structure discovered during the entire process.

However, here we are using a different scoring criterion and we need to consider how that should alter our search strategy. In our greedy search, we need to consider how to choose among the possible successor structures $\mathcal{S}'$ of a given structure $\mathcal{S}$. The most obvious approach is to simply choose the structure $\mathcal{S}'$ that provides the largest improvement in score, i.e., that maximizes $\Delta_l(\mathcal{S}', \mathcal{S}) = \text{score}_l(\mathcal{S}' : \mathcal{D}) - \text{score}_l(\mathcal{S} : \mathcal{D})$. However, this approach is very shortsighted, as it ignores the cost of the transformation in terms of increasing the size of the structure relative to the storage constraint we have for the model. We now present two approaches that address this concern.

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

normalized by the additional space requirement:

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

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

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

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

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

All three approaches involve the computation of $\Delta_l(\mathcal{S}', \mathcal{S})$. Eq. (6.2) then provides a key insight for improving the efficiency of this computation. As we saw, the score decomposes into a sum, each of which is associated only with a node and its parents in the structure. Thus, if we modify the parent set or the CPD of only a single attribute, the terms in the score corresponding to other attributes remain unchanged [Heckerman, 1998]. Thus, to compute the score corresponding to a slightly modified structure, we need only recompute the local score for the one attribute whose dependency model has changed. Furthermore, after taking a step in the search, most of the work from the previous iteration can be reused.

The example shown earlier, Figure 6.4, shows an SRM that was constructed using this algorithm for the TB domain.

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

6.6 SRMs for Selectivity Estimation

SRMs are useful for answering a wide range of queries. Like PRMs, they are useful for data mining and exploratory data analysis. SRMs answer a different category of queries from PRMs, statistical queries, rather than queries about particular individuals. SRMs are also useful as a compact representation for a particular database. Here we show how this latter capability can be used for a fundamental task in database processing: estimating the result size of a query, or *selectivity estimation*.

Accurate estimates of the result size of queries are crucial to several query processing components of a database management system (DBMS). Cost-based query optimizers use intermediate result size estimates to choose the optimal query execution plan. Query profilers provide feedback to a DBMS user during the query design phase by predicting resource consumption and distribution of query results. Precise selectivity estimates also allow efficient load balancing for parallel join on multiprocessor systems. Selectivity estimates can also be used to approximately answer counting (aggregation) queries. This is an area that is currently receiving a lot of research attention; see [Garofalakis and Gibbons, 2001] for an excellent tutorial on approximate query answering.

6.6.1 Current Approaches

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

The first common assumption is the *attribute value independence assumption*, under which the distributions of individual attributes are independent of each other and the joint distribution is the product of single-attribute distributions. However,

real data often contain strong correlations between attributes that violate this assumption, leading to very inaccurate approximations. For example a census database might contain highly correlated attributes such as *Income* and *Home-Owner*. The attribute value independence assumption would lead to an overestimate of the result size of a query that asks for low-income home-owners.

A second common assumption is the *join uniformity* assumption, which states that a tuple from one relation is equally likely to join with any tuple from the second relation. Again, there are many situations in which this assumption is violated. For example, assume that our census database has a second table for online purchases. High-income individuals typically make more online purchases than average. Therefore, a tuple in the purchases table is more likely to join with a tuple of a high-income individual, thereby violating the join uniformity assumption. If we consider a query for purchases by high-income individuals, an estimation procedure that makes the join uniformity assumption is likely to substantially underestimate the query size. Section 6.1 gave a similar example in the TB domain.

To relax these assumptions, we need a more refined approach, that takes into consideration the *joint distribution* over multiple attributes, rather than the distributions over the attributes in isolation. Several approaches to joint distribution approximation, also referred to as *data reduction*, have been proposed recently; see [Barbará et al., 1997] for an excellent summary of this area.

One simple approach for approximating the query size is via random sampling. For select selectivity estimation over a single table, a set of samples is generated, and then the query result size is estimated by computing the actual query result size relative to the sampled data. However, the amount of data required for accurate estimation can be quite large. For join selectivity estimation, we can randomly sample the two tables, and compute their join. This approach is flawed in several ways [Acharya et al., 1999], and some work has been devoted to alternative approaches that generate samples in a more targeted way [Lipton et al., 1990].

More recently, several approaches have been proposed that attempt to capture

the joint distribution over attributes more directly. The earliest of these is the multi-dimensional histogram approach [Muralikrishna and Dewitt, 1988, Poosala and Ioannidis, 1997]. They provide an extensive exploration of the taxonomy of methods for constructing multidimensional histograms and study the effectiveness of different techniques. They also propose an approach based on *singular value decomposition*, applicable only in the two-dimensional case. A newer approach is the use of wavelets to approximate the underlying joint distribution [Matias et al., 1998, Vitter and Wang, 1999, Chakrabarti et al., 2001].

Contemporaneously with the development of the work presented here, there has been recent work applying graphical models to the problems of approximate query answering and database compression. In an approach closely related to our work, Markov models have been used to approximate the joint distribution over attributes in a single table [Deshpande et al., 2001]. The learned models are then used to compute approximate answers to OLAP queries. There has also been recent work on using a learned Bayesian network to guide the construction of compact models of large tables [Babu et al., 2001]. Compact models are achieved by building CART trees for certain attributes; the choice of columns is guided by the structure of the learned Bayesian network. This approach differs from ours in that in the end it does not attempt to capture the full joint distribution (although as an intermediate step, it *does* construct a model of the joint distribute), but focuses on building good predictors for single attributes. However, the most important distinction is the support in our approach for queries over multiple tables.

6.6.2 Our Approach

In this section, we provide a framework for using probabilistic graphical models to estimate selectivity of queries in a relational database. SRMs can be used to represent the interactions between attributes in a single table, providing high-quality estimates of the joint distribution over the attributes in that table. Furthermore, SRMs allow us to represent skew in the join probabilities between tables, as well as correlations between attributes of tuples joined via a foreign key. They thereby allow us to estimate

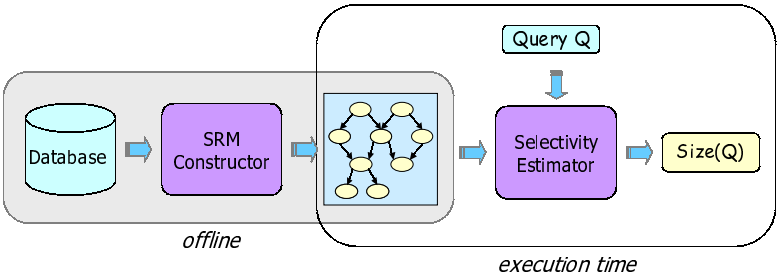


Figure 6.8: A high-level description of the selectivity estimation process.

selectivity of queries involving both selects and joins over multiple tables.

Our approach has several important advantages. First, it provides a uniform framework for select selectivity estimation and foreign-key join selectivity estimation, introducing a systematic method for estimating the size of queries involving both operators. Second, our approach is not limited to answering a small set of predetermined queries; a single statistical model can be used to effectively estimate the sizes of any (select foreign-key join) query, over any set of tables and attributes in the database.

Figure 6.8 shows the high-level architecture for our algorithm. Like most selectivity estimation algorithms, our algorithm consists of two phases. The *offline* phase, in which the SRM is constructed from the database. This process is automatic, based solely on the data and the space allocated to the statistical model. The second, *online* phase, is the selectivity estimation for a particular query. The selectivity estimator receives as input a query and an SRM, and outputs an estimate for the result size of the query. Note that the same SRM is used to estimate the size of a query over any subset of the attributes in the database; we are not required to have prior information about the query workload. While we will be describing a batch learning algorithm, standard methods for handling updates to the database can incorporated. We discuss this issue further in Section 6.6.6.

The next subsection describes selectivity estimation for select operations over a single table. We show how SRMs can be used to approximate the joint distribution over the entire set of attributes in the table. In the following section, Section 6.6.4,

we move to the more complex case of queries over multiple tables. Finally, in Section 6.6.5, we provide empirical validation of our approach, and compare it to some of the most common existing approaches. We present experiments over several real-world domains, showing that our approach provides much higher accuracy (in a given amount of space) than previous approaches, at a very reasonable computational cost, both offline and online.

6.6.3 Estimation for Queries over Single Tables

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

Let R be some table and let $\mathcal{A}(R)$ be the descriptive (non-key) attributes of R, $A_1, \ldots, A_n$. We denote the joint frequency over $A_1, \ldots, A_n$ as $F_{\mathcal{D}}(A_1, \ldots, A_n)$. As before, it is convenient to deal with the normalized frequency distribution, $P_{\mathcal{D}}(A_1, \ldots, A_n)$, where:

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

Now, consider a query Q over a set of attributes $A_1, \ldots, A_k \subseteq \mathcal{A}(R)$, which is a conjunction of selections of the form $A_i = v_i$. The size of the result of the query Q is:

$$size_Q[\mathcal{D}] = F_{\mathcal{D}}(Q) = |R| \cdot P_{\mathcal{D}}(Q), \tag{6.3}$$

where $F_{\mathcal{D}}(Q)$ is the number of tuples satisfying Q and $P_{\mathcal{D}}(Q)$ is the probability, relative to $\mathcal{D}$, of the event $\mathcal{I}_Q$.

An SRM (or in this case, a Bayesian network, since we have only one table) is a compact representation of a full joint distribution. Hence, it implicitly contains the

answer to any query about the probability of any assignment of values to a set of attributes. Thus, if we construct an SRM Ψ that approximates $P_{\mathcal{D}}$, we can easily use it to estimate $P_{\mathcal{D}}(Q)$ for any query Q over R. Assume that our query Q has the form $r.\mathbf{A} = \mathbf{a}$. Then we can compute

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

6.6.4 Join Selectivity Estimation

Next, we consider queries over multiple tables. As above, we restrict attention to databases satisfying *referential integrity* and for now, consider only foreign-key joins in our queries.

Consider two tables R and S such that $R.F$ points to $S.K$. Now, consider any select-keyjoin query Q over R and S: $r.\mathbf{A} = \mathbf{a}, s.\mathbf{B} = \mathbf{b}, r.F = s.K$. It is easy to see that the size of the result of Q is:

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

In other words, we can estimate the size of any query of this form using the joint distribution $P_{\mathcal{D}}$ defined in Section 6.2.1.

Making use of our earlier theorem, Theorem 6.22, we can estimate this joint distribution using an SRM. Let Q be a keyjoin query, and let Q^+ be its foreign-key closure. Let $r_1, \ldots, r_k$ be the tuple variables in Q^+. Let $P_{\mathcal{D}}(r_1, \ldots, r_k)$ be the distribution obtained by sampling each tuple $r_1, \ldots, r_k$ independently. Then for any query Q' which extends Q^+, the SRM allows us to approximate $P_{\mathcal{D}}(\mathcal{I}_{Q'})$, precisely the quantity required for estimating the query selectivity. We can compute the SRM estimate using the query-evaluation Bayesian network $\Upsilon[Q]$ defined in Definition 6.19.

Continuing with our previous example, if query Q has the form $r.\mathbf{A} = \mathbf{a}$ & $s.\mathbf{B} = \mathbf{b}$ & $r.F = s.K$, then size of the result of Q is:

$$size_Q[\mathcal{D}] = F_{\mathcal{D}}(Q)$$

$$
\begin{aligned}
&= |R| \cdot |S| \cdot P_{\mathcal{D}}(\mathbf{A} = \mathbf{a}, \mathbf{B} = \mathbf{b}, J_F = true) \\
&\approx |R| \cdot |S| \cdot P_{\Psi}(\mathbf{A} = \mathbf{a}, \mathbf{B} = \mathbf{b}, J_F = true) \\
&= |R| \cdot |S| \cdot P_{\Upsilon[(\mathbf{A}=\mathbf{a}, \mathbf{B}=\mathbf{b}, J_F = true)]}
\end{aligned}
$$

The result follows from Theorem 6.22 and referential integrity. It can easily be extended to queries involving more than two tables.

6.6.5 Experimental Results

In this section, we present experimental results for a variety of real-world data, including: a census dataset [U.S., Census Bureau, 1992-93]; FIN, a subset of the database of financial data used in the 1999 European KDD Cup [Berka, 1999]; and TB, the database of tuberculosis patients in San Francisco that we have already seen in Section 3.5. We begin by comparing the accuracy of our methods with the existing techniques on small select queries over a single relation. We next evaluate the accuracy of our methods on models that support arbitrary select queries over a single relation. We then consider more complex, select-join queries over several relations. Finally, we discuss the running time for construction and estimation for our models.

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

6.6.5.1 Select Queries over Fixed Attributes

We evaluated accuracy for selects over a single relation on a dataset from Census database described above (approximately 150K tuples). In the first set of experiments, we compared our approach to an existing selectivity estimation technique — multidimensional histograms.

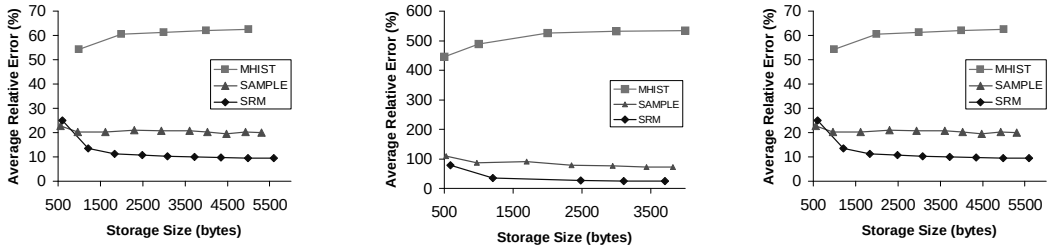


Figure 6.9: Relative error vs. storage size for a query suite over the Census dataset. (a) Two attribute query (Age and Income). The relative error for AVI is 7395. (b) Three attribute query (Age, HoursPerWeek and Income). The relative error for AVI is 865. (c) Four attribute query (Age, Education, HoursPerWeek and Income). The relative error for AVI is 70.19.

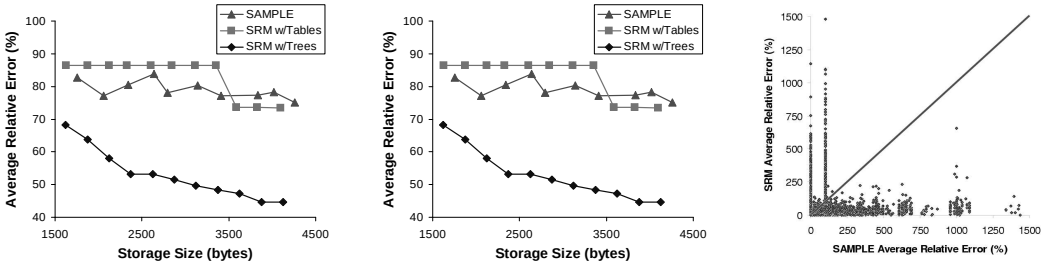
We compared the performance of four algorithms. **AVI** is a simple estimation technique that assumes attribute value independence: for each attribute a one dimensional histogram is maintained. In this domain, the domain size of each attribute is small, so it is feasible to maintain a bucket for each value. This technique is representative of techniques used in existing cost-based query optimizers such as System-R. **MHIST** builds a multidimensional histogram over the attributes, using the V-Optimal(V,A) histogram construction of Poosala and Ioannidis [1997].[4] This technique constructs buckets that minimize the variance in area (frequency $\times$ value) within each bucket. Poosala *et al.* found this method for building histograms to be one of the most successful in experiments over this domain. **SAMPLE** constructs a random sample of the table and estimates the result size of a query from the sample. **SRM** uses our method for query size estimation. Unless stated otherwise, **SRM** uses tree CPDs and the SSN scoring method.

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

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

these methods as we varied the space allocated to each method (with the exception of AVI, where the model size is fixed). Figure 6.9 shows results on Census for three query suites: over two, three, and four attributes. In all cases, SRM outperforms both MHIST and SAMPLE, and all methods significantly outperform AVI. Note that an SRM with tree CPDs over two attributes is just a slightly different representation of a multi-dimensional histogram. Therefore, it is interesting that our approach still dominates MHIST, even in this simple case. As the power of the representations is roughly equivalent here, in both cases we simply use a collection of two-dimensional counts to compute our estimate, the success of SRMs in this setting is due to the different scoring function for evaluating different models, and the associated search algorithm.

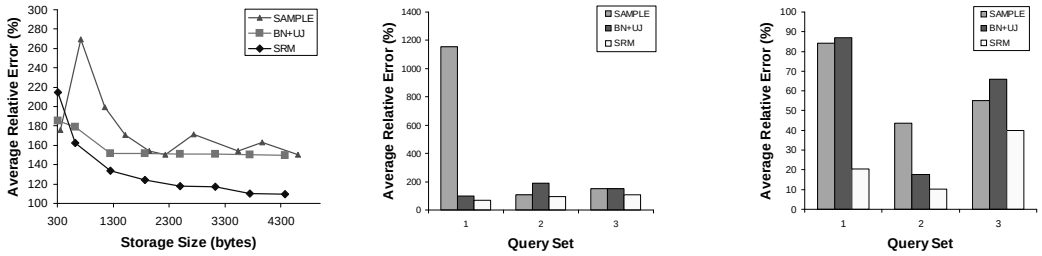
6.6.5.2 Arbitrary Select Queries over a Single Table



(a) (b) (c)

Figure 6.10: Relative error vs. storage size for a query over the Census dataset, with models constructed over 12 attributes. (a) Three attribute query (WorkerClass, Education, and MaritalStatus) (b) Four attribute query (Income, Industry, Age and EmployType). (c) A scatter plot showing the error on individual queries for a three attribute query (Income, Industry, Age) for SAMPLE and SRM (using 9.3K bytes of storage). In the scatter plot, each point represents a query, where the x coordinate is the relative error using SAMPLE and the y coordinate is the relative error using SRM. Thus, points above the diagonal line correspond to queries on which SAMPLE outperforms SRM and points below the diagonal line correspond to queries on which SRM performs better.

In the second set of experiments, we consider a more challenging setting, where a single model is built for the entire table, and then used to evaluate any select query



(a) (b) (c)

Figure 6.11: (a) Relative error vs. storage size for a select-join query over three tables in the TB domain with selection on 3 attributes. (b) Relative error for three query sets on TB (c) Relative error for three query sets on FIN

over that table. In this case, MHIST is no longer applicable, so we compared the accuracy of SRM to SAMPLE, and also compared to SRMs with table CPDs. We built an SRM (which is really a BN, as there is just one table) for the entire set of attributes in the table, and then queried subsets of three and four attributes. (The BN shown in Figure 2.2 is the model that was constructed for this dataset.) Similarly, for SAMPLE, the samples included all 12 attributes.

We tested these approaches on the Census dataset with 12 attributes. The results for two different query suites are shown in Figure 6.10(a) and (b). Although for very small storage size, SAMPLE achieves lower errors, SRMs with tree CPDs dominates as the storage size increases. Note also that tree CPDs consistently outperform table CPDs. The reason is that table CPDs force us to split all bins in the CPD whenever a parent is added, wasting space on making distinctions that might not be necessary. Figure 6.10(c) shows the performance on a third query suite in more detail. The scatter plot compares performance of SAMPLE and SRM for a fixed storage size (9.3K bytes). Here we see that SRM outperforms SAMPLE on the majority of the queries. (The spike in the plot at SAMPLE error 100% corresponds to the large set of query results estimated to be of size 0 by SAMPLE.)

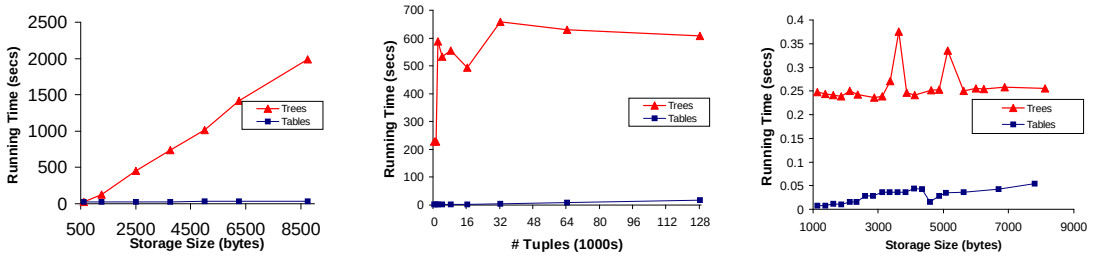
6.6.5.3 Select-Join Queries

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

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

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

Figure 6.11(a) compares the accuracy of the three methods for various storage sizes on a three attribute query in the TB domain. The graph shows both BN+UJ and SRM outperforming SAMPLE for most storage sizes. Figure 6.11(b) compares the accuracy of the three methods for several different query suites on TB, allowing each method 4.4K bytes of storage. Figure 6.11(c) compares the accuracy of the three methods for several different query suites on FIN, allowing 2K bytes of storage for each. These histograms show that SRM always outperforms BN+UJ and SAMPLE.



(a) (b) (c)

Figure 6.12: (a) The construction time for an SRM for Census using tree and table CPDs as a function of model storage space. (b) The construction time for an SRM for Census using tree and table CPDs as a function of data size. (c) The running time for query size estimation as a function of model size.

6.6.5.4 Scoring

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

6.6.5.5 Running Time

Finally, we examine the running time for construction and estimation for our models. These experiments were performed on a 360 MHz Sparc60 workstation running Solaris2.6 with 256MB of internal memory.

We first consider the time required by the offline construction phase, shown in Figure 6.12(a). As we can see, the construction time varies with the amount of storage allocated for the model: Our search algorithm starts with smallest possible model in its search space (all attributes independent of each other), so that more search is required to construct the more complex models that take advantage of the additional space. Note that table CPDs are orders of magnitude easier to construct than tree CPDs; however, as we discussed, they are also substantially less accurate.

The running time for construction also varies with the amount of data in the

database. Figure 6.12(b) shows construction time versus dataset size for tree CPDs and table CPDs for fixed model storage size (3.5K bytes). Note that, for table CPDs, running time grows linearly with the data size. For tree CPDs, running time has high variance and is almost independent of data size, since the running time is dominated by the search for the tree CPD structure once sufficient statistics are collected.

The online estimation phase is, of course, more time-critical than construction, since it is often used in the inner loop of query optimizers. The running time of our estimation technique varies roughly with the storage size of the model, since models that require a lot of space are usually highly interconnected networks which require somewhat longer inference time. The experiments in Figure 6.12(c) illustrate the dependence. The estimation time for both methods is quite reasonable. The estimation time for tree CPDs is significantly higher, but this is using an algorithm that does not fully exploit the tree-structure; we expect that an algorithm that is optimized for inference with tree CPDs [Zhang and Poole, 1999], would be more efficient.

6.6.6 Discussion of Selectivity Estimation Algorithm

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

database.

At the start of this section, we made several assumptions. We now describe how to relax these assumptions. First, we made the assumption that our select operations used only equality predicates. It is straightforward to extend the techniques that we have just described to handle range queries by computing the probability that an assignment of values to the attributes falls in that range. We simply sum over all potential value assignments satisfying the range constraints. While at first glance, this may sound quite expensive, the BN inference algorithms described above can be easily adapted to compute these values without any increase in computational complexity.

The second important assumption that we have been making is that the domains for the attributes are small to moderately sized. We can lift this restriction on our models by using techniques that have been developed for discretization of domain values [Friedman and Goldszmidt, 1996, Monti and Cooper, 1998]. In cases where domain values are not ordinal, we can use feature hierarchies if they are available [desJardins et al., 2000] or we may make use of any of a number of clustering algorithms. Once we have built an SRM over the discretized or abstracted attribute value space, we must now modify our query estimation techniques to provide estimates for queries over the base level values. One method for doing this is to simply compute the selectivity estimate for an abstract query, which maps the base level values to their appropriate discretized or abstracted value, and to then compute an estimate for the base level query by assuming a uniform distribution or the attribute's marginal distribution on the base level values for each abstract value.

There are several important topics that we have not fully addressed. One is the incremental maintenance of the SRM as the database changes. It is straightforward to extend our approach to adapt the parameters of the SRM over time, keeping the structure fixed. To adapt the structure, we can apply a variant of the approach of [Friedman and Goldszmidt, 1997]. We can also keep track of the model score, relearning the structure if the score decreases drastically.

Another important topic that we have not discussed is joins over non-key attributes. In this presentation, our queries use only foreign-key joins. While this

category of queries stands to benefit most from the probabilistic models that we propose, our methods are more general. We can compute estimates for queries that join non-key attributes by summing over the possible values of the joined attributes, and our estimates are likely to be more accurate than methods that do not model any of the dependencies between tuples. However an empirical investigation is required to evaluate our methods on this category of queries.

6.7 Conclusion

In this chapter, we have introduced a second probabilistic model for relational data, a statistical relational model. This model captures the domain frequencies in the database and provides a coherent probabilistic model which has many potential uses. An SRM is a compact statistical model of a database and can be used to answer queries about randomly chosen individuals. We have described the semantics for these models and the circumstances under which an SRM precisely captures the database frequencies. We have described a class of queries that these models are capable of answering. We have an important application of SRMs to the task of query result size estimation.

Chapter 7

Conclusions

In this final chapter, we summarize the contributions of this thesis and discuss a number of promising areas for future work.

7.1 Summary

In this thesis, we have presented a number of different structured probabilistic models together with algorithms for their automatic construction.

We began with our simplest probabilistic relational model, which allows uncertainty over attribute values. These models allow the values of attributes within a class to depend on other attributes in the class, or on attributes in another, related, class. The PRM describes a template for the distribution, and like a collection of first-order rules, is instantiated (or propositionalized) in different ways depending on the universe of discourse provided. In this case, the universe of discourse describes both the objects of each class, and the relationships that hold between objects.

The next probabilistic relational model that we considered allowed for link uncertainty. In addition to describing a probabilistic model for the attributes of objects, this PRM describes a probabilistic model for the existence of links or relationships between objects. Again, the PRM describes a template for a distribution, and is instantiated in different ways depending of the universe of discourse. But in this case, the universe of discourse requires only that the objects be specified. The relationships

between objects must be specified.

The third category of probabilistic relational model that we explored was a model with class hierarchies. We showed how class hierarchies allow the probabilistic model to be refined or specialized where necessary, and at the same time allow the probabilistic model to be specified at a the most abstract level in the hierarchy which adequately captures the dependencies in the domain. More importantly, we showed how the use of subclassing can allow us to describe dependencies that would not be allowed in a PRM with no class hierarchy mechanism. Finally, we noted how this simple class hierarchy mechanism can, in theory, bridge the gap between a class-based probabilistic model (such as a PRM) and an instance-based probabilistic model (as represented by a BN).

The final model we introduced was the statistical relational model. While in spirit similar to the preceding probabilistic relational models, the important fundamental difference is in the semantics of the models. In PRMs, we always made inferences about particular instances from our universe of discourse. For SRMs, we showed that a general category of queries can be answered efficiently, without resorting to building the full unrolled Bayesian network for the domain.

The learning problem for PRMs and SRMs can be defined as an optimization problem: find the best model, where we score the model according to our optimization criterion. There are at least two criteria that make sense to consider, and the choice depends on the intended use of the learned model. One is a generalization criterion: we are learning a model from an existing collection of data, and we intend to apply our model to new, unseen instances; we would like our model to show good generalization performance. The second criterion is a summarization or compression criterion; we may be interested in constructing a compact model that captures our existing collection of data as precisely as possible subject to certain storage restrictions. In this thesis, we considered learning PRMs subject to optimizing their generalization performance and we considered learning SRMs subject to optimizing their use of the allocated space, but it certainly makes sense to consider the other two combinations.

The first learning algorithm we presented, for PRMs with attribute uncertainty,

builds on methods for learning Bayesian networks. Regardless of the optimization criteria, learning the optimal probabilistic relational model from a database is NP-hard. So, as is typical in the face of most NP-hard problems, we resort to a heuristic search through the space of models. The two major components of the algorithm are the methods for doing parameter estimation and methods for doing model selection. We showed how these are done for PRMs with attribute uncertainty. For each extension to our model, we extended the basic learning algorithm to accommodate the changes in the model.

7.2 Related Work

The two key characteristics of our approach are that it is applicable to relational data and that it builds a statistical model. We have already mentioned the commonalities in our approach to model construction and to traditional approaches to building statistical models in Chapter 1. But we haven't discussed other approaches to learning from relational data. Here we provide some background on the learning from relational data.

7.2.1 History

Early work on machine learning often focused on learning relational concepts. This was typically done by learning a logical concept definition. Methods were typically ad hoc and were sensitive to noise; they were often applied to 'toy' domains.

One of the earliest relational learning systems was Winston's arch learning system [Winston, 1975]. This system was given a sequence of training instances label as positive and negative examples of arches. The system maintains has current hypothesis. The hypothesis is relational and represented as a semantic network. When a new example is presented, the system makes a prediction using the current hypothesis. If the prediction is correct, no changes are made to the hypothesis. If it is incorrect, then the set of differences between the current hypothesis and the example are identified. If the example was a positive instance, the differences are used to generalize

the concept; on the other hand if the example was a negative instance, they are used to specialize the concept. Following this there were a number of interesting relational learning systems [Dietterich and Michalski, 1986, Hayes-Roth and McDermott, 1997, Stepp and Michalski, 1985], but all used a similar logic based representation for the concepts.

This branch of machine learning has not received as much attention in recent years. Recently the machine learning (ML) community has mostly focused attention on models that are statistical (or can have a probabilistic interpretation) such as neural networks, decision trees and Bayesian networks, but that apply only to simple non-relational attribute-value feature vectors. The major exception has been the inductive logic programming (ILP) community.

7.2.2 Inductive Logic Programming

The ILP community has concentrated its efforts on learning (deterministic) first-order rules from relational data [Lavrac and Dzeroski, 1994, Muggleton, 1992]. Initially the ILP community focused its attention solely on the task of program synthesis from examples and background knowledge. However, recent research has tackled the discovery of useful rules from larger databases [Dzeroski and Lavrac, 2001]. These rules are usually used for prediction and may have a probabilistic interpretation.

A current weakness of these approaches is that the learning algorithms typically do not interact directly with the database; the data must be extracted from the database before applying the learning algorithms. While in theory this is not an insurmountable obstacle, in practice, having an algorithm that interacts directly with one of the standard database managements systems is an important practical feature.

The ILP community has had successes in a number of application areas including discovery of 2D structural alerts for mutagenicity/carcinogenicity [King et al., 1996], 3D pharmacophore discovery for drug design [Finn et al., 1998] and analysis of chemical databases [Dehaspe et al., 1998].

Recently both the ILP community and the statistical ML community have begun to incorporate aspects of the complementary technology. Many ILP researchers are

developing stochastic and probabilistic representations and algorithms [Muggleton, 2000, Kersting et al., 2000, Cussens, 1999]. In more traditional machine learning circles, researchers who have in the past focused on attribute-value or propositional learning algorithms are exploring methods for incorporating relational information [Craven et al., 1998, Neville and Jensen, 2000, Cohn and Hofmann, 2001]. It is our hope that this trend will continue, and that the work presented in this thesis will provide one bridge connecting relational and statistical learning.

7.2.3 Probabilistic ILP approaches

There are two recent developments within the ILP community that are related to PRMs: the work on stochastic logic programs (SLPs) [Muggleton, 1996, Cussens, 1999] and the work on Bayesian logic programs (BLPs) [Kersting et al., 2000]. The semantics for these two approaches are quite different, with the BLP semantics being the closest to PRMs. An SLP defines a sampling distribution over logic programming proofs; as a consequence, it induces a probability distribution over the possible ground facts for a given predicate. On the other hand, a BLP consists of a set of rules, along with conditional probabilities and a combination rule; following the approach of knowledge-based model construction [Wellman et al., 1992], the BLP essentially specifies a propositional Bayesian network. This approach is very similar to the probabilistic logic programs of [Ngo and Haddawy, 1996, Poole, 1993].

Learning algorithms for these approaches are being developed. Methods for learning SLPs are described in [Muggleton, 2000]. A maximum likelihood approach is taken for parameter estimation for an SLP which is based on maximizing the posterior probability of the program. The task of learning the structure of an SLP is quite different from learning a PRM structure and is based on more traditional ILP approaches. On the other hand, while BLPs are more closely related to PRMs, and methods for learning BLPs have been suggested in [Kersting et al., 2000], learning algorithms have not yet been developed. Methods for learning PRMs may be found to be applicable to learning BLPs.

7.2.4 Relational Graph Analysis

Among the strong motivations for using a relational model is its ability to model dependencies between related instances. Intuitively, we would like to use our information about one object to help us reach conclusions about other, related objects. For example, we should be able to propagate information about the topic of a document p to documents it has links to and documents that link to it. These, in turn, would propagate information to yet other documents.

Recently, several papers have proposed a process along the lines of this "influence propagation" idea. Chakrabarti et al. [1998] describe a relaxation labeling algorithm that makes use of the neighboring link information. The algorithm begins with the labeling given by a text-based classifier constructed from the training set. It then uses the estimated class of neighboring documents to update the distribution of the document being classified. They show that even using small neighborhoods around the test document significantly increases accuracy.

Neville and Jensen [2000] propose a very similar approach. Their *iterative classification* algorithm essentially implements this process exactly. It builds a classifier based on a fully observed relational training set; the classifier uses both base attributes and more relational attributes (e.g., the number of related entities of a given type). It then uses this classifier on a test set where the base attributes are observed, but the class variables are not. Those instances that are classified with high confidence are temporarily labeled with the predicted class; the classification algorithm is then rerun, with the additional information. The process repeats several times. The classification accuracy is shown to improve substantially as the process iterates.

Slattery and Mitchell [2000] propose an iterative algorithm called FOIL-HUBS for the problem of classifying web pages, e.g., as belonging to a university student or not. They note that several pages in the dataset have links to many other pages, most of which were classified as student home pages. Their approach uses recursive predicate rules to identify such a page as a student directory page based on whether the pages it points to are student pages, and conclude that other pages to which it points are also more likely to be student pages. These rules are combined with text-based classifiers in an iterative relaxation scheme. They show that classification accuracy improves

by exploiting the relational structure.

7.3 Future Work

There are a number of promising areas for future work. At the conclusion of each chapter, we mentioned some of the more immediate directions for further study. Rather than repeat those here, instead we will describe three of the more interesting and broad range directions that are suggested by this work: discovery of relational features, generalized probabilistic relational models and approximate query answering using SRMs.

In Chapter 4, we examined two simple models for link uncertainty. A promising yet challenging untapped area of further research is the discovery of relational features, or the discovery of patterns in the graph describing the relations between objects. The most well-studied of these are the notions of hubs and authorities [Kleinberg, 1999], and in fact we made use of these attributes in the experiments in Section 4.7.3 and showed how they improved our classification accuracy. However, there are other relational features that may prove useful as well. There are certain generic graph connectivity characteristics, such as diameter, in-degree, out-degree, path length, and strongly connected components, that may be predictive in certain domains. For example the diameter of the authorship of publications from a given conference (the longest distance between two authors) may be an indication of how inbred the research is or how welcoming the community is open to outside approaches. More generally, we may be interested in discovering common subgraphs. This has been studied, most notably by [Cook and Holder, 2000], however to our knowledge it has not been studied in the context of a coherent generative probabilistic model. This problem is obviously a difficult and computationally challenging problem. Even doing the subgraph matching is NP-complete, so clearly we will need useful domain information or heuristics if we hope to find useful models. Another area of difficulty is ensuring that the models we learn are coherent. We need some way of maintaining the consistency of models with some collection of probabilistic patterns.

Another quite interesting direction is the notion of a generalized probabilistic

relational algebra. By this, we mean a model in which the probabilistic dependencies can be described using arbitrary expressions in relational algebra. In all the models considered in this thesis, we made heavy use of the relational schema to restrict our search for dependencies. We considered only dependence between objects related via a foreign-key join. One could construct arbitrary selection and filtering criteria and have parents that are outcomes to arbitrary queries. For example, the impact of a professor's work could depend on their publications in the past five years that have appeared in top conferences and have been cited at least 5 times:

```
SELECT COUNT(*)
FROM professor, publication, cites
WHERE publication.author = professor.name
        publication.year > (current_year - 5) AND
        top_conference(publication.venue) AND
        cites.cited = publication.title AND
GROUP BY cites.citing
HAVING COUNT(*) > 10
```

Of course this *greatly* complicates both the model search and the coherency maintenance. While there may be some charm in coming up with a fully general framework and devising learning algorithms for them, first one must verify that the learned models are in fact worth the extra overhead. It is our belief that exploring dependencies between objects related via foreign-key joins is generally more likely to result in the discovery of useful dependencies than are dependencies constructed from arbitrary relational queries.

As a final area in which we feel there are a number of compelling open-problems is the use of SRMs as a compact statistical model of a database. There are many basic questions such the sample size required to learn a high-quality SRM, how to incrementally maintain the models, and how to handle large attribute value spaces. There are also some larger questions. One is how to use SRMs to provide approximate answers to range and aggregate queries, Another is the construction of an SRM from a collection of statistics that were gathered while passively watching results for a stream of user queries. Alternatively, the statistics may be provided by an association rule miner. How do we make the best use of these constraints, in constructing our SRM?

Finally, there may be some way in which SRMs can be used for both data integration and possibly schema reformulation.

7.4 Conclusion

Statistical models and relational data are two fundamental concepts in computer science. Each, in its own right, has created a revolution within the field and spawned its own subfields. The relational model, first introduced by Codd [1970], is unarguably the foundation of modern database theory [Date, 1999, Ullman and Widom, 1997]. Its importance today is witnessed by the huge market for database management systems produced by companies such as Oracle, IBM and Microsoft. Statistical models, on the other hand, are a much older innovation. In addition to defining the field of statistics, they have importance in a wide variety of other fields including physics, chemistry, economics and social science. More recently a number of areas of computer science have embraced the use of statistical models: vision, robotics and artificial intelligence are all areas which have seen a significant increase in the use of statistical approaches.

This thesis has described our attempt to marrying these two fundamental concepts. We have described our approach to learning statistical models from relational data. Our goal is to build statistical models that are able to capture and extract the information available in relational data. Our hope is that these statistical models will more accurately capture the dependencies and correlations in the domain than previous approaches and that they will prove useful for both data exploration and data summarization in relation domains.

Bibliography

S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 275–286. ACM Press, 1999.

G. Ausiello, G. F. Italiano, A. Marchetti Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12:615–638, 1991.

S. Babu, M. Garofalakis, and R. Rastogi. Spartan: A model-based semantic compression system for massive data tables. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 283—294. ACM Press, 2001.

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

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

D. Barbará, W. DuMouchel, C. Faloutsos, P. Haas, J. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. Sevcik. The New Jersey data reduction report. *IEEE Data Engineering Bulletin: Special Issue on Data Reduction Techniques*, 20(4):3–45, 1997.

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

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

J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty*, pages 43–52, Madison, WI, 1998. Morgan Kaufman.

K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, 2001.

Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 307–318, Seattle, Washington, 1998.

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

D. M. 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*, pages 80–89. Morgan Kaufman, 1997.

E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

D. Cohn and T. Hofmann. The missing link—a probabilistic model of document content and hypertext connectivity. In *Proc. NIPS 13*, 2001.

D. Cook and L. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2): 32–41, 2000.

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

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 Fifteenth National Conference on Artificial Intelligence, pages 509–516. AAAI Press, 1998.

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

C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, 7th edition, 1999.

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

L. Dehaspe, H. Toivonen, and R.D. King. Finding frequent substructures in chemical compounds. In R. Agrawal, P. Stolorez, and G. Piatetsky, editors, *Fourth International Conference on Knowledge Discovery and Data Mining*, pages 30–36, Menlo Park, 1998. AAAI Press.

A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 199—210. ACM Press, 2001.

M. desJardins, L. Getoor, and D. Koller. Using feature hierarchies in Bayesian network learning. *Lecture Notes in Artificial Intelligence*, 1864:260–270, 2000.

T. Dietterich and R. S. Michalski. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16:257—294, 1986.

S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. Kluwer, Berlin, 2001.

P. Finn, S. Muggleton, D. Page, and A. Srinivasan. Discovery of pharmacophores using the inductive logic programming system progol. *Special issue of MACHINE LEARNING on Applications and the Knowledge Discovery Process*, 30(1-2):241–270, 1998.

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

N. Friedman and M. Goldszmidt. Discretization of continuous attributes while learning Bayesian networks. In *Proceedings of the International Conference on Machine Learning*, pages 157–165, 1996.

N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 165–174. Morgan Kaufman, 1997.

N. Friedman, I. Nachman, and D. Peér. Learning of Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 139–147, Stockholm, Sweden, 1999b. Morgan Kaufman.

M. Garofalakis and P. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB'01 Tutorial, Proceedings of the International Conference on Very Large Data Bases*. Morgan Kaufmann, 2001.

L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 307–335. Kluwer, 2001a.

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

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

L. Getoor, D. Koller, B. Taskar, and N. Friedman. Learning probabilistic relational models with structural uncertainty. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20. AAAI Press, 2000b.

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

Rayid Ghani, Sean Slattery, and Yiming Yang. Hypertext categorization using hyperlink patterns and meta data. In *Proceedings of the International Conference on Machine Learning*, pages 178—185, 2001.

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

D. Hand, H. Mannila, and P. Smyth. Principles of data mining, 2001.

F. Hayes-Roth and J. McDermott. Knowledge acquisition from structural descriptions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 356—362, 1997.

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.

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

Hugin Expert. Hugin Expert AS. `http://www.hugin.com`, 2001.

G. F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Information Processing Letters*, 28:5–11, 1988.

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

R.D. King, S.H. Muggleton, A. Srinivasan, and M.J.E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. In *Proceedings of the National Academy of Sciences*, volume 93, pages 438–442, 1996.

J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 302–313. Morgan Kaufman, 1997.

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

N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, NY, 1994.

R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–11. ACM Press, 1990.

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

Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 448–459. ACM Press, 1998.

A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

S. Monti and G. Cooper. A multivariate discretization method for learning Bayesian networks from data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 404–413. Morgan Kaufman, 1998.

S. Muggleton, editor. *Inductive Logic Programming.* Academic Press, London, UK, 1992.

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

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

M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensionnal queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 28–36. ACM Press, 1988.

K. Murphy and Y. Weiss. Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufman, 1999.

J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20. AAAI Press, 2000.

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

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

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

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*, pages 541–550. Morgan Kaufman, 1999.

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

V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 486–495. Morgan Kaufmann, 1997.

P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, New York, NY, 1994. ACM Press.

U. Shardanand, P. Maes, i ltering, and A. automating. Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of the Human Computer Interaction Conference*, pages 210–217, 1995.

S. Slattery and T. Mitchell. Discovering test set regularities in relational domains. In *Proceedings of the International Conference on Machine Learning*, pages 895–902. Morgan Kaufmann, 2000.

R. Stepp and R. S. Michalski. CONCEPTUAL CLUSTERING: Inventing goal-oriented classifications of structured objects. Technical Report 940, University of Illinois, Urbana, Illinois, 1985.

Ben Taskar, Eran Segal, and Daphne Koller. Probabilistic classification and clustering in relational data. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 870–876, Seattle, Washington, 2001. Morgan Kaufman.

Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice Hall, Upper Saddle River, New Jersey, 1997.

U.S., Census Bureau. Census bureau databases. `http://www.census.gov`, 1992-93.

J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 193–204. ACM Press, 1999.

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

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

M. Wilson, J.D. DeRisi, H.H. Kristensen, P. Imboden, S. Rane, P.O. Brown, and G.K. Schoolnik. Exploring drug-induced alterations in gene expression in Mycobacterium tuberculosis by microarray hybridization. *National Academy of Sciences*, 96(22): 12833–128338, 1999.

P. Winston. Learning structural descriptions from examples. In Patrick Henry Winston, editor, *The Psychology of Computer Vision*, pages 157–209. McGraw-Hill Book Company, Inc., 1975.

N.L. Zhang and D. Poole. On the role of context-specific independence in probabilistic reasoning. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1288–1293. Morgan Kaufman, 1999.