Approximate inference in probabilistic relational models

by

Fabian Kaelin
School of Computer Science
McGill University, Montreal, Canada
August 2011

A thesis
submitted to McGill University
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science

Acknowledgements

I would like to thank Doina Precup for her invaluable mentorship.

Abstract

Probabilistic Relational Models (PRMs) are a type of directed graphical model used in the setting of statistical relational learning. PRMs are an extension to Bayesian networks, a popular model which assumes independence between observations. A PRM aims to exploit the logical structure that is often present between observations. We present two approximate inference methods for PRMs. First, we propose an algorithm based on Gibbs sampling that makes use of the relational structure for fast and scalable inference. We then consider PRMs with reference uncertainty, which are models that contain uncertainty about the relational structure itself. We propose an inference method based on a Metropolis-Hasting algorithm which depends on a sparse data structure that decreases the model complexity. Finally we present a software framework called ProbReM that can be used to model and perform inference on PRMs.

Abrégé

Les modèles probabiliste relationel (MPRs) sont un type de modèle graphique utilisés dans le cadre de l'apprentissage relationnelle. Les MPR sont une extension des réseaux Bayésiens, soit un modèle aprécié qui suppose l'indépendance entre les observations. Un MPR vise à exploiter la structure logique qui est souvent présente entre les observations. Nous présentons deux méthodes d'inférence approximative pour les MPR. Premièrement, nous proposons un algorithme basé sur l'échantillonnage de Gibbs qui fait usage de la structure relationnelle pour l'inférence rapide et évolutif. Nous considérons ensuite les MPR à l'incertitude de référence, qui sont des modèles qui contiennent des incertitudes au sujet de la structure relationnelle elle-même. Nous proposons une méthode d'inférence basée sur un algorithme de Metropolis-Hasting, qui repose sur une structure de données éparses qui diminue la complexité du modèle. Enfin, nous présentons un logiciel appelé ProbReM qui peut être utilisé pour modeler et faire de l'inférence sur les MPR.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Researchers in machine learning and related areas propose tools that aim to accurately represent some domain in the real world. Often, the goal is to teach a computer to make inference, but in order to do that the machine has to have some form of understanding of what it is asked to reason about. A computer's representation of the domain knowledge is referred to as a model, which can be learned in a manner similar to how we humans learn, by observing samples and trying to incorporate these observations into a knowledge base. This knowledge base is then used to make inferences about unobserved events or to predict outcomes of future events. One observation often consists of a vector of data points, where each data point corresponds to a measured variable. For example, the observations for a text classification system could consist of a corpus of emails, each represented by a vector of words found in the emails. In the setting of a computer vision system that aims to recognize objects in images, one observation could be a vector of pixels making up one image.

A large number of statistical frameworks that are able to learn mathematical models from data have been proposed by researchers. The model definition, as well as the learning and the inference algorithms vary wildly depending on which approach is being used and the type of inference the model is designed to make. However one central and very common assumption in many frameworks is that the observations are independent of one another. For example, in the case of the text classification system this independence assumption could mean that the words

in one email contain no information about the words of a reply to that email. Or, in the computer vision model, that the pixel values for one image are independent of the pixel values in the next image of a video sequence. Often, and including for the two examples given above, the observations are obviously not independent. However, in many cases, this assumption leads to a mathematically elegant model that can be implemented. In fact, it is often required because inference would become intractable otherwise. As useful as the independence assumption can be, one consequence is that we are not making use of the potentially valuable information about the relational structure between observations. For example, imagine an email classification system that routes messages to the appropriate recipient within an institution. A traditional method would consider the email text as one independent observation. In contrast, a relational approach could also exploit information about the subject, sender and organizational structure of the institution.

The field of statistical relational learning (SRL) aims to make use of as much of this relational information as possible while still offering concise mathematical models and tractable inference algorithms. SRL has emerged as a separate area of research in the last decade, synthesizing previous research on artificial intelligence and related fields like inductive logic programming. A number of SRL frameworks have been proposed in recent years, most of which draw heavily from previous work on graphical models and first order logic. In this thesis we focus on a specific directed graphical model used in the setting of relational learning, Probabilistic Relational Models (PRM). PRMs are an extension to Bayesian networks, a popular directed graphical model which is based on the independence between observations.

The thesis is structured as follows. Chapter 2 gives a brief introduction to the concepts on which our work builds on. This includes the basic laws of probability motivating Bayesian methods, a description of the types of graphical models and an intuitive introduction to Markov Chain Monte Carlo methods. The reader is expected to have been exposed to these subjects before. In Chapter 3 we introduce the Probabilistic Relational Model, a type of graphical model which is the basis of all original work presented in this thesis. In Chapter 4 we present a novel approximate inference method for PRMs. We leverage the special relational structure of the model in order to design an efficient algorithm. The experiments show that the algorithm is fast and that it scales well with the size of the model. We then proceed to consider a more complex model, called reference uncertainty. This model is appropriate in situations when the relational

structure, which we leveraged in Chapter 4, is itself uncertain. We propose a second, more involved, inference algorithm that depends on additional novel data structures that we describe in detail in Chapter 5. In Chapter 6 we present a software framework called ProbReM which is used to model PRMs. It implements the inference methods presented in Chapters 4 and 5 as well as other algorithms. It is available as open source. Finally, in Chapter 7 we offer a brief conclusion and we discuss avenues for future work.

We point out that the inference algorithm in Chapter 4 has been published in the proceedings of the Asian Conference on Machine Learning [KP10]. The author of this thesis is also the primary author of the paper. The coauthor, his supervisor, provided valuable input. The open source software presented in Chapter 6 was implemented by the author in its entirety.

Chapter 2

Background

In this chapter we provide a brief review of the concepts necessary for this thesis. It is by no means comprehensive and the reader is assumed to have some knowledge of probability, graph theory and inference algorithms. For a complete introduction, the reader is referred to [KF09]. In Section 2.1 we review the basic laws of probability that will be necessary to define the notion of a graphical model introduced in Section 2.2. Finally, in Section 2.3, we describe the idea behind Markov Chain Monte Carlo methods on which we heavily depend for the inference algorithms introduced in Chapters 4 and 5.

2.1 Probability

For the purpose of the algorithms presented in this thesis, we consider discrete random variables. We denote by $\mathcal{V}(X)$ the finite *domain* of the variable X. The probability of an event $x \in \mathcal{V}(X)$ is non-negative, $P(X = x) \geq 0$, and the probabilities of all events sum to one, $\sum_{x_i \in \mathcal{V}(X)} P(X = x_i) = 1$.

For a set of random variables X_i, $i = 1, 2, .., n$, the *joint probability distribution* is defined to be the distribution over all variables $P(X_1, .., X_n)$. In the case of discrete variables, a joint probability distribution can be represented in the form of a probability table. An example of a joint probability $P(X, Y)$ table for two binary variables X, Y can be found in Table 2.1.

	X^0Y^0	X^0Y^1	X^1Y^0	X^1Y^1
	0.2	0.1	0.3	0.4

	X^0	X^1
Y^0Z^0	0.3	0.7
Y^0Z^1	0.8	0.2
Y^1Z^0	0.9	0.1
Y^1Z^1	0.4	0.6

Table 2.1: Examples of discrete probability distributions in a tabular representation. On the left is the joint distribution $P(X, Y)$, on the right the conditional distribution $P(X \mid Y, Z)$. All variables are binary.

A *conditional probability distribution* (CPD) defines a distribution over variable(s) conditioned on other variables. The distribution $P(X \mid Y)$ specifies a distribution over X for each possible value of Y. The CPD $P(X \mid Y, Z)$ in Table 2.1 is an example conditional distribution where X,Y,Z are all binary variables.

In general, the conditional probability is defined as

$$P(X = x \mid Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} \tag{2.1.1}$$

The notation used in this thesis is the following. The columns are used for the distributions and the rows are used for conditional assignments. Thus the entries in each row sum to 1. When there are no conditional assignments, the first row is the distribution. This representation is convenient as in the upcoming chapters we will deal with a lot of conditional distributions over just one variable, but which are conditioned on multiple variables.

In general, we observe a subset of the variables (called the *evidence variables* $E \subset \{X_1, .., X_n\}$), and the goal is to find the conditional distribution over the unobserved variables (called the *event variables* $Y = \{X_1, .., X_n\} \backslash E$). This type of distribution, $P(Y \mid E)$, is referred to as a *query*,

which we can express in terms of the joint distribution as

$$P(Y \mid E) = \frac{P(Y, E)}{P(E)} = \frac{P(Y, E)}{\sum_Y P(Y, E)}$$

If two random variables X,Y are independent, their joint probability can be factorized as $P(X, Y) = P(X) * P(Y)$. Naturally, using Equation(2.1.1), we have $P(X \mid Y) = P(X)$ in this case. This extends to the notion of conditional independence, i.e. X and Y are independent given Z, denoted $X \perp\!\!\!\perp Y \mid Z$, if and only if:

$$P(X, Y \mid Z) = P(X \mid Z) * P(Y \mid Z) \tag{2.1.2}$$

Combining the previously described concepts, we arrive at the central probability law used in this thesis, *Bayes rule*:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} \tag{2.1.3}$$

Bayes rule allows us to write a conditional probability in terms of the *likelihood* $P(Y \mid X)$, which is a function of X, and the *prior* $P(X)$. Using the law of total probability, by summing over all possible values that X can take, the denominator becomes $P(Y) = \sum_{x \in \mathcal{V}(X)} P(Y \mid X = x)P(X = x)$. Thus the expression for $P(Y)$ does not depend on X. As stated above, often we are interested in finding a conditional distribution over unobserved variables given the evidence variables. Because $P(Y)$ is constant but usually hard to compute, the proportional notation can be used to indicate if $P(Y)$, also called the *evidence*, is not of importance:

$$P(X \mid Y) \propto P(Y \mid X)P(X) \tag{2.1.4}$$

Equation (2.1.3) is the central probability law used in Bayesian methods for machine learning. We will make use of the proportional form of Bayes rule extensively when presenting the inference algorithms in Chapters 4 and 5. But first, we will discuss how the probability laws

described in this section can be applied to graphical models.

2.2 Graphical models

In machine learning, *graphical models* have become very popular tools for reducing computational complexity by applying the Bayesian paradigm. A *model* is a computer's representation of the domain that we aim to analyze. We denote by θ the parameters that fully describe the model. The parameters can be defined by hand, but it is common to learn θ from a *training dataset*. Learning model parameters is sometimes a goal in and of itself. More often though, we aim to do inference with the learned model. A question that the model can answer is referred to as a *query*. A query usually consists of information that we know is true (the *evidence*), and we ask an *inference* algorithm to find an explanation for an *event* that we are interested in. For example, common queries are to infer the value of missing or unobserved data points; or to predict future values. The type of answers vary as well. We might be interested in the most probable explanation, which is called the maximum likelihood estimate (MLE). But more often we are interested in the maximum a posteriori (MAP) estimate, which is the probability distribution over all possible explanations given the evidence and the prior. This distribution is called the *posterior distribution*, a term we encounter often as the focus of this thesis are MAP estimates.

The Bayesian rule, Equation (2.1.1), is commonly used to answer queries. In general we can write a simple query as

$$P(\theta \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \theta) \times P(\theta)}{P(\mathbf{x})}$$

To find the posterior distribution $P(\theta \mid \mathbf{x})$ over the model parameters θ given the data $\mathbf{x}$, Bayes rule allows us to write the posterior as an expression of the likelihood of the data given the parameters $P(\mathbf{x} \mid \theta)$, the prior for the parameters $P(\theta)$ and the normalizing constant $P(\mathbf{x})$, which is the probability of the data (the evidence). Often, as the evidence does not depend on the model parameter θ, it is not necessary to compute it exactly and we use the proportional form in Equation (2.1.4). Computing the normalizing constant is difficult as it requires integrating over

all model parameters, $P(\mathbf{x}) = \sum_\theta \mathbf{P}(\mathbf{x} \mid \theta) \times \mathbf{P}(\theta)$.

Graphical models are based on a representation that captures conditional independence assumptions between certain model parameters. These independencies enable us, using Equation (2.1.1), to factorize the joint probability distribution over the full model. The benefit of a factorized joint distribution is that it is represented in a memory-efficient way and makes inference computationally cheaper. In fact, often it would be intractable to make inferences with a non-factorized joint density. There are two main approaches to graphical models; directed graphical models, described in Section 2.2.1, and undirected models (Section 2.2.2). For an excellent and complete introduction to probabilistic graphical models, the reader is referred to [KF09].

2.2.1 Directed models

We begin by introducing the well known *Bayesian network*, which is the foundation of our work in the relational domain. Assume we have a set of variables X_1,..,X_n for which we define a joint probability distribution $P(X_1, .., X_n)$. Each X_i is associated with a conditional distribution, sometimes called a *local distribution*, $P(X_i \mid \mathbf{pa}(X_i))$. The parents $\mathbf{pa}(X_i) \subset \{X_1, .., X_n\}$ specify that the parameters of X_i depend on $X_j \in \mathbf{pa}(X_i)$. The distribution can be represented via a directed graph as follows. Each X_i is represented by a vertex in the graph, and a directed edge from X_j to X_i indicates that X_j is a *parent* of X_i. For example, the local distribution of Z on the left in Figure 2.2 is $P(Z \mid X)$. If a vertex does not have any parents, the local distribution is unconditioned (e.g. $P(X)$).

Using the law of conditional probability (Equation 2.1.2), it is easy to show that the joint distribution of X_1,..,X_n can then be factorized:

$$P(X_1, ..., X_n) = \prod_{X_i} P(X_i \mid \mathbf{pa}(X_i))$$

Figure 2.1 depicts a simple Bayesian network that models the success of a student that depends on the teaching ability of his teacher as well as the students intelligence. The success variable has two parents, and the conditional distribution is $P(s \mid i, t)$.

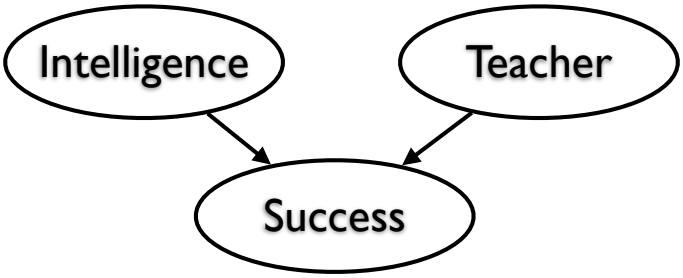


Figure 2.1: A simple Bayesian network modelling the success of a student that depends on the teaching ability of his teacher as well as the student's intelligence

The process of inference in a directed graphical model can be seen as a flow of influence through the graph. In general, the values of some vertices have been observed as evidence and we seek to find the posterior distribution over a set of vertices in which we are interested (which we call *event vertices*).

Not every vertex in the graph will inform the posterior; there are six constellations of observed and unobserved variables that fully describe the conditions for when variables are dependent. The top three cases in Figure 2.2 display the cases where the influence flows between vertices X and Y, i.e. the vertices are not independent. The special case on the right is known as *v-structure*, which is the cause of much trouble in designing efficient inference algorithms. Observing Z renders X and Y dependent; an effect also known as 'explaining away'. If we have not observed whether a student is successful, the teacher's teaching ability is independent of the students' intelligence. But if a student is known to be successful, observing a low intelligence indicates that the teacher is proficient.

The Bayes-Ball algorithm, originally introduced by Pearl [Pea88], uses these rules to determine the subgraph of the full network that will inform the posterior distribution. We say that this subgraph is, given the evidence, *d-separated* from the full network. In Section 3.5.2 we describe in detail the algorithm for determining this graph that is implemented in our software.

To compute the posterior distribution, one has the choice between exact and approximative algorithms. Variable elimination is an exact method that sums (and thus eliminates) over all

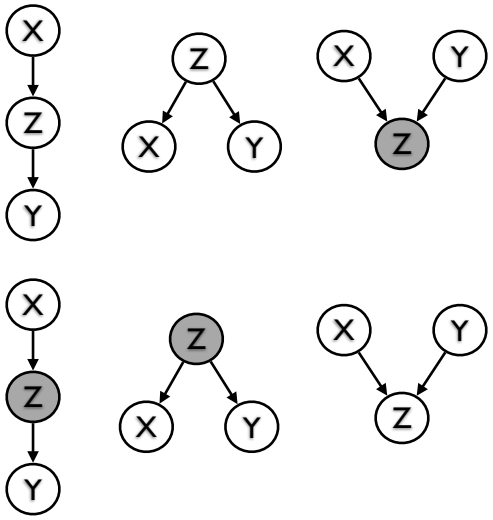


Figure 2.2: Top three cases: X and Y are dependent conditioned on Z. Bottom three cases: X and Y are independent. An observed node is shaded, unobserved nodes are transparent. All statements are conditioned on the observed variables.

variables that are not evidence or event variables. Performance strongly depends on the ordering of the variables that need to be integrated out and the structure of the network itself. Exact inference is computationally expensive, and in the Bayesian networks community, it is often replaced by approximate inference methods, which can handle large models much better. Belief propagation is an approach in which messages are passed between the vertices of an auxiliary graph until they 'agree' on a solution. Another approach is the use of Markov Chain Monte Carlo methods which will be described in Section 2.3.

2.2.2 Undirected models

Undirected graphical models are a related type of model, in which the edges in the graphical representation are not directed, thus making the dependencies between variables bidirectional. These approaches are commonly known as Markov Networks or Markov Random Fields [KSS80]. The joint density cannot be factorized using Equation (2.1.2); instead, the graph is partitioned up into a set of cliques C. The joint density is represented as

$$P(X_1, ..., X_n) = \frac{1}{Z} \prod_{c \in C} \Phi_c(X_c),$$

where Φ_c is a potential associated with clique c and $X_c \subset \{X_1, ..., X_n\}$ is the set of vertices in the clique. The normalizing constant Z is called the *partition function* and guarantees that the joint density is a valid probability distribution.

2.3 Markov Chain Monte Carlo

Originally introduced in the context of particle physics in the 1940s, Monte Carlo methods have become widely used in many fields, including machine learning. Here, we will introduce *Markov Chain Monte Carlo* (MCMC), an approximate inference method for graphical models. A *Markov chain* is a random process consisting of a collection of states with fixed probabilities of moving between any two given states. Additionally, such a process has the *Markov property*, i.e. the conditional probability distribution of future states depends only on the current state. A *transition model* (also called *transition kernel*) specifies the probability of moving from one state to another for all possible state pairs. A random walk on a Markov chain is a process that starts at an initial state, and then at each step the transition model chooses a new state. The Markov chain is constructed in such a way that a random walk is guaranteed to reach an *equilibrium distribution*, which means that there is steady probability for being in any particular state. The chain is said to have *converged* at this point. The reader is referred to [GGRS96] for a rigorous mathematical definition of these concepts.

In the case of a directed graphical model, an MCMC algorithm performs a random walk over the state space, designed so that the visited states allow us to infer an answer to the query at hand. One state of the Markov chain is associated with an assignment to all variables in the model. One step of the random walk is therefore equivalent to assigning a new value to the variables that do not have the same value as in the previous step. Collecting the assignment to each variable for every step of the random walk will result in a vector of values for each variable of the model. From the collected data points we can compute a *density estimate* of the

underlying probability density. If the chain has converged, the density estimate is equivalent to the equilibrium distribution and, as the Markov chain was constructed for that purpose, to the posterior distribution of the query.

It turns out that it is straightforward to design a Markov chain which we can use to answer queries to the graphical model. Often, constructing a Markov chain that theoretically converges to the correct posterior distribution is not the hard part.

First, there is the problem of monitoring convergence which is guaranteed only in the limit. One reason for slow convergence can be that the state space is traversed inefficiently. For example, if a step of the random walk only changes one of many model variables, the chain has to take many steps until it has visited all states enough times for us to be confident that we have reached the equilibrium distribution. The initial state can be another reason for slow convergence, even though theoretically the starting point of the random walk does not matter. Yet another situation that leads to slow convergence is when it is computationally expensive to sample a next state from the transition model. There is no deterministic method for determining convergence. However, various convergence diagnostics have been proposed to help decide when a chain has converged. The period of the random walk until we reached convergence is called *burn-in period*. We are not collecting samples from the random walk during burn-in as they are not drawn from the posterior distribution (i.e. the equilibrium distribution).

Second, once the chain is burned-in and we are collecting samples from the desired posterior distribution, we have to decide how many samples need to be collected. The density estimate that is constructed from the collected samples should be a good approximation of the real posterior distribution. Once again, there are only diagnostic metrics that are at our disposal to decide how many samples to collect. Due to the randomness of algorithm, every inference run for the same query will yield a different result. This is in fact an advantage because it allows us to execute multiple random walks and 'by comparing the convergence between chains' improve our confidence in the results.

We make use of two Markov chains in this thesis. In Chapter 4 we propose a chain that uses the *Gibbs* distribution as transition model, and in Chapter 5 we describe a chain based on a *Metropolis Hastings* algorithm.

Chapter 3

Probabilistic relational models

In this chapter we discuss the *Directed Acyclic Entity-Relationship* (DAPER) model in detail. Originally introduced by Heckerman et al. [HMK04], the DAPER model is semantically equivalent to the *Probabilistic Relational Model* (PRM).

In Section 2.2.1 we described Bayesian networks, which capture the conditional independencies between a set of variables. One instantiation of this set, which is a vector of data points, corresponds to one observation in the data. If we consider that observations are independent, a common assumption as discussed in the introduction, there are tractable inference algorithms for queries. Another limitation of Bayesian networks is that the length of one observation vector is fixed beforehand, thus it is not possible to model domains where we might have observations of varying vector length. For example in the case of the Bayesian network in Figure 2.1 that models the success of a student, instead of one teacher, the student might have a varying number of teachers that all influence her success.

A PRM is the extension of Bayesian networks for relational domains. The independence assumption is weakened by introducing probabilistic dependencies between data points or variables in different observations. Variables are considered to be instantiations of a variable class, and the probabilistic dependencies are defined between variable classes rather than on the level of instantiations. A PRM model is defined by a relational structure consisting of classes and attributes (Section 3.1), on top of which a probabilistic structure defines the dependencies between

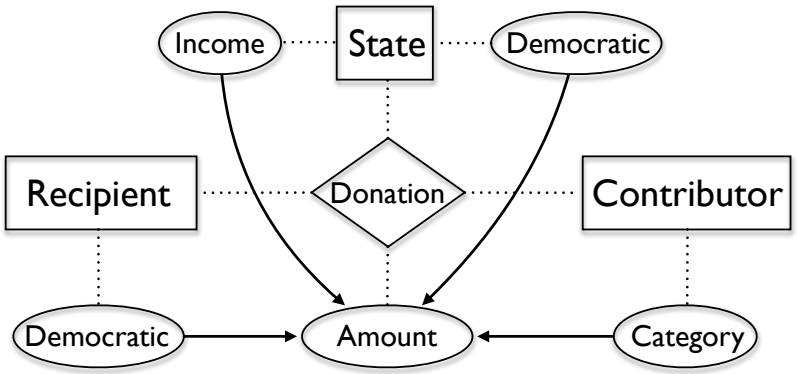


Figure 3.1: The DAPER model for the political contributions domain. The constraints $\mathcal{C}$ of the dependencies are not displayed for simplicity

elements of the relational structure (Section 3.2). Inference in PRMs is challenging, the most common approach is to transform the problem to that of inference in a traditional Bayesian network. This process is called 'unrolling' a ground Bayesian network (Section 3.5.2) and involves an auxiliary aggregation step described in Section 3.3. The problem of parameter learning from data is described in Section 3.4. Finally, in Section 3.5 we introduce the basic ideas behind doing inference and how the existing approaches tackle this challenge.

3.1 Relational structure

In database design, the *entity-relationship model* is a representation of the structure of a data set, and the DAPER framework (which is the basis of our work) is very closely related to this model. It consists of a set of *entities* $E \in \mathcal{E}$, where E represents a class of objects, a set of *relationships* $R \in \mathcal{R}$, where R links a set of entity classes $R(E_1, ..., E_n)$ and a set of *attributes* $A \in \mathcal{A}$, where each attribute A is associated with a class $X \in \mathcal{X} \equiv \mathcal{E} \cup \mathcal{R}$. We denote $\mathcal{A}(X)$ as the set of attributes associated with X. A DAPER model can be represented graphically by using rectangles to represent entities, diamonds to represent relations and ellipses to represent attributes; we use dashed lines to connect entities to relationships and attributes. An example

(which we discuss in detail later) is shown in Figure 3.1.

Every entity $E \in \mathcal{E}$ and every relationship $R \in \mathcal{R}$ contains a set of objects that are instantiations of that specific class; we refer to these sets as *entity sets* and *relationship sets* respectively. We denote by $\sigma_{\mathcal{ER}}$ the set that contains all objects in our data set (also called the *skeleton*), by $\sigma_{\mathcal{E}}$ and $\sigma_{\mathcal{R}}$ the sets of all entity objects and all relation objects, and by $\sigma_{\mathcal{ER}}(X)$ the set of objects in $X \in \mathcal{X}$. Every object $x \in \sigma_{\mathcal{ER}}(X)$ is associated with a set of attributes $\mathcal{A}(x)$. We use $x.A$ to denote an attribute of object x and $X.A$ to denote an attribute class associated with class X. Each $x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})$ has a domain $\mathcal{V}(A)$ of possible values. Finally, an instance of an entity-relationship model, $\mathcal{I}_{\mathcal{ERA}}$, consists of a skeleton $\sigma_{\mathcal{ER}}$ where for each $x \in \sigma_{\mathcal{ER}}$ and $A \in \mathcal{A}(x)$, $x.A$ is assigned a valid value in $\mathcal{V}(A)$.

Making the connection to a relational database, a table corresponds to an entity or relationship class $X \in \mathcal{E} \cup \mathcal{R}$ and the rows of the table to the skeleton objects $\sigma_{\mathcal{ER}}(X)$. The columns of the table are the set of attributes $X.A \in \mathcal{A}(X)$ where the entry in row x and column A is denoted $x.A$. The relational structure is defined by the foreign keys of the relationship class tables, which are the set of linked entities $R(E_1, ..., E_n)$.

Example 1. *In the political system of the United States, money is an important factor. Recipients of political contributions are required by law to report details about each donation they receive. The domain is suitable for modelling using PRMs. Fig. 3.1 contains an example model. The set of entities is* $\mathcal{E} = \{Recipient, Contributor, State\}$ *and the set of relationships* $\mathcal{R}$ *contains the relationship* $Donation$*. The linked entities of* $Donation$ *are* $R(Recipient, Contributor, State)$ *and every donation object* $d \in \sigma_{\mathcal{R}}(Donation)$ *has a attribute object* $d.Amount$*.*

3.2 Probabilistic structure

All attributes $A \in \mathcal{A}(\mathcal{E} \cup \mathcal{R})$ are random variables that can depend on each other. Recall that the probabilistic structure defines a set of parents for each attribute object $x.A$, denoted $\mathbf{pa}(X.A)$. A conditional probability distribution is defined for each attribute $X.A \in \mathcal{A}(\mathcal{X})$. This local distribution class - $P(X.A \mid \mathbf{pa}(X.A))$ - is shared by every object of that attribute class $x.A \in$

$\mathcal{A}(\sigma_{\mathcal{ER}}(X))$. Graphically, probabilistic dependencies among the attributes $\mathcal{A}(\mathcal{X})$ are specified by solid arrows.

Finally, given an entity-relationship model with probabilistic dependencies and a skeleton $\sigma_{\mathcal{ER}}$, a DAPER model $\mathcal{D}$ defines a joint probability distribution:

$$\begin{aligned} P(\mathcal{I}_{\mathcal{ERA}} \mid \mathcal{D}, \sigma_{\mathcal{ER}}) &= \prod_{x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})} P(x.A \mid \mathbf{pa}(x.A)) \\ &= \prod_{X \in \mathcal{E} \cup \mathcal{R}} \prod_{x \in \sigma_{\mathcal{ER}}(X)} \prod_{A \in \mathcal{A}(x)} P(x.A \mid \mathbf{pa}(x.A)) \end{aligned} \tag{3.2.1}$$

Example 2. *In the political contribution domain there is a probabilistic dependency between the* $a = Amount$ *attribute of the* $Donation$ *relationship and the* $cat = Category$ *attribute associated with the* $Contributor$ *entity. Similar dependencies exist for* $dem_{rec} = Recipient.Democratic$, $dem_{state} = State.Democratic$ *and* $inc = State.Income$. *Thus the shared local distribution of the amount is* $P(a \mid cat, dem_{rec}, dem_{state}, inc)$.

3.3 Constraints and aggregation

When a probabilistic dependency is not intra-class, i.e. the parent and child attribute are not within the same entity or relationship class, then it is the relational structure that defines which attribute objects are "connected". Friedman et al. [FGKP99] define a *slot chain*, which specifies the parents of a given child attribute object. In the context of relational databases, this leads to an expression in terms of the primary keys of the entity/relationship classes that connect the parent and child attribute. In the context of a DAPER model, Heckerman et al. [HMK04] associate a constraint $\mathcal{C}_{AB}$ with each dependency. The notion of a constraint is a generalization of what [FGKP99] refers to as *slot chain*, and can be seen as a first-order expression that defines a subset of $\sigma_{\mathcal{ER}}(X.A) \times \sigma_{\mathcal{ER}}(Y.B)$ for which the probabilistic dependency is active. Note that a constraint is more expressive than a *slot chain* as there can be multiple paths connecting two attributes; a slot chain is limited to one path.

Example 3. *The constraint $\mathcal{C}_{Amount,Category}$ will activate the dependency only between $Contributor$ objects for which the $Donation$ object is intended ($contributor[Category] = contributor[Amount]$).*

It is possible for an attribute instance $x.A$ to have multiple parents for a given dependency. The local distribution is shared among all objects of attribute $X.A$; i.e. there is only one parameter for a given parent attribute. However, this renders the conditional probability for an object with multiple parents ill-defined. Aggregation [ZP96] is an approach to this problem based on the idea that the value of the child node can be computed as a function of the values of the parent nodes, whose computation does not require knowing the number of parents in advance. If the constraint $\mathcal{C}_{AB}$ of a dependency $X.A \leftarrow Y.B$ applied to $x.A$ returns a multiset $\{y.B\}$, then the function $\gamma : \{y.B\} \rightarrow y.B_\gamma$ returns a single-valued summary of the multiset. $\gamma(\{y.B\})$ can be any function that does not depend on the actual number of elements in the set, e.g. the average, mean, mode, min, max, logical "and" (for binary variables) etc.

In Section 5.2 we introduce *Expectation Aggregation*, a novel approach to aggregation. We calculate an expected conditional probability distribution which depends on the distributions of the parent objects themselves, rather than their value. With this approach, the local distribution of an attribute object $x.A$ is always well defined.

3.4 Learning

In general, neither the local distributions nor the actual probabilistic dependencies are readily available. In a fixed world there is no uncertainty about the relational skeleton. If all variables have been observed, i.e. there are no latent variables and no missing data, the parameters can be learned using a simple maximum likelihood estimation (MLE). The likelihood of the PRM model is the probability of data given the model parameters $\mathcal{D}$ as function of $\mathcal{D}$, denoted by $L(\mathcal{D}, \sigma_{\mathcal{ER}} \mid \mathcal{I}_{\mathcal{ERA}}) = P(\mathcal{I}_{\mathcal{ERA}} \mid \mathcal{D}, \sigma_{\mathcal{ER}})$. By maximizing the likelihood we compute the parameters that best explain the data according to our model. The likelihood function is equivalent to Eq. 3.2.1, thus

the MLE is conveniently computed using the log likelihood

$$l(\mathcal{D}, \sigma_{\mathcal{ER}} \mid \mathcal{I}_{\mathcal{ERA}}) = \log P(\mathcal{I}_{\mathcal{ERA}} \mid \mathcal{D}, \sigma_{\mathcal{ER}}) \tag{3.4.1}$$

$$= \sum_{X \in \mathcal{E} \cup \mathcal{R}} \sum_{x \in \sigma_{\mathcal{ER}}(X)} \sum_{A \in \mathcal{A}(x)} logP(x.A \mid \mathbf{pa}(x.A)) \tag{3.4.2}$$

Alternatively, one could also employ Bayesian parameter learning. A prior is associated with each parameter, which avoids overfitting and also smooths irregularities in the training data. We refer the reader to [GT07, page 164] for more details.

If there are missing data and/or latent variables, we need to make use of an expectation maximization (EM) algorithm to learn the parameters of the missing data. EM is a hard problem in large relational domains. In Chapter 5 we present an inference method for a specific kind of missing data called reference uncertainty. It applies when the constraint of a probabilistic dependency is unknown, and thus all child attribute objects depend on all parent attribute objects. One of the advantages of our approach is that the learning of the parameters remains unchanged because we incorporate the uncertainty into the probabilistic model itself. If the probabilistic structure is not known, learning a model becomes a very challenging problem. In addition to having to learn which attributes depend on each other, the constraints of the dependencies would have to be learned as well.

The software package that we will present in Chapter 6 is using the MLE method. It is worth noting that the MLE method is very easy to implement using a relational database, because the number of objects with a certain configuration can usually be computed with a single database query. We further restrict the complexity of the problem by considering only discrete variables. However, it would be easy to extend the software for other cases.

3.5 Inference

Like in Bayesian networks, probabilistic inference in PRMs can be viewed as a process by which *influence* flows through the network. But instead of constraining that flow to be between the random variables of one instance, like in Bayesian networks, PRMs allow flow between interrelated

objects as well. In this section we describe how inference can be done in PRMs. First we will define what a query is in Section 3.5.1. Most previous approaches to query answering make use of the concept of a ground Bayesian network, described in Section 3.5.2. However, this is not the case for all previous work; we briefly describe other approaches in Section 3.5.3.

3.5.1 Querying

Given a model $\mathcal{D}$, a *query* $\mathbb{Q} = (\mathbb{Y}, \mathbb{E})$ is defined by a set of event variables $\mathbb{Y} \subseteq \mathcal{A}(\sigma_{\mathcal{ER}})$ and a set of evidence variables $\mathbb{E} \subseteq \mathcal{A}(\sigma_{\mathcal{ER}})$. The set of all classes in the query is $\mathbb{Q}\langle\mathcal{X}\rangle = \mathbb{Y}\langle\mathcal{X}\rangle \cup \mathbb{E}\langle\mathcal{X}\rangle$ and the sets of all objects of a class $X \in \mathbb{Q}\langle\mathcal{X}\rangle$ in the event and the evidence are denoted, respectively, by $\mathbb{Y}(X)$ and $\mathbb{E}(X)$. Finally, $\mathbb{Y}(X.A)$ designates the set $\{x.A\} \subseteq \mathbb{Y}(X)$ associated with the same attribute class $X.A$. The goal is to infer the posterior $P(\mathbb{Y} \mid \mathbb{E})$.

Example 4. *In the political contribution domain, we might be interested in predicting the political affiliation of a recipient based on the information about the donations, the contributors and the state information. The query* $\mathbb{Q}$ *would consist of* $\mathbb{Y} = \{Recipient.Democratic_i\}$*, where* i *references the* $Recipient$ *object of a specific politician and the evidence* $\mathbb{E}$ *would contain all information about the donations, the contributors and the states.*

3.5.2 Ground Bayesian networks

An *unrolling* process can be used to generate a "ground" Bayesian network. A node is created for every attribute instance of every object $x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})$. Then an arc is added to every pair $x.A$ and $y.B$, $y.B \in \mathcal{A}(\sigma_{\mathcal{ER}})$, if $X.A \in \mathbf{pa}(Y.B)$ and if $x.A$ and $y.B$ satisfy the constraint $\mathcal{C}_{AB}$. The resulting directed acyclic graph (DAG) is called the *ground Bayesian network* (GBN).

A query can be answered in a Bayesian network by taking into account only the subgraph that contains all event nodes and is d-separated from the full GBN given the evidence nodes. The d-separated ground Bayesian network generated by the unrolling process for query $\mathbb{Q}$ should therefore satisfy

$$(GBN_{\text{d-sep}} \perp\!\!\!\perp GBN_{\text{full}}) \mid \mathbb{E}, \tag{3.5.1}$$

where GBN_{full} refers to the ground Bayes net induced by the full model. In the following we refer to $GBN_{\text{d-sep}}$ simply as ground Bayesian network.

For traditional BNs there exist algorithms for determining d-separation - e.g. Bayes-Ball, proposed by [Sha98] and described in detail by [KF09, page 75]. When using PRMs, the structure of the GBN is stored in a first-order representation rather than explicitly, therefore a different approach is needed. We will describe a recursive algorithm, similar to the First-order Bayes-ball algorithm by [MTB10], that constructs a partial ground Bayesian network on an 'as needed' basis. The algorithm follows directly from the principles of d-separation and does not introduce any novel ideas, but to our knowledge it has not been described in detail in the literature so far. Starting off with the set of event variables $\mathbb{Y}$, the parents and children are iteratively loaded subject to the rules that open/break a probabilistic path in the graph according to Fig. 2.2.

Algorithm 1 implements this approach. The GBN initialized on line 5 has three vertex types: $event$, $latent$ and $evidence$ vertices. The queue on line 6 is a dictionary whose "pop()" method returns a vector $\mathbf{v}$ of vertices of the same attribute class. Thus, instead of inserting vertices individually, we load all attribute objects of the same attribute class with just one call, because the data is stored in a relational database. In lines 7-10, all event variables are added to the GBN; the goal is to find a posterior distribution over these variables.

On line 6 of Algorithm 2, if p is not in the evidence $\mathbb{E}$, the parents and children of p will also inform the posterior. For the children however, line 14 of Algorithm 1, if $\mathbf{v}$ is in the evidence $\mathbb{E}$, the children will not inform the posterior and therefore can be ignored. Eventually, either the evidence variables $\mathbb{E}$ will break all paths, at which point the partial GBN will be d-separated; or all attribute objects have been added and inference has to be done on the complete GBN. As the probabilistic dependencies are usually between attributes of different classes, the structure of the resulting GBN depends on the relational model. Examples of resulting GBNs can be found in Figure 3.2. Finally, the GBN induced by a query $\mathbb{Q}$ will consist of the evidence nodes in $\mathbb{E}$ whose values are fixed to the observed values; all other nodes $\mathbb{S} = \mathcal{A}(\text{GBN}) \backslash \{\mathbb{E}\}$ are not observed and

Algorithm 1 Unrolling the GBN

Input:
  $\mathbb{Y}$: Set of event attribute objects
  $\mathbb{E}$: Set of evidence attribute objects

Initialize empty GBN
Initialize $Queue$
for $y \in \mathbb{Y}$ **do**
  $GBN.event \leftarrow GBN.event \cup y$
  $Queue$.put(y)
end for
while $Queue \neq \emptyset$ **do**
  $\mathbf{v} \leftarrow Queue$.pop()
  Add_Parents(**v**)
  if $\mathbf{v} \notin \mathbb{E}$ **then**
    Add_Children(**v**)
  end if
end while

Algorithm 2 Add_Parents(**v**)

for $dep \in \{ \mathbf{Dep} \mid \mathbf{v}$ Is Child$\}$ **do**
  Parents(**v**) $\leftarrow$ QueryDatabase()
  for $p \in$ Parents(**v**) **do**
    if $p \notin GBN$ **then**
      if $p \notin \mathbb{E}$ **then**
        $GBN.latent \leftarrow GBN.latent \cup p$
        $Queue \leftarrow Queue \cup p$
      else
        $GBN.evidence \leftarrow GBN.evidence \cup p$
      end if
    end if
    $GBN.edges \leftarrow GBN.edges \cup `p \rightarrow \mathbf{v}`$
  end for
end for

Algorithm 3 Add_Children($\mathbf{v}$)

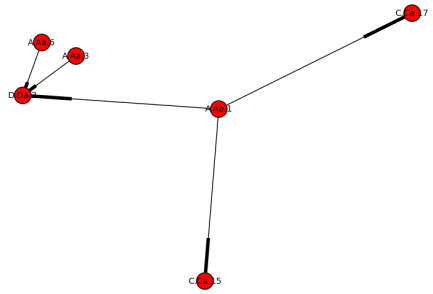
for $dep \in \{ \mathbf{Dep} \mid \mathbf{v} \text{ Is Parent} \}$ **do**
  Children($\mathbf{v}$) $\leftarrow$ QueryDatabase()
  for $c \in$ Children($\mathbf{v}$) **do**
    if $c \notin GBN$ **then**
      if $c \notin \mathbb{E}$ **then**
        $GBN.latent \leftarrow GBN.latent \cup c$
      else
        $GBN.evidence \leftarrow GBN.evidence \cup c$
      end if
      $Queue \leftarrow Queue \cup c$
    end if
    $GBN.edges \leftarrow GBN.edges \cup$ ‘$\mathbf{v} \rightarrow c$‘
  end for
end for

therefore must be inferred.

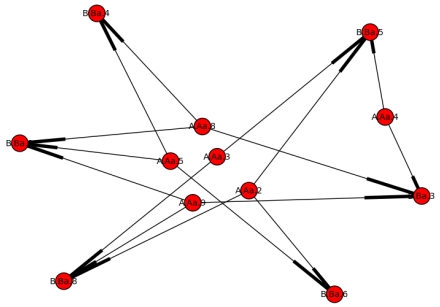
3.5.3 Related work

In directed models such as PRMs, much of the existing work involves constructing a *ground Bayesian network* (GBN) and performing inference in this model. This is similar to translating first-order formulae into propositional ones; hence, it is easy to see that the network generated can be very large, making inference very expensive. Recent work (e.g. Milch et al., 2008, Kisynski & Poole, 2009) has proposed algorithms for exact inference in lifted (i.e. non-grounded) models using aggregation. In this case, aggregation is used to avoid creating the ground Bayesian network for as long as possible. Structured variable elimination introduced by Pfeffer et al. [PK00] is an extension of the original variable elimination method [ZP96] to the relational domain.

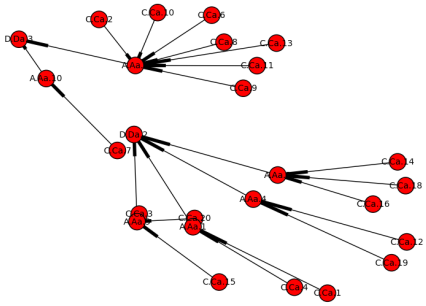
In the setting of undirected models, Taskar et al. extended Markov Networks to the relational domains by introducing the Relational Markov Network [TAK02]. Another related approach that was recently introduced is Markov Logic [RD06], presented by Richardson and Domingos. Markov Logic aims to combine Markov Networks with first-order logic, allowing inference over first-order formulae. Undirected models such as Markov Logic or Relational Markov Networks



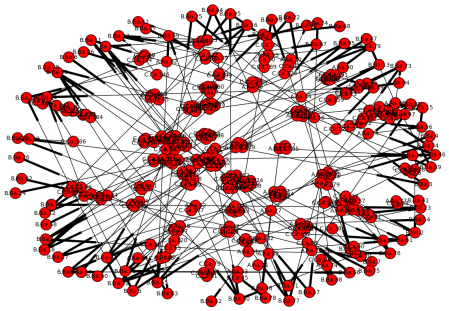
(a) A simple 1:n (A-C) and n:1 (A-D) relationship



(b) A m:n relationship



(c) Two linked relationships (C-A-D)



(d) A larger GBN based on a more complex query

Figure 3.2: The structure of the resulting ground Bayesian network for different probabilistic relationships

have the advantage that influence can flow in both directions; the drawback is increased complexity during learning and inference. We refer the reader to [GT07] for a deeper treatise of these topics.

Chapter 4

Approximate inference

In large domains, exact inference quickly becomes intractable and approximate inference is necessary. In this section we describe an approximate inference method for a fixed world scenario. That is, it is assumed that there is no uncertainty about the relational structure, and that there is a well defined DAPER model and that the parameters are already known and fixed. Given a query, we first construct the ground Bayesian network as described in Section 3.5.2. Then we use a Gibbs sampling method for approximate inference that makes use of the inherent structure induced by the relational skeleton in order to increase efficiency. We call this algorithm Lazy Aggregation Block Gibbs (LABG).

We proceed as follows: In Section 4.1 we derive the Gibbs sampling distribution for the GBN, in Section 4.2 we describe the LABG algorithm that makes efficient use of transition kernels and in Section 4.3 we present results on both artificial and real datasets.

4.1 Gibbs distribution for the ground Bayesian network

The underlying independence assumptions in Bayesian networks make Gibbs sampling a natural choice. It is well known that in a Bayesian network, samples from the proposal distribution for a variable X_i can be computed based only on the assignment of $\mathbf{x}_{(-i)}$ to the Markov blanket of

X_i [Pea88]; as a result, samples can be repeatedly drawn for all variables, generating in the end true samples from the desired conditional distribution (see section 2.3). We now describe how to leverage the structure of the GBN to reduce the computational complexity.

All sampling nodes in the GBN are associated with attribute objects $x.A_i \in \mathbb{S}$ and all $x.A_i \in \mathbb{S}(X.A)$ for some $X \in \mathbb{S}\langle\mathcal{X}\rangle$ share the same local distribution $P(X.A \mid \mathbf{pa}(X.A))$. Each $x.A_i$ can have a varying number of parents and children, therefore the full conditional distribution is $P_\phi(x.A_i \mid x.A_{(-i)})$, where $x.A_{(-i)}$ is an assignment to $\mathcal{A}(\sigma_{\mathcal{ER}}) \setminus \{x.A_i\}$.

Let $\mathcal{C} = x.A_i \cup \text{Children}(x.A_i)$; then:

$$
\begin{aligned}
P_\phi(x.A_i \mid x.A_{(-i)}) &= \frac{P(x.A_i, x.A_{(-i)})}{\sum_{x.A_i} P(x.A_i, x.A_{(-i)})} \\
&= \frac{\prod_{x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})} P(x.A \mid \mathbf{pa}(x.A))}{\sum_{x.A_i} \prod_{x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})} P(x.A \mid \mathbf{pa}(x.A))} \\
&= \frac{\prod_{x.A \in \mathcal{C}} P(x.A \mid \mathbf{pa}(x.A)) \prod_{x.A \notin \mathcal{C}} P(x.A \mid \mathbf{pa}(x.A))}{\sum_{x.A_i} \prod_{x.A \in \mathcal{C}} P(x.A \mid \mathbf{pa}(x.A)) \prod_{x.A \notin \mathcal{C}} P(x.A \mid \mathbf{pa}(x.A))} \\
&\propto P(x.A_i \mid \mathbf{pa}(x.A_i)) \prod_{y.B \in \text{Children}(x.A_i)} P(y.B \mid \mathbf{pa}(y.B))
\end{aligned}
\tag{4.1.1}
$$

The contribution of each child $y.B$ is therefore the likelihood $L(x.A_i \mid y.B, \mathbf{pa}(y.B) \backslash \{x.A_i\})$. As $x.A_i$ is in $\mathbf{pa}(y.B)$ for each $y.B$, the influence is flowing 'upwards' without the need for aggregation. Furthermore, if $\mathbf{pa}(y.B) \backslash \{x.A_i\} \neq \emptyset$ (e.g. $y.B$ has other parents besides $x.A_i$) there will also be influence flowing through the resulting V-structure. The other contribution to P_ϕ is the factor $P(x.A_i \mid \mathbf{pa}(x.A_i))$ which is influence flowing 'downwards', in aggregated form if necessary.

In general, if the number of parent attribute classes of any $x.A$ is smaller than the number of parent attribute objects, there is at least one parent attribute class for which aggregation has to be performed, because the shared local distribution constrains each parent to be single-valued. On the other hand, influence from child to parent is not aggregated since the above equation contains a product of the likelihoods of all children nodes.

4.2 Block sampling algorithm

This observation allows for "lazy" computation of aggregations. Algorithm 4 presents our approach. The sampling nodes $\mathbb{S}$ are partitioned into blocks, where each block contains all attribute objects of the same attribute class $X.A$. Then an attribute class is randomly selected with probability proportional to the size of its block to ensure that each attribute object is equally likely to be sampled. After selecting a sampling attribute class, only attribute objects of that type will be sampled in that step. In the *LazyAggregation()* step we precompute all aggregation values of parent attributes for which two attribute objects $x.A_i, x.A_j \in \mathbb{S}(X.A)$ are conditionally independent. This is the case for all parent attributes $\mathbf{pa}(X.A)$ since $(x.A_i \perp\!\!\!\perp x.A_j) \mid \mathbf{pa}(X.A)$ as well as for the parents $\mathbf{pa}(Y.B)\backslash X.A$ of the children attribute objects $y.B$ except for $X.A$ itself. In this case, because $x.A_i$ and $x.A_j$ would not be mutually independent given a common child attribute object, the aggregation is computed in the *Aggregation()* step.

Algorithm 4 generates a Gibbs trajectory guaranteed to converge to $P(\mathbb{S} \mid \mathbb{E})$ if the PRM model satisfies the standard constraints defined by [Get00]. The desired marginal posterior $P(\mathbb{Y} \mid \mathbb{E})$ can be found by summing over the latent variables $\mathbb{S}\backslash\{\mathbb{Y}\}$ since $\mathbb{Y} \subseteq \mathbb{S}$.

4.3 Experiments

The algorithm we presented only samples new values for a subset of all sampling variables during a Gibbs step. In order to compare convergence and performance properties, we compare LABG against two other samplers. The *Lazy Aggregation Standard Gibbs* (LASG) sampler makes use of the GBN structure when computing aggregate values, but samples all inference variables in $\mathbb{S}$. We also use a traditional Gibbs sampler that does not make use of the GBN structure and thus recomputes aggregations redundantly.

Algorithm 4 Lazy Aggregation Block Gibbs (LABG)

Input:
Query $\mathbb{Q} = (\mathbb{Y}, \mathbb{E})$
Number of samples N

$\mathbb{S} \leftarrow$ Unroll GBN for $\mathbb{Q}$
$P_\phi \leftarrow$ Compute Full Conditional for $x.A \in \mathbb{S}$
$\mathbf{s}^{(0)} \leftarrow$ Sample initial state
for $t = 1$ to N **do**
  $\mathbf{s}^{(t)} \leftarrow \mathbf{s}^{(t-1)}$
  $X.A \leftarrow$ Select attribute class in $\mathcal{A}(\mathbb{S})$
  LazyAggregation($X.A$) , if necessary
  for all $x.A \in \mathbb{S}(X.A)$ **do**
    Aggregation($x.A$) , if necessary
    $\mathbf{s}^{(t)}\langle x.A\rangle \leftarrow$ Sample $P_\phi(x.A)$
  end for
end for
$P(\mathbb{S} \mid \mathbb{E}) \leftarrow$ Density Estimate of $\{\mathbf{s}^{(0)},...,\mathbf{s}^{(N)}\}$
$P(\mathbb{Y} \mid \mathbb{E}) \leftarrow$ Marginalize $\mathbb{S}\backslash\{\mathbb{Y}\}$ from $P(\mathbb{S} \mid \mathbb{E})$
return $P(\mathbb{Y} \mid \mathbb{E})$

	A^0	A^1
	0.29	0.71

	D^0	D^1
A^0	0.34	0.66
A^1	0.6	0.4

	C^0	C^1
A^0	0.27	0.73
A^1	0.42	0.58

	B^0	B^1
A^0	0.59	0.41
A^1	0.31	0.69

Table 4.1: The local distributions for $P(A.a)$, $P(D.a \mid A.a)$, $P(C.a \mid A.a)$, $P(B.a \mid A.a)$

4.3.1 Artificial dataset

To illustrate the computational performance of our algorithm, we use an artificial data set whose associated relational and probabilistic model are depicted in Fig. 4.1; the local distributions are given in Table 4.1. The set of entities and relationship classes are $\mathcal{E} = \{A, B, C, D\}$ and $\mathcal{R} = \{AB, AC, AD\}$ respectively and all attribute classes are Bernoulli distributed. The model is designed to cover the set of possible basic configurations, namely one 1:n relationship ($A \to C$), one n:1 relationship ($A \to D$) and one m:n relationship ($A \to B$). The constraint $\mathcal{C}$ for all dependencies is the traditional slot chain, e.g. $A.a[A] = A.a[AB], B.a[AB] = B.a[B]$ for the dependency $A.a \to B.a$. Where necessary, the aggregation function used is the average. We are assessing the quality of the posterior (besides checking the convergence) by performing a simple classification query $P(A.a \mid B.A, C.A, D.A)$ for 19 query variables of attribute class $A.a$. The unrolled GBN is of moderate size with 19 sampling nodes and a total of 526 nodes. Figure 4.2 shows the cumulative mean of the LABG algorithm. Convergence was fast; we used a burn-in of 100 samples and then collected 200 samples from three parallel Gibbs trajectories. This proved sufficient to classify 17 out of the 19 variables correctly using a simple MAP estimate.

The model presented above allows a meaningful comparison of the proposed inference methods both in terms of convergence as well as performance. The query selected to ensure that the structure of the GBN is the same for all experiments performs inference on one attribute object $\mathbb{Y} = \{A.a_1\}$ given all attribute objects $\mathbb{E} = \{C.a_i\}$. The structure induced by the 1:n dependency $(A \to C)$ is Naive Bayes since the nodes of class C can only have one parent which is already in the event set. Every child node of class D of the n:1 dependency $(A \to D)$ can have multiple

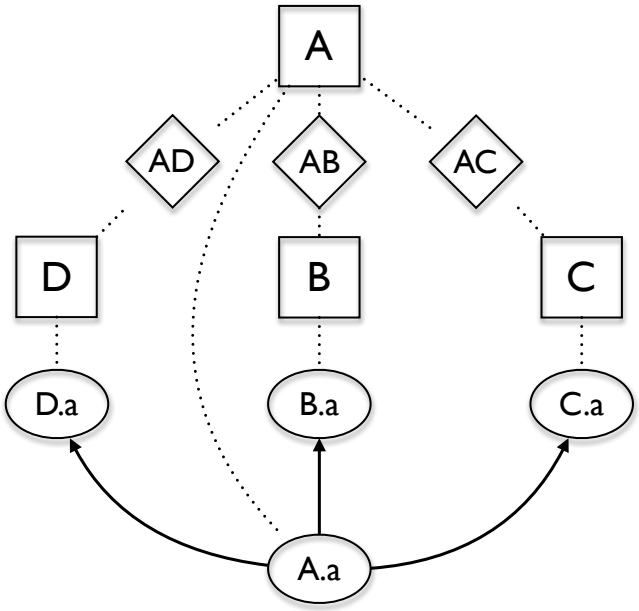


Figure 4.1: A DAPER model based on the artificial dataset

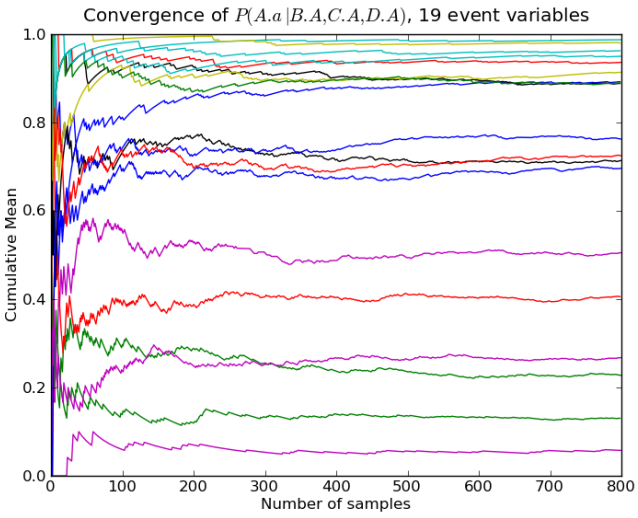


Figure 4.2: Inference on 19 $A.a$ attribute objects shows solid convergence

	Table size
A	100
B	50
C	2000
D	500
AB	200

	Table size
A	500
B	10
C	10000
D	50
AB	1000

Table 4.2: A small and a large instance of artificial datasets generated using the same local distributions (Table 4.1)

parents in A, and since neither A or D are in the evidence, all loaded nodes of type A and D will be sampling nodes. Hence, the lazy construction of the GBN will also load all the children nodes C of the A nodes that have already been loaded as parents of D. The same recursive mechanism loads nodes of type B as well. The resulting GBN will quickly grow in size and it will contain many children of type B and D with multiple parents in A. This type of query is well suited for the proposed *Lazy Aggregation* method.

To assess the convergence properties, three parallel Gibbs trajectories were run for both LABG and LASG. As the LABG algorithm samples only the variables of one randomly selected attribute class during one Gibbs step, the auto-covariance of the posterior samples is larger than when using LASG. This in turn leads to a slower traversal of the posterior probability mass because only one variable is sampled at each step. Hence, the LABG algorithm needs more samples to approximate the posterior. These effects are illustrated in Figures 4.3 and 4.4: LABG needs around 700 samples for convergence whereas LASG converges after around 400 samples. LASG also displays a lower inter-chain variance, because it is more 'guided' than LABG. These effects can be seen in Figure 4.5, which shows two chains respectively of LABG and LASG.

The *Lazy Aggregation* algorithm makes use of the GBN structure while the *Block Gibbs*

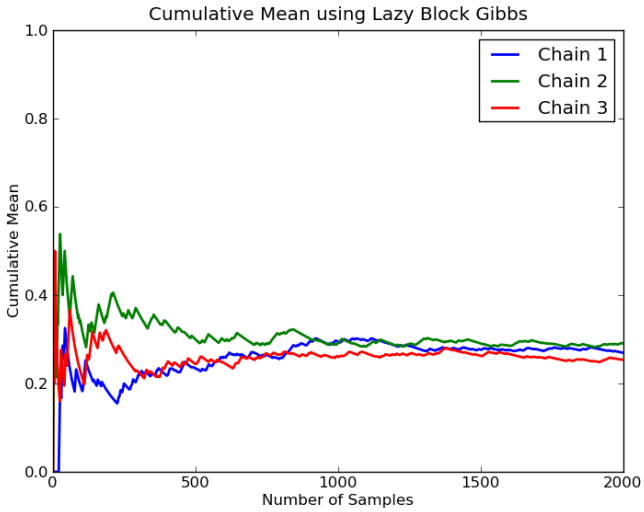


Figure 4.3: Convergence plot of three parallel chains for the Lazy Aggregation Block Gibbs sampler

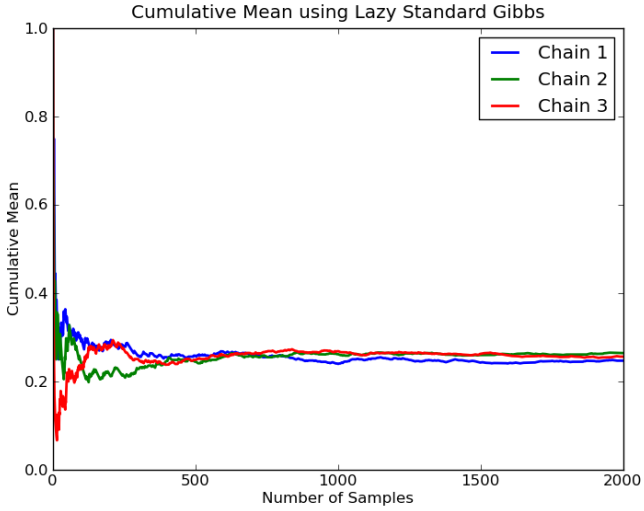


Figure 4.4: Convergence plot of three parallel chains for the Lazy Aggregation Standard Gibbs sampler

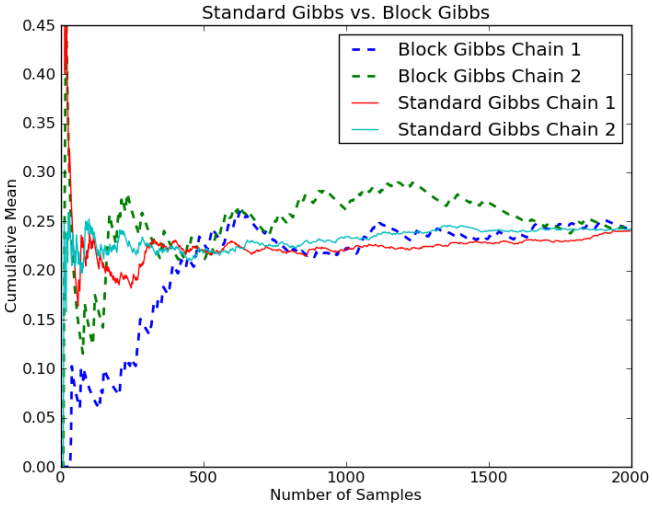


Figure 4.5: Comparison of convergence between the LABG (Block Gibbs) and LASG (Standard Gibbs) samplers. Note that the limit of the y-axis has been changed to allow greater detail in the plot

algorithm is a compromise in regards to convergence speed and precision. Both mechanisms are increasing the computational efficiency of the LABG algorithm (Figure 4.6): the former by minimizing the number of aggregations that need to be computed, and the latter by sampling fewer variables. This tradeoff seems to pay off, as LABG is about three times faster than LASG, but convergence only takes roughly twice as many samples. Furthermore, *Lazy Aggregation* seems to have a greater effect when the size of the GBN increases, which confirms that we avoid more computation if we have more nodes that share the same children. We conclude that the proposed algorithm scales well with larger queries or when increasing the data set.

Thus the compromise between speed and accuracy is flexible and the algorithm can be adapted depending on the precision needed for the inferential goals.

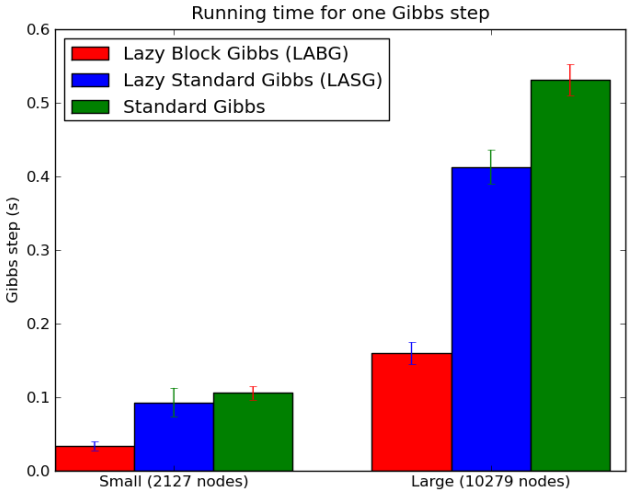


Figure 4.6: For both the big and small artificial dataset, a comparison of the average time needed to compute one Gibbs step for the three discussed Gibbs samplers.

4.3.2 Political contributions

In the United States, running a political campaign to be elected into Congress is an expensive endeavour. The amount of money at the disposal of a candidate is an important factor in a successful campaign. Recognizing this influence and the problems that come with it - corrupt lobbying, vested corporate interests - the recipient of a political contribution is required by law to report the donation. As a consequence of the recent trend towards government transparency and data digitalization, this data is now publicly available for bulk download[1] .

In order to model the data with a PRM, we considered a subset of the data, consisting of the federal contributions for the cycle 2007-2008. The recipients are either individuals running for Congress (House or Senate), Political Action Committees (PACs) or presidential candidates (Barack Obama, John McCain). To ensure statistical relevance, only recipients who received more than 1000 contributions are included in the model. The political affiliation of candidates for Congress is easily available. PACs on the other hand usually redistribute their donations to

[1] `http://www.transparencydata.com/`

candidates of both parties, which makes it harder to determine their affiliation. Each contributor is associated with a name, the US state where the donation was made, and an industry category (e.g. Oil & Gas, Gun Rights, Retired). The size of the dataset and the cardinality of the attributes are displayed in Table 4.3. We augmented the data with information about the contributor state: a binary variable indicating the income level (above or below the US average) and a binary variable for the political affiliation of the state based on the outcome of the previous presidential election.

Class	Size
Recipient	430
State	50
Donation	~ 2300000
Contributor	~ 2300000

Attribute	Cardinality
Category	115
Recipient.Democratic	2
Amount	6
Income	2
State.Democratic	2

Table 4.3: The number of objects in the entities and relationships of the PRM and the cardinality of the attributes

The query from Example 4 attempts to predict the political affiliation based on the donation, contributor and state information. As mentioned above, there is no clear ground truth available for PACs. To examine the quality of the model, we split the data about the individual recipients in a training and test set. The test set contains 30 democratic and 30 republican individual candidates for Congress, randomly selected; the model is trained using the contributions for the remaining 370 recipients. To compute the accuracy of the learned PRM, the proposed inference method (Algorithm 4) is run for each recipient in the test set. The structure of the GBN is different for each query. The number of nodes in a specific GBN depends on the number of contributions the recipient has received. The size of the resulting GBNs and the performance of the algorithm is presented in Table 4.4.

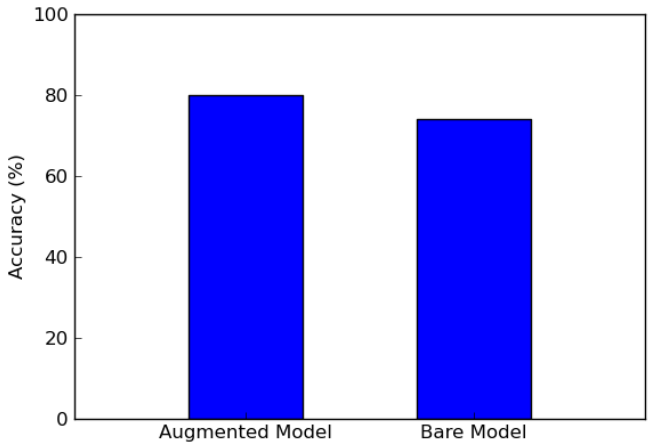


Figure 4.7: The model making use of the information about the state of the contributor is performing better (80%) than the bare model only making use of the industry category of the contributor (74%)

	GBN Size (nodes)
Min	2045
Max	23201
Average	4482

	Running Time (s)		
	Average	Min	Max
Unrolling GBN	235	209	303
Gibbs Sampling	4.8	2.8	13.3

Table 4.4: Statistics about the size of the GBNs and the running time of the algorithm

Fig. 4.7 shows the accuracy of the augmented model compared to a simplified model that did not make use of the additional information about the $State$. The inclusion of the additional information increases the quality of the model. Among the 12 recipients that were classified incorrectly, nine are Democrats and three are Republicans. To gain more insight into the results, we also examine the log-likelihood information. Specifically, we are interested in the log-likelihood of the individual recipients, which can be computed from the Markov blanket of the recipient

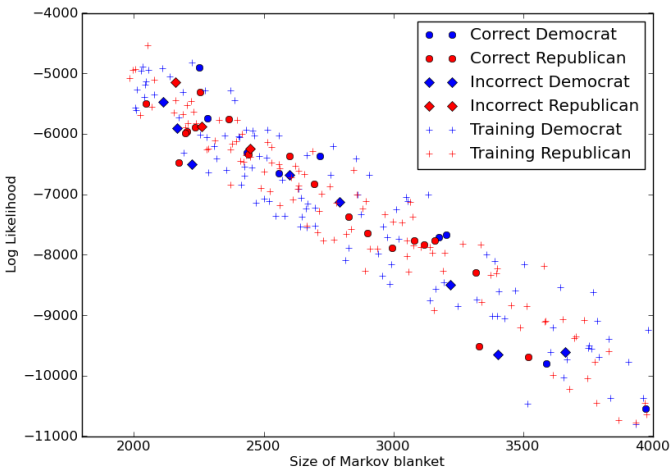


Figure 4.8: A scatter plot of the size of the Markov blanket and the log likelihood of the recipient. The red and blue colours indicate republican and democratic affiliations respectively. A circle means the recipient has been correctly classified whereas a diamond indicates a misclassification.

object. As the number of nodes in the Markov blanket depends on the number of donations a recipient receives, the log-likelihood values of the individual recipients cannot be compared directly. Fig. 4.8 illustrates this correlation; we note that most misclassifications are found in the top-left region where the log-likelihood is high and the number of donations low. By "normalizing" the log-likelihood by the number of nodes in the Markov blanket, we obtain scaled values for each recipient, which are more directly comparable. In Fig. 4.9 the model's struggle with democratic recipients becomes apparent; even though the mean is almost the same (d$em = -2.57$, r$ep = -2.58$), the variance of the scaled likelihood is larger (d$em = 0.041$, r$ep = 0.015$) for the democratic recipients. We conclude that the pattern of contributions of democratic recipients is more complex to model than for republican contributions.

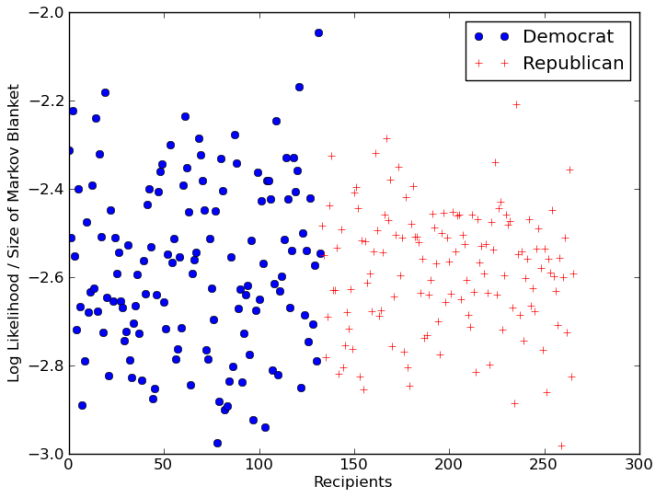


Figure 4.9: The ratio between the log-likelihood and the size of the Markov blanket allow for a direct comparison of recipients. The mean of the democratic and republican recipients is similar, but the democratic recipients display a higher variance.

Chapter 5

Reference uncertainty

When the relational structure is given, learning and inference in such models is quite straightforward. However, in real data (e.g. analysis of publication corpuses, social networks, or video information) the relational structure may not be known precisely ahead of time. Two types of uncertainty in the relational structure may arise. *Reference uncertainty* means that the objects of a relationship class may be uncertain. For example, when receiving a paper for blind review, one does not know the author of the paper (and may attempt to infer it from the paper title or abstract). *Identity uncertainty* refers to the fact that some data items may correspond to a single object, or multiple objects may actually be one and the same. This type of aliasing happens, for instance, when two authors have the same last name and initial, but they are indeed distinct people.

In this chapter, we focus on reference uncertainty in Probabilistic Relational Models, and propose a way to model it, which is flexible, has very clear semantics and leads to efficient approximate inference. We build on top of the DAPER model described in chapter 3. The DAPER model is augmented with binary "exist" variables, which can be constrained to capture known structure in the domain (e.g. a paper cannot have an arbitrary number of authors). These variables allow us to still use a traditional model, even in the presence of reference uncertainty. This is in contrast with the previous approach of Pasula & Russell [PR01], who modify the probabilistic model structure in order to capture the relational uncertainty; we discuss their previous work in

section 5.1.

The rest of the chapter is organized as follows. We provide new definitions for the concepts of a *constraint* and *aggregation* in Section 5.2. In Section 5.3 we describe an augmented version of a DAPER model which captures reference uncertainty. Most importantly, this type of model allows for an efficient Metropolis-Hastings algorithm, which we will describe in Section 5.4. In Section 5.5 we present experiments with the proposed model on a student-professor domain, we show that the model is more expressive than that proposed by Pasula et al., and that inference can be done efficiently.

5.1 Previous work

Our work is based on a previous approach by Pasula & Russell [PR01]. They present a student-professor model that depicts an academic dataset in which the *success* of a student depends on the the *fame* of a professor. The constraint specifies that the success of a given student only depends on the fame of the professor(s) that advise(s) him or her. In addition, the *fame* of a professor is also a parent attribute of the *funding* of the professor (the DAPER version of that model is shown in Figure 5.3). All attributes are binary variables. They introduce a reference attribute v_{ref}, which models the possible edges for an uncertain dependency. The CPD of $v_{ref.S_1}$ is defined as $P(S.advisor \mid P_1.\$\$, .., P_n.\$\$)$, as seen in Figure 5.1. This distribution depends on the objects in the relational skeleton $\sigma_{\mathcal{ER}}$. Thus, the reference attribute is mixing the relational structure with the probabilistic structure. However, one of the goals for defining relational probability models is the separation of these structures. Without it, inference quickly becomes intractable. For this reason, Pasula et al. are forced to constrain the admissible CPDs to a family of distribution with certain structural properties. In addition, the model is restricted to uncertain relationships of type *n:1* (a student is advised by only one professor). We propose a different representation which generalizes the approach by Pasula & Russell, while at the same time adhering to the original design choices for relational probability models.

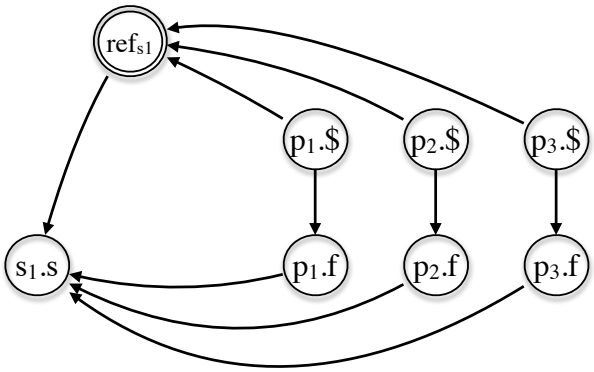


Figure 5.1: The student-professor model with reference uncertainty as proposed by [PR01]

5.2 Expectation aggregation

In the traditional definition of a PRM [Get00], every vertex $v_{x.A}$ of the GBN is associated with an attribute $X.A$ and they all share the same conditional probability distribution (CPD), $P(X.A \mid \mathbf{pa}(X.A))$. Thus it is possible, e.g. if a dependency leads through a relationship of type *1:n*, that the constraint $\mathcal{C}_{AB}$ for $v_{x.A}$ returns multiple parent vertices for a particular attribute $Y.B \in \mathbf{pa}(X.A)$. In section 3.3 we introduced the concepts of constraints and aggregation.

Here we propose a more general approach. When unrolling the network, the CPD for $v_{x.A}$ is augmented with a set of binary 'exist' variables. For every possible parent vertex, $v_{y.B}$ for $y.B \in \mathbf{pa}(X.A)$, we associate one 'exist' variable $e_{y.B}$. Its value is set to 1 if there is an edge between $v_{x.A}$ and $v_{y.B}$. The CPD of $v_{x.A}$ is given by $P(x.A \mid \{e_{y.B}\}, \{y.B\})$.

The CPD of a vertex defines a probability distribution over $\mathcal{V}(x.A)$ for every possible assignment of $\{e_{y.B}\}, \{y.B\}$. Clearly, if only one $e^{i}_{y.B}$ is 1 (all other $e^{(-i)}_{y.B}$ set to 0) for each $Y.B \in \mathbf{pa}(X.A)$, then there is a 'valid' assignment and the conditional distribution for $v_{x.A}$ is equivalent to corresponding assignment in $P(X.A \mid \mathbf{pa}(X.A))$. If more than one 'exist' variable per parent is set to 1, aggregation is necessary. We propose a novel aggregation method called *expectation aggregation* which is based on the probability of the parent values rather than on the values themselves.

We define the conditional distributions for parent assignments which require aggregation as

a linear combination of the shared distribution $P(X.A \mid \mathbf{pa}(X.A))$. First, consider the simple case where $X.A$ has only one parent attribute ($|\mathbf{pa}(X.A)| = 1$). Assuming $Y.B = \mathbf{pa}(X.A)$, the CPD of vertex $v_{x.A}$ has $2 * |\sigma_{\mathcal{ER}}(X)|$ conditional variables. For each possible conditional assignment with multiple 'exist' variables set to 1, we define the conditional distribution to be:

$$P(x.A \mid \{e_{y.B}\}, \{y.B\}) = \frac{\sum_{\{e_i | e_i \in \{e_{y.B}\}, e_i = 1\}} P(X.A \mid Y.B = y.B_i)}{\sum_{X.A} \sum_{\{e_i | e_i \in \{e_{y.B}\}, e_i = 1\}} P(X.A \mid Y.B = y.B_i)} \tag{5.2.1}$$

P(S.s | P.f) =

f	s=0	s=1
0	0.8	0.2
1	0.3	0.7

P(S_1.s | { e_i }, { P_i.f }) =

e_1	e_2	e_3	P.f_1	P.f_2	P.f_3	s=0	s=1
0	0	1	x	x	0	0.8	0.2
0	0	1	x	x	1	0.3	0.7
	.....			.....		.	.
0	1	0	x	0	0	0.8	0.2
0	1	0	x	1	0	0.3	0.7
	.....			.....		.	.
1	0	1	0	x	1	0.55	0.45
	.....			.....		.	.

Figure 5.2: The CPD of the $S_1.sucess$ vertex in the GBN. Only a few representative rows are displayed. A x indicates that the conditional distribution is the same independent of that value. The distribution for the last row (S_1 is co-advised by P_1 and P_3) is computed using Equation (5.2.1)

In Figure 5.2, the CPD of $S_1.sucess$ gives a simple example. The last row defines the distribution over the success of S_1 given that he is co-advised by P_1 and P_3. P_1 is famous while P_3 is not. Using Equation (5.2.1), we can compute these probabilities as $[\frac{0.8+0.3}{1.1+0.9}, \frac{0.2+0.7}{1.1+0.9}]$.

In the case of two parent attributes $\mathbf{pa}(X.A) = \{Y.B, Z.C\}$, both parent attributes could require aggregation. Every possible assignment of the parents vertices is considered by the

cartesian product of the sets of 'exist' variables associated with each parent. The conditional distribution $P(x.A \mid \{e_{y.B}\}, \{e_{z.C}\}, \{y.B\}, \{z.C\})$ is defined as:

$$\frac{\sum_{(e_i,e_j)\in\{\{e_{y.B}\}\times\{e_{z.C}\}|e_i,e_j=1\}} P(X.A \mid Y.B = y.B_i, Z.C = z.C_j)}{\sum_{X.A}\sum_{(e_i,e_j)\in\{\{e_{y.B}\}\times\{e_{z.C}\}|e_i,e_j=1\}} P(X.A \mid Y.B = y.B_i, , Z.C = z.C_j)} \tag{5.2.2}$$

Similarly, for n parent attributes, Equation (5.2.2) generalizes to an n-ary cartesian product. In general, assuming binary attributes, the CPD of a vertex will have $2^{2*\sum_{Y.B\in\mathbf{pa}(X.A)}|\sigma_{\mathcal{ER}}(Y.B)|}$ possible assignments to the conditioning variables. Clearly, this representation would not scale if it required the CPD to be stored in memory or on disk. However, this is not necessary because the conditional probability can be computed as a linear combination of the simple shared CPD. Further, we can exploit the sparsity of the 'exist' variables by only storing the edges that actually exist.

Conceptually, a constraint $\mathcal{C}_{AB}$ is equivalent to an assignment to all 'exist' variables for a dependency. This assignment is introduced as part of the evidence $\mathbb{E}$ for a given query $\mathbb{Q}$. If the world is stationary and there is no uncertainty, we are assured that 'exist' variables are never sampled. Finally we point out that the *expectation aggregation* method eliminates the need for specifying aggregators, because the CPDs of the GBN vertices are guaranteed to always be well defined.

5.3 DAPER model with reference uncertainty

In the context of a DAPER model $\mathcal{D}$, reference uncertainty means that the objects $\sigma_{\mathcal{ER}}(R_u)$ of a relationship $R_u \in \mathcal{R}$ are uncertain. It is important to realize that uncertainty refers to the existence of an object $x \in \sigma_{\mathcal{ER}}(R_u)$, not to the probabilistic attributes $\mathcal{A}(R_u)$. If the constraint $\mathcal{C}_{AB}$ of a dependency depends on R_u, the existence of each possible edge between the parents vertices $\{v_{y.B}\}$ and the child vertex $v_{x.A}$ is uncertain. Naturally, the notion of 'exist' variables introduced in Section 5.2 can be easily used to model this situation. We append a binary $R_u.exist$ variable to the set of attributes $\mathcal{A}(R_u)$ and we consider an edge to be active if $R_u.exist = 1$.

However, note that the 'exist' variables do not necessarily correspond to the $R_u.exist$ attribute. The difference is that the 'exist' variables are auxiliary variables representing a possible edge in the GBN. If the constraint of a dependency is introduced as evidence, thus defining which edges are active, it is possible that the constraint depends on multiple relationship classes (i.e. a slotchain of length 2 or more in traditional PRMs). The $R_u.exist$ on the other hand is part of the probabilistic model, with a set of parents $\mathbf{pa}(R_u.exist)$. It still determines the existence of an edge, but in a probabilistic manner and conditioned only on the $R_u.exist$ attribute of the uncertain relationship.

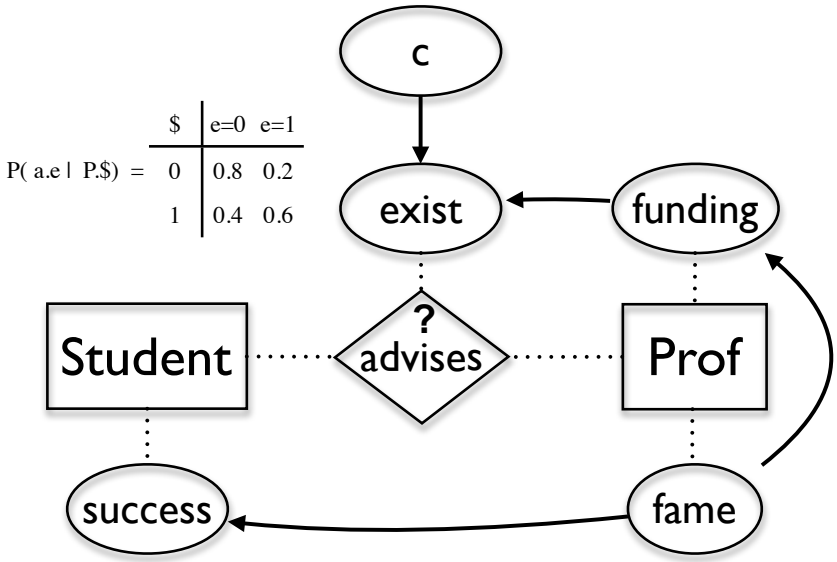


Figure 5.3: The student-professor model with reference uncertainty.

To introduce our approach, we use the student-professor model previously introduced by Pasula et al. Figure 5.3 displays the DAPER version of the model, the relationship *advises* is now uncertain and the $exist$ attribute of an *advises* object depends on the funding of the professor associated with the *advises* object. The state space of this model would grow exponentially with the number of professors, as every possible assignment to the *exist* variables has to be considered. To render this approach tractable, we introduce a deterministic binary constraint *c* which enforces certain structural properties. Specifically, $c = 1$ is always part of the evidence for every query for which we run inference. The deterministic CPD $P(c|\{exist\})$ assigns a probability of 1 to every

'allowed' assignment to the *exist* variables and a probability of 0 to all others. The unrolled GBN for the student-professor model in figure 5.4 illustrates the simplest of all possible constraints. It enforces a *n:1* relationship, so that every student is advised by exactly one professor. In general, we introduce a hyper parameter k, where k is the maximal number of $exist$ vertices that are allowed to be equal to 1. We denote as *n:k* the type of the uncertain relationship. This represents a fixed parameter approach that allows us to set k such that the state space is still tractable for inference. Naturally, both the $exist$ vertices as well as the constraint *c* are integrated into the full joint probability distribution of the model.

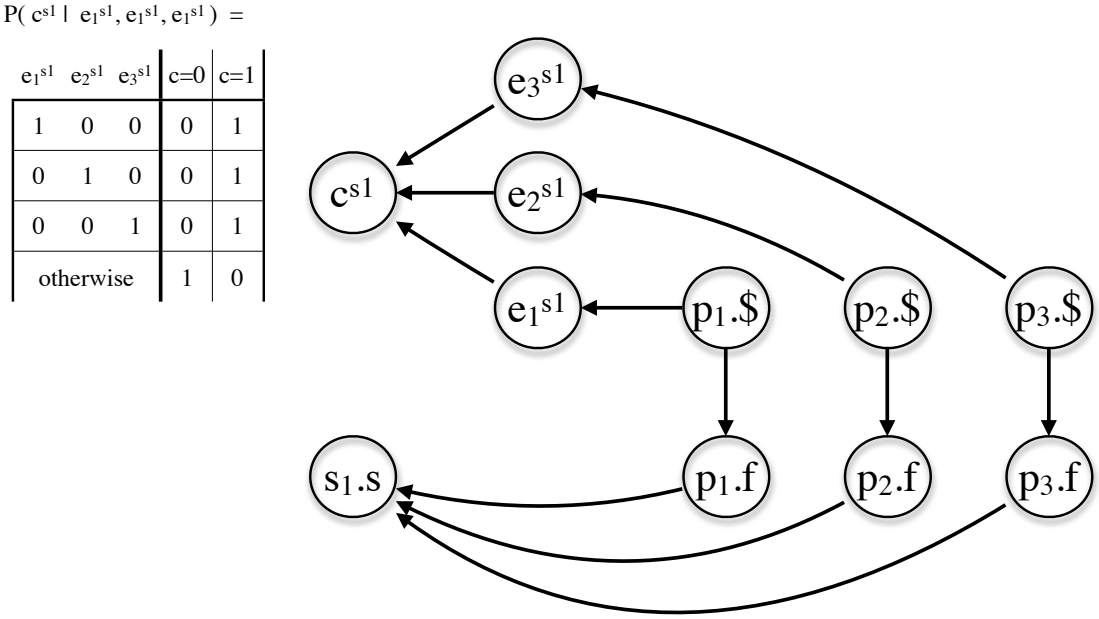


Figure 5.4: An example GBN for the student-professor model. The constraint enforces a *n:1* relationship, every student is advised by only one professor.

It is important to point out that it is not necessary to store the value of all $exist$ variables. The domain is very sparse, so there are at most k $exist$ vertices set to 1 at any given point. This observation allows us to store only the k references necessary to fully describe the state space. In the next section we describe an MCMC algorithm which allows efficient inference in this domain.

5.4 Metropolis-Hastings algorithm

To infer the posterior probability distribution $P(\mathbb{Y} \mid \mathbb{E})$ given a query $\mathbb{Q} = (\mathbb{Y}, \mathbb{E})$, we resort to an approximate inference method as the state space is very large. A MCMC algorithm generates samples from a Markov chain whose stationary distribution is equivalent to the desired posterior distribution, as described in section 2.3. As described in Section 4.1, in Gibbs sampling the value for each vertex is sampled from its full conditional distribution P_ϕ. Let $\mathcal{C} = v_{x.A} \cup \text{Children}(v_{x.A})$ and $v_{x.A}^{(-i)} = \text{GBN} \setminus v_{x.A}$. We denote the full conditional as

$$P_\phi(v_{x.A} \mid v_{x.A}^{(-i)}) = \frac{P(v_{x.A}, v_{x.A}^{(-i)})}{\sum_{v_{x.A}} P(v_{x.A}, v_{x.A}^{(-i)})} = \frac{\prod_{v \in \text{GBN}} P(v|\mathbf{pa}(v))}{\sum_{v_{x.A}} \prod_{v \in \text{GBN}} P(v|\mathbf{pa}(v))}$$
$$= \frac{\prod_{v \in \mathcal{C}} P(\cdot|\cdot) \prod_{v \notin \mathcal{C}} P(\cdot|\cdot)}{\sum_{v_{x.A}} \prod_{v \in \mathcal{C}} P(\cdot|\cdot) \prod_{v \notin \mathcal{C}} P(\cdot|\cdot)} \propto P(v_{x.A} \mid \mathbf{pa}(v_{x.A})) \prod_{c \in \text{Children}(v_{x.A})} P(c \mid \mathbf{pa}(c)) \quad (5.4.1)$$

The full conditional of a vertex $v_{x.A}$ only depends on the parents, the children and the children's parents. Thus this transition kernel is local and efficient to compute for all attributes except the $exist$ variables. Let $\{v_{exist}^c\} = \{v_{exist}^1, .., v_{exist}^n\}$ be the set of all $exist$ variables associated with constraint c. First we note, using Equation (5.4.1), that the full conditional for one $exist$ vertex,

$$P_\phi(v_{exist} \mid v_{exist}^{(-i)}) \propto P(v_{exist} \mid \mathbf{pa}(v_{exist})) P(c \mid \{v_{exist}^c\})$$

depends on the deterministic constraint c. As c is a deterministic or-gate, the Markov chain is thus reducible and the chain would stay in the initial state forever. This could be avoided by using a transition kernel that block-samples all $exist$ attributes in one step:

$$P_\phi(\{v_{exist}^c\} \mid \text{GBN} \setminus \{v_{exist}^c\}) \propto P(c \mid \{v_{exist}^c\}) \prod_{v_e \in \{v_{exist}^c\}} P(v_e \mid \mathbf{pa}(v_e))$$

This expression however depends on the value of all $exist$ variables and their parents. Even with the sparse representation offered by the hyperparameter k, all $exist$ variables need to be evaluated as their parent assignments cannot be sparsely represented. Clearly, Gibbs sampling is intractable for the $exist$ attributes.

Metropolis-Hastings (M-H) offers a transition kernel that relies on a distribution $Q(\mathbf{x}' \mid \mathbf{x})$ for proposing a new state $\mathbf{x}'$. The new state is then accepted with probability

$$\alpha(\mathbf{x}' \mid \mathbf{x}) = \frac{P_{\mathcal{I}_{\mathcal{ERA}}}(\mathbf{x}')Q(\mathbf{x} \mid \mathbf{x}')}{P_{\mathcal{I}_{\mathcal{ERA}}}(\mathbf{x})Q(\mathbf{x}' \mid \mathbf{x})}$$

We introduce the M-H step for $exist$ vertices. The distribution $Q(\{v'_{exist}\} \mid \mathbf{Z})$, where $\mathbf{Z}$ will be described below, will propose a new assignment to $\{v_{exist}\}$. We denote $\{v^{ch}_{exist}\}$ as the set containing all $exist$ variables that have changed their value and $v'_{x.A}$ the vertex whose $exist$ variables were affected. The acceptance probability will simplify to the following form:

$$\alpha(\mathbf{x}' \mid \mathbf{x}) = \frac{P(v'_{x.A} \mid \mathbf{pa}(v'_{x.A})) \prod_{v'_e \in \{v^{ch}_{exist}\}} P(v'_e \mid \mathbf{pa}(v'_e))Q(\{v_{exist}\} \mid \mathbf{Z})}{P(v_{x.A} \mid \mathbf{pa}(v_{x.A})) \prod_{v_e \in \{v^{ch}_{exist}\}} P(v_e \mid \mathbf{pa}(v_e))Q(\{v'_{exist}\} \mid \mathbf{Z})} \tag{5.4.2}$$

As the hyperparameter k limits the number of active $exist$ vertices, the product contains at most $2k$ factors. For example, the α for the example in Fig. 5.4, if the reference changes from v^j_e to v^i_e, is then

$$\alpha(\mathbf{x}' \mid \mathbf{x}) = \frac{P(S_1.s \mid P_i.f)P(v^i_e = 1 \mid P_i.\$)P(v^j_e = 0 \mid P_j.\$)Q(\{v_{exist}\} \mid \mathbf{Z})}{P(S_1.s \mid P_j.f)P(v^i_e = 0 \mid P_i.\$)P(v^j_e = 1 \mid P_j.\$)Q(\{v'_{exist}\} \mid \mathbf{Z})}$$

The proposal Q allows us to combine the relational skeleton and the probabilistic structure in a clean way. Q depends on $\mathbf{Z}$, and we can choose $\mathbf{Z}$ to include constraints that depend on the relational skeleton. For example, we could use the CPD of the reference attribute introduced by Pasula et al. as a proposal for our sampler. We can still maintain the desired separation between the relational structure of the data and the probabilistic dependencies because Q is not part of the joint distribution of the model. In the experiments in Sec. 5.5 we will introduce other examples of proposal distributions.

One might be under the impression that the network structure changes during inference,

which could have consequences on the convergence guarantees of the MCMC algorithm. But as the vertices in the GBN are conceptually dependent on all possible parent vertices (through the 'exist' variables described in section 5.2), the network is always well defined and our algorithm can efficiently exploit the sparsity of the domain.

5.5 Experiments

To test the proposed algorithm, we perform a set of experiments based on the model in Figure 5.3 introduced by Pasula et al. First we show that the proposed inference method converges to the correct posterior. The GBN (Figure 5.4) consists of three professors and one student. The hyperparameter is fixed to $k = 1$, meaning that a student is only advised by one professor. The goal is to infer the posterior probability of the success of the student given the information about the professors (funding,fame). We use a uniform proposal distribution. At each M-H step, the advisor of the student is uniformly chosen among the professors. Figure 5.5 shows that the posterior converges to the correct posterior value. As expected, the number of samples needed for convergence grows linearly with the number of professors.

This experiment very closely resembles the model introduced by Pasula et al. However, in our model, the existence of an advisor relationship depends on the funding of a professor and this dependency is part of the probabilistic structure (and thus the full joint distribution). In order to make the two models equivalent, we would have to choose a uniform prior distribution on the $exist$ attribute; it would have no parents and would thus cancel in Equation (5.4.2). Additionally, we would choose our proposal $Q(\mathbf{X}|\mathbf{Z})$ to be equivalent to the CPD of their *reference attribute*. But as the dependency on the relational structure would be in the Z, the probabilistic model is still well defined and we do not have to introduce restrictions on the CPDs of the model in order to make inference tractable. Hence, our model provides a generalization of Pasula et al's approach.

In the second experiment, we extend the previous model by setting the hyperparameter to $k = 3$ and by increasing the number of professors to 20. The query remains the same, but the student is now co-advised by three professors. We choose a two-step proposal distribution. First

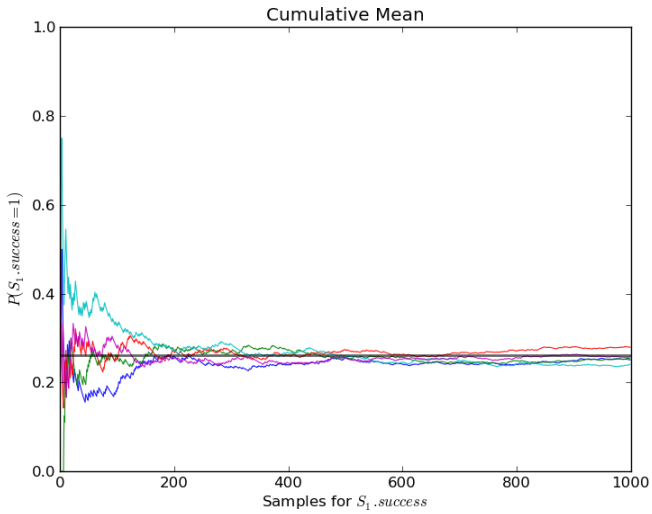


Figure 5.5: Convergence plot for 5 MCMC chains of the GBN in Figure 5.4, the horizontal line represents the exact posterior probability

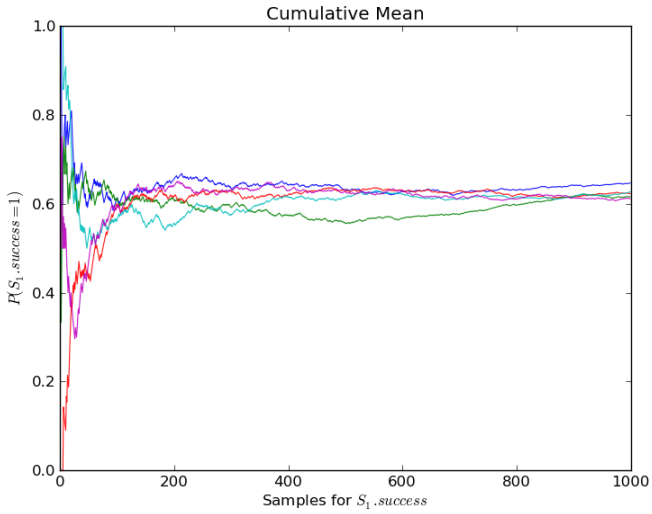


Figure 5.6: Convergence plot with the hyperparameter fixed to $k = 3$. Even though the data is generated from the same underlying distributions, the chance of success of a student is higher because he can be co-advised by 3 professors

a reference is uniformly chosen among the k references, then it is replaced with a uniformly chosen professor. As before, Q cancels out when calculating α. In Figure 5.6 we can see that convergence is still very clear; due to the larger state space convergence is somewhat slower. We consider two convergence diagnostics to monitor the convergence of our algorithm. The autocorrelation by [GGRS96] displayed on top of Figure 5.7 provides a normalized estimate that quantifies to what extend the chains have mixed in l steps. We note that the dependence on previous samples decreases quickly for both the success as well as the exist variables, although the latter display a slightly higher autocorrelation. Despite the oscillation, it seems that the proposed MCMC algorithm is traversing the state space efficiently. This is confirmed by the Gelman-Rubin diagnostic by [GGRS96], displayed in Figure 5.8, which measures the level of disagreement between chains. The closer the value is to 1, the higher the agreement.

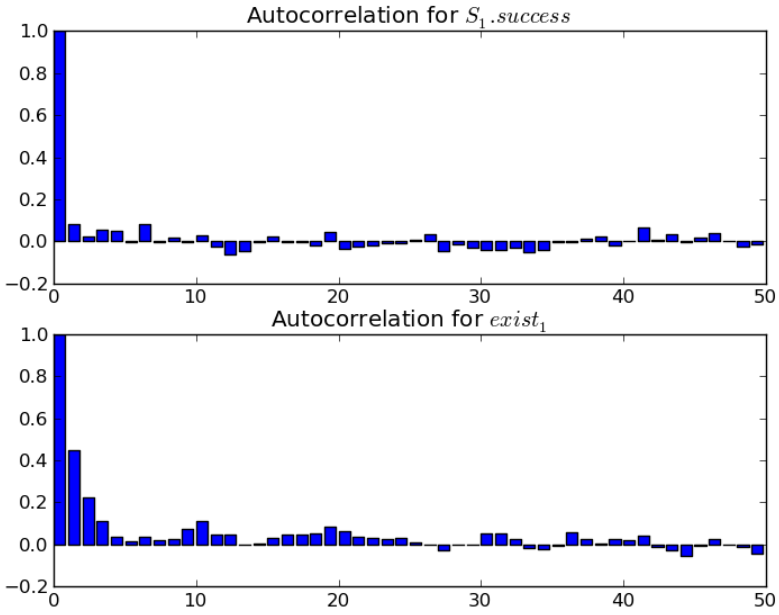


Figure 5.7: Autocorrelation for lag 1 through 50.

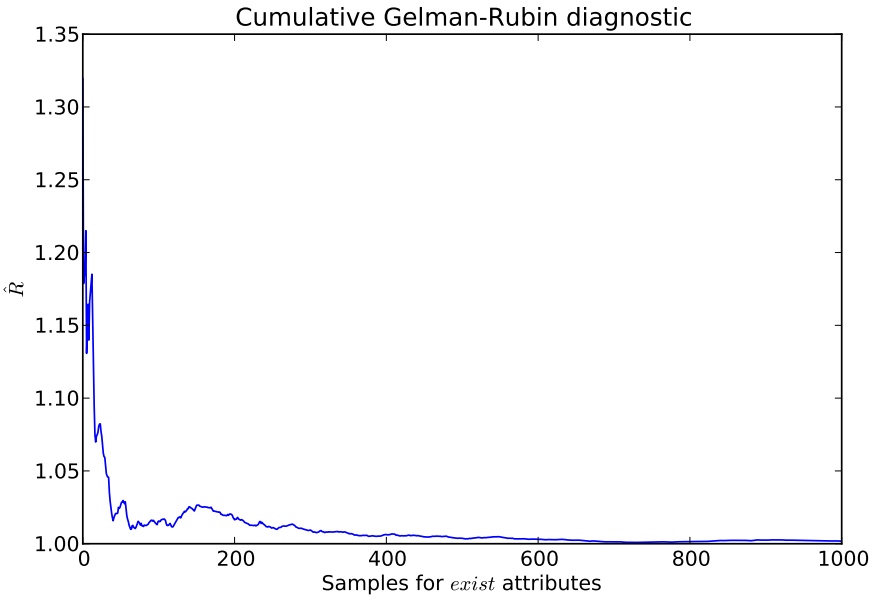


Figure 5.8: Gelman-Rubin convergence diagnostic, [GGRS96]

5.6 Related work

The problem we addressed is part of the broader category of *link mining*. [GD05] identifies several sub-types of link mining, including link prediction (which is our task) and gives examples of practical applications of such algorithms. [XD05] propose a reconciliation model for citation analysis that addresses both reference and identity uncertainty. It is based on constructing a dependency graph, whose edges are weighted by similarities between the vertices they tie. An iterative graph algorithm is used for inference. The model we propose would apply to the same type of task but its semantics are better defined.

Comparatively more work has been devoted to identity uncertainty (also known as entity matching). [SLD05] present a probabilistic, constraint-based approach; while the use of constraints is somewhat similar to our approach, their algorithm is more expensive in terms of computation cost. [BG07] propose a "collective" entity resolution scheme, where several articles (and their link information) are used together to figure out which objects correspond to the same

entity. This is essentially an inference problem that could be solved in our model; hence, we plan to extend our work to address such problems in the future.

Chapter 6

Software

In this chapter we present a framework called **P**rob**R**e**M**. This software was implemented as part of this thesis, and from the start it was designed to be general, modular and easy to extend. It available as open source package on `http://cs.mcgill.ca/~fkaeli/probrem/`. As this project combined scientific computing, complex data structures and relational databases, we used *python* as our programming language of choice. The straightforward nature of *python* itself allowed for fast prototyping, while the packages *numpy* [O+09], *scipy* [JOP01] offer efficient scientific libraries and *matplotlib* [Hun07] is convenient for visualization.

The **P**rob**R**e**M** framework allows the specification of a Directed Acyclic Probabilistic Entity Relationship (DAPER) model in an XML format. At the moment, only discrete variables are supported. The local distributions, i.e. the conditional probability distributions, can either be specified by hand or can be learned from data using a maximum likelihood estimate (MLE). Given a fully specified model, inference can be run using the MCMC algorithms presented in this thesis. Figure 6.1 shows an overview of the framework and the algorithms involved. Note that at this point, an expectation maximization (EM) algorithm for latent variables or missing data has not been implemented yet.

This chapter is organized as follows: Section 6.1 provides an overview of the modeling workflow when creating new models. In Section 6.2 we describe how the political domain model used in Chapter 4 can be implemented using **P**rob**R**e**M**.

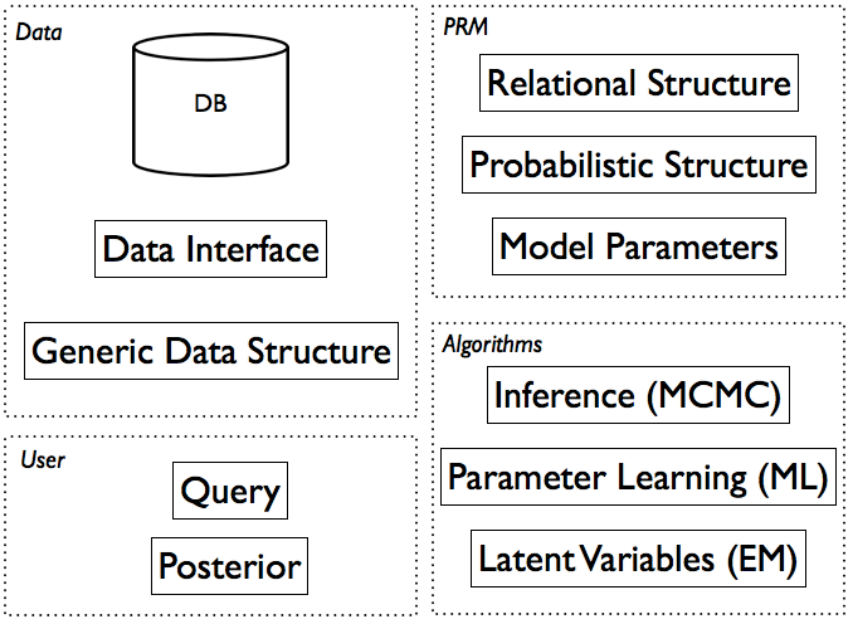


Figure 6.1: Overview of the framework structure

6.1 Modelling workflow

The illustration in Figure 6.1 shows the structure between the **P**rob**R**e**M** packages and modules. In this section we will go through the process of designing a new model which we call *PRMexample*. The different steps have also been separated, as best as possible, in the design of the software package. This separation guarantees well defined interfaces which allow for easy integration of future extensions or additions. For example, the unrolling of the ground Bayesian network should not depend on the type of relational database used. Similarly, the representation of a conditional probability distribution should be independent of a given inference algorithm.

A **P**rob**R**e**M** project is a python script which specifies a valid PRM, defines a data interface (the connection to a relational database) and configures the algorithms that will be used. For example, if the local distributions of the model have to be learned from data the script must define a module which implements a *CPDLearner*. The script must also configure the inference engine, which comprises the choice of inference algorithm (e.g. one of the MCMC algorithm described in the previous chapters). The directory ./ is defined to be the home directory of a

Prob**R**e**M** project.

Data

The data itself is stored in a separate relational database, and **P**rob**R**e**M** accesses the data using a *data.datainterface* which makes the choice of database irrelevant (theoretically). Currently **P**rob**R**e**M** supports the SQLite format which is implemented in *data.sqliteinterface.SQLiteDI*. All SQL databases have a very similar flavour, so the extension to other SQL formats is straight forward but untested. Assuming the data is stored in the database file *./data/database.sqlite*, a data interface specification is defined in XML and saved in the file *./DIexample.xml*.

```
<?xml version="1.0" ?>
<DataInterface name="DIexample">
    <Crossvalidation folds='1'>
            <Dataset type='SQLite' path='./data/database.sqlite'/>
    </Crossvalidation>
</DataInterface>
```

This is a simple example where just one data source is specified. It is also possible test the model using cross validation by specifying multiple data sources, in which case the data has to be split up on the database level. Otherwise the different folds would have to be accessed by querying one database which decreases the performance. The documentation of the XML parser for the data interface *xml_prm.parser.DataInterfaceParser* contains the specifications for the tags.

The *ground Bayesian Network* (GBN) is a generic data structure (a graph) that contains the data necessary to answer a given query. The GBN is stored in propositional form, as opposed to the first-order representation of the PRM; thus, only the subgraph which d-separates the full graph given the query is loaded. The *network.groundBN* module implements this data structure. The inference engine loads the GBN using the method *inference.engine.unrollGBN()*.

PRM specification

The PRM model itself is also specified in XML and saved in a file, for example *./PRMexample.xml*.

```
<?xml version="1.0" ?>
<PRM name="PRMexample"  datainterface="./DIexample.xml" >
    <RelationalSchema>
            <Entities>
                    <Entity name="A">
                            <Attribute name="Aa" type="Binary"/>
                    </Entity>
                    <Entity name="B">
                            <Attribute name="Ba" type="Integer" description="1,20"/>
                    </Entity>
            [......]
            </Entities>
            <Relationships>
                    <Relationship name="AB" foreign="A.pk,B.pk" type="1:n">
                            <Attribute name="ABa" type="Binary"/>
                    </Relationship>
                    [......]
            </Relationships>
    </RelationalSchema>
    <DependencyStructure>
            <Dependency name="Aa_Ba" child="A.Aa" parent="B.Ba" constraints="A.pk=B.pk" />
            [......]
    </DependencyStructure>
    <LocalDistributions>
            <LocalDistribution attribute='A.Aa' file='./localdistributions/Da_Aa.xml'/>
            <LocalDistribution attribute='B.Ba' file='./localdistributions/Ba_Aa.xml'/>
            <LocalDistribution attribute='AB.ABa' file='./localdistributions/Ca_Aa.xml'/>
    </LocalDistributions>
</PRM>
```

For a list of all possible tags as well as attributes, please see the documentation of the XML parser *xml_prm.parser.PRMparser* used by **P**rob**R**e**M**. The PRM is defined by the relational structure <*RelationalSchema*>, the probabilistic structure (<*DependencyStructure*>) and the model parameters (<*LocalDistributions*>; the conditional probability distributions of the attributes). If not all local distributions have been defined by the model they will have to be learned from data in order for the specification to be complete. Additionally, there is the functionality to save the local distributions to disk and load them for the next execution of the model.

Algorithms

Usually the local conditional probability distributions (CPDs) are learned from data. **P**rob**R**e**M** uses a Maximum Likelihood estimate (MLE) which is implemented in *CPDTabularLearner*. Currently only tabular CPDs are supported, *prm.localdistribution.CPDTabular*. In the future it might be necessary to implement alternative representations, for example decision trees. The learner instance can be configured to save the distributions to a file, the necessary files will be created automatically and the parser loads the distributions from disk if a corresponding file is available.

So far, all inference methods in **P**rob**R**e**M** are based on Markov Chain Monte Carlo methods. They have been thoroughly described in Chapters 4 and 5. MCMC algorithms in practice require a lot of fiddling around with parameters, e.g. burn in, number of samples collected, proposal distribution, convergence diagnostics. Depending on the type of query, different algorithms with different parameters are necessary. For this reason a **P**rob**R**e**M** project has to allow a flexible configuration of the inference method. More complex models may also require custom proposal distributions. The *inference* module offers the framework for MCMC inference; please refer to the documentation for details.

ProbReM Project

Given all the building blocks described so far, a simple python script is used to configure a **P**rob**R**e**M** project:

```
probremI = Probrem()

''' PRM '''
prmSpec = "PRMexample.xml"
probremI.prmI = config.loadPRM(prmSpec)

''' DATA INTERFACE '''
diSpec = probremI.prmI.datainterface
#diSpec = "DIexample.xml"
probremI.diI = config.loadDI(diSpec)
#configure datainterface with the prm instance
probremI.diI.configure(probremI.prmI)
```

Next, the local distributions are learned from data. If the probabilistic structure and the data do not change, the CPDs can be loaded from disk the next time the *PRMexample* model is instantiated.

```
''' LEARNERS '''
#we load a cpd learner to learn the CPDs for our attributes
probremI.learnersI['ourCPDlearner'] = config.loadLearner('CPDTabularLearner')
#we configure the learner to use the prm and data interface we instantiated
probremI.learnersI['ourCPDlearner'].configure(probremI.prmI,probremI.diI,learnCPDs=False)

probremI.learnersI['ourCPDlearner'].learnCPDsFull(saveDistributions=True,forceLearning=True)
```

After the model parameters are defined the inference method can be configured. The parameters are used by the engine to optimize inference, e.g. by precomputing the likelihood functions in the case of a Gibbs sampler.

```
''' INFERENCE ENGINE '''
probremI.inferenceI = config.loadInference('MCMC')
#we configure the engine to use the prm and data interface we instantiated
probremI.inferenceI.configure(probremI.prmI,probremI.diI)
```

Assuming that the script is saved in *./probremExample.py*, the model can now be used for queries by creating another script *./queryExample.py* which imports the model specification. A very simple example is given below:

```
from probremExample import *

# creating a query
exQuery = Query(event,evidence)
probremI.inferenceI.infer(exQuery)

# display the cumulative mean to test the convergence
posterior.cumulativeMean()
```

6.2 Example model

In Chapter 4 we presented results for a Gibbs inference algorithm using data of political campaign contributions. In this section we provide instructions on how to create a **P**rob**R**e**M** project

to make inference in this domain. For a detailed description of the model attributes, the relational structure and the probabilistic dependencies, please see section 4.3.2.

The data

As discussed in section 4.3.2, the data was obtained from Transparency Data. The data had to be preprocessed and stored in a format that is supported by **P**rob**R**e**M**. This step is not part of the framework. In this case for example, we excluded recipients that do not have the minimum number of contributions and to discretize the *amount* (we chose 20 bins). We stored the data in a SQLite database './data/policont.sqlite' based on the following schema:

```
-- ENTITIES
DROP TABLE IF EXISTS Recipient;
CREATE TABLE Recipient
(
recipient_id INTEGER PRIMARY KEY,
political_id INTEGER NOT NULL,
recipient_democratic INTEGER,
recipient_name CHAR(255) NOT NULL
);
DROP TABLE IF EXISTS Contributer;
CREATE TABLE Contributer
(
contributer_id INTEGER PRIMARY KEY,
industry INTEGER
);
DROP TABLE IF EXISTS Political;
CREATE TABLE Political
(
political_id INTEGER PRIMARY KEY,
state_name CHAR(255) NOT NULL,
income INTEGER,
democratic INTEGER
);
-- RELATIONSHIPS
DROP TABLE IF EXISTS donation;
CREATE TABLE donation
(
donation_id INTEGER PRIMARY KEY,
recipient_id INTEGER NOT NULL,
contributer_id INTEGER NOT NULL,
```

```
contributer_political_id INTEGER NOT NULL,
amount INTEGER NOT NULL,
FOREIGN KEY (recipient_id) REFERENCES Recipient(recipient_id),
FOREIGN KEY (contributer_id) REFERENCES Contributer(contributer_id),
FOREIGN KEY (contributer_political_id) REFERENCES Political(political_id)
);
```

Data interface specification

The specification file for the data interface, *./poliContDI.xml*, is straightforward. For this model we created different SQlite databases for the test and training set. Then the model parameters were learned using the training database and the same PRM model was loaded with test database for inference. In this simple training/test setting it is easier to do this by hand, i.e. by setting the *path* first to the training set, and then change it to the test set once the model is learned and parameters can be loaded from disk. Thus the number of *folds* is just 1 in this case.

```
<?xml version="1.0"?>
<DataInterface name="PoliContDI">
    <Crossvalidation folds='1'>
            <Dataset type='SQLite' path='./data/policont.sqlite'/>
    </Crossvalidation>
</DataInterface>
```

PRM specification

The PRM specification file './poliContPRM.xml' must use the same names as the database schema above.

```
<?xml version="1.0"?>
<PRM name="PoliContPRM"  datainterface="poliContDI.xml"  >
  <RelationalSchema>
    <Entities>
            <Entity name="Recipient">
                    <Attribute name="recipient_democratic" type="Binary"/>
            </Entity>
            <Entity name="Contributer">
                    <Attribute name="industry" type="Integer" description="1,115"/>
            </Entity>
```

```
            <Entity name="Political">
                    <Attribute name="democratic" type="Binary"/>
                    <Attribute name="income" type="Binary"/>
            </Entity>
    </Entities>
    <Relationships>
            <Relationship name="donation" foreign="Recipient.pk,Contributer.pk">
                    <Attribute name="donation_id" pk='1' type="NotProbabilistic"/>
                    <Attribute name="amount" type="Integer" description="1,20"/>
            </Relationship>
    </Relationships>
  </RelationalSchema>
  <DependencyStructure>
     <Dependency name="political_dem_amount" parent="Political.democratic" child="donation.amount"
        constraint="Political.political_id=donation.contributer_political_id"/>
     <Dependency name="income_amount" parent="Political.income" child="donation.amount"
      constraint="Political.political_id=donation.contributer_political_id"/>
     <Dependency name="recipient_dem_amount" parent="Recipient.recipient_democratic"
     child="donation.amount" constraint="Recipient.recipient_id=donation.recipient_id"/>
     <Dependency name="industry_amount" parent="Contributer.industry" child="donation.amount"
      constraint="donation.contributer_id=Contributer.contributer_id"/>
  </DependencyStructure>
  <LocalDistributions>
     <LocalDistribution attribute='Political.democratic'
     file='./localdistributions/democratic.xml'/>
     <LocalDistribution attribute='Political.income'
     file='./localdistributions/income.xml'/>
     <LocalDistribution attribute='Contributer.industry'
     file='./localdistributions/industry.xml'/>
     <LocalDistribution attribute='Recipient.recipient_democratic'
     file='./localdistributions/recipient_democratic.xml'/>
     <LocalDistribution attribute='donation.amount'
     file='./localdistributions/amount_recipient_democraticcontributer_id.xml'/>
  </LocalDistributions>
</PRM>
```

ProbReM Project

The python script './poliCont.py' initializes a **P**rob**ReM** project:

```
import sys
# interactive iPython console is used to interact with a probrem project
from IPython.Shell import IPShellEmbed
ipshell = IPShellEmbed()
```

```
# add the relative or absolute path to the 'Probrem/src' folder
sys.path.append("../../src")
#sys.path.append("/Users/xxxx/Documents/Projects/Probrem/src")

from probrem import Probrem
from ui import config

'''
the ProbReM instance
'''
probremI = Probrem()

''' PRM '''
prmSpec = "poliContPRM.xml"
probremI.prmI = config.loadPRM(prmSpec)

''' DATA INTERFACE '''
diSpec = probremI.prmI.datainterface
#diSpec = "poliContDI.xml"
probremI.diI = config.loadDI(diSpec)
# Configure data interface with the prm instance
probremI.diI.configure(probremI.prmI)

''' LEARNERS '''
# Load a cpd learner to learn the CPDs for our attributes
probremI.learnersI['ourCPDlearner'] = config.loadLearner('CPDTabularLearner')
# Configure the learner to use the prm and data interface we instantiated
probremI.learnersI['ourCPDlearner'].configure(probremI.prmI,probremI.diI,learnCPDs=False)
# For ease of use
ourCPDlearner = probremI.learnersI['ourCPDlearner']

# Learn all conditional probability distributions from data and save them to disk
ourCPDlearner.learnCPDsFull(saveDistributions=True,forceLearning=False)

''' INFERENCE ENGINE '''
#we load an inference engine
probremI.inferenceI = config.loadInference('MCMC')
#we configure the engine to use the prm and data interface we instantiated
probremI.inferenceI.configure(probremI.prmI,probremI.diI)
# for ease of use
mcmcInference = probremI.inferenceI


# uncomment if you want to interact with the model at this point
# ipshell()
```

Using the ProbReM project

We create a separate script for using the *poliCont* ProbReM project, e.g. ‘./poliContTesting.py’. Now we can interact with the model, for example the following code will display all model attributes and their CPDs.

```
# Load the ProbReM instance
import poliCont

for a in poliCont.probremI.prmI.attributes.values():

    print '%s, probabilistic=%s, Pa=%s'%(a.fullname,a.probabilistic,[pa.name for pa in a.parents])
    if a.probabilistic:
        print a.CPD.cpdMatrix
```

To infer the political affiliation of Rahm Emanuel for example, create the following query.

```
# Load the ProbReM instance
import poliCont

from inference.query import Query, createQvar,ObjsVariable
from inference import posterior

recipient = "Rahm Emanuel (D)"
rec_pk = static_data.recipients[recipient] # a dictionary mapping politians to their primary key


objs = ObjsVariable('incl', rec_pk)
event = [ createQvar('Recipient.recipient_democratic',objs)]

objsAll = ObjsVariable('excl', [])
evidence = [    createQvar('donation.amount',objsAll,None),
                createQvar('Contributor.industry',objsAll,None),
                createQvar('Political.democratic',objsAll,None),
                createQvar('Political.income',objsAll,None)]

query = Query(event,evidence)

# Runs one chain using the current settings in `inference.mcmc`
poliCont.probremI.inferenceI.infer(query)

# Check the convergence using the cumulative mean
```

```
posterior.cumulativeMean()

# Print the mean of the posterior distribution
print posterior.mean()[0]

# The log likelihood of Rahm Emanuel
loglik = poliCont.probremI.inferenceI.GBN.logLikelihood()
# The size of the GBN required for inference on Rahm Emanuel
sizeGBN = len(poliCont.probremI.inferenceI.GBN)

# Interactive iPython session for further analysis
poliCont.ipshell()
```

All convergence diagnostic plots in this thesis have been created using methods that are built into **P**rob**R**e**M**. Most of these methods automatically create visualizations of the output. For a complete overview and documentation, please see the *inference.mcmc.posterior* module. It is recommended to work in an interactive environment, for example *ipython*, as models can be loaded and changed on the fly, inference can be interrupted to monitored convergence, and the workflow can be saved to a file for later duplication of the results.

Chapter 7

Conclusion and Future Work

In this thesis, we have presented two inference algorithms for Probabilistic Relational Models. In Chapter 4 we introduced an inference method based on Gibbs sampling which makes use of the relational structure of the ground Bayesian network to find an efficient sampling order. We have shown that this algorithm is fast and scalable on both artificial and real data. In Chapter 5 we proposed a new approach to model reference uncertainty in PRMs. Expectation aggregation is a novel method to handle aggregation, which removes the need for specifying aggregation methods and provides an intuitive interpretation for the constraints of the probabilistic dependencies. An Metropolis-Hastings algorithm allows for an efficient inference algorithm that exploits the sparsity of the domain. In Chapter 6 we describe **P**rob**R**e**M**, an open source framework for modeling PRMs which implements the proposed algorithms.

The next step will be to validate the algorithms more extensively. In experiments on large data sets with a rich relational structure, we have to show that the presented algorithms continue to display high accuracy in terms of inference and that they scale well, i.e. that the speed of convergence remains acceptable. To accurately model a richer relational structure of the data, the use of hidden variables would allow the specification of more expressive models. Real world data is often continuous, but currently only discrete probabilities are implemented. The software framework needs to be extended to support both hidden and continuous variables.

This thesis leads to a couple of interesting ideas for future work. Most importantly, we

plan to apply the proposed approach for reference uncertainty in Chapter 5 to citation analysis, which is one of the applications of link mining methods [LKM+07]. We expect our algorithm to perform well on this sort of task because it benefits directly from the typical sparsity of link mining domains. Furthermore, with minor changes to the model definition we are able to extend our inference method to identity uncertainty, another form of relational uncertainty in which two objects in the relational skeleton could point to the same real world object. A separate issue, only marginally discussed in this thesis, is the problem of feature selection. So far, we assume that all variables are valuable components of the model. Our approach to reference uncertainty, combined with expectation aggregation, gives us a new way of assigning greater impact to some variables. Little work has been done on feature selection in the setting of relational data, which makes this field interesting for future research.

References

[BG07] I. Bhattacharya and L. Getoor, *Collective entity resolution in relational data*, ACM Transactions on Knowledge Discovery from Data **1** (2007), no. 1, 1–36. 51

[FGKP99] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, *Learning probabilistic relational models*, International Joint Conferences on Artificial Intelligence (IJCAI), vol. 16, 1999, pp. 1300–1309. 16

[GD05] L. Getoor and C. Diehl, *Link mining: A survey*, SigKDD Explorations Special Issue on Link Mining **7** (2005), no. 2. 51

[Get00] L. Getoor, *Learning probabilistic relational models*, International Symposium on Abstraction, Reformulation, and Approximation (SARA), Springer-Verlag, 2000, pp. 322–323. 27, 41

[GGRS96] W.R. Gilks, W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, *Markov chain Monte Carlo in practice*, Chapman & Hall/CRC, 1996. ix, 11, 50, 51

[GT07] L. Getoor and B. Taskar, *Introduction to statistical relational learning (adaptive computation and machine learning)*, The MIT Press, 2007. 18, 24

[HMK04] D. Heckerman, C. Meek, and D. Koller, *Probabilistic models for relational data*, Tech. Report MSR-TR-2004-30, Microsoft Research, 2004. 13, 16

[Hun07] J.D. Hunter, *Matplotlib: A 2d graphics environment*, Computing in Science & Engineering (2007), 90–95. 53

[Jae97] M. Jaeger, *Relational bayesian networks*, Uncertainty in Artificial Intelligence (UAI), 1997, pp. 266–273.

[JOP01] E. Jones, T. Oliphant, and P. Peterson, *Scipy: Open source scientific tools for python*, http://www. scipy. org/ (2001). 53

[KF09] D. Koller and N. Friedman, *Probabilistic graphical models: Principles and techniques*, MIT Press, 2009. 4, 8, 20

[KP09] J. Kisynski and D. Poole, *Lifted aggregation in directed first-order probabilistic models*, International Joint Conferences on Artificial Intelligence (IJCAI), vol. 9, 2009, pp. 1922–1929.

[KP10] F. Kaelin and D. Precup, *An approach to inference in probabilistic relational models using block sampling*, Asian Conference on Machine Learning (ACML), 2010, pp. 325–340. 3

[KSS80] R. Kindermann, J.L. Snell, and American Mathematical Society, *Markov random fields and their applications*, American Mathematical Society, 1980. 10

[LKM+07] D. Lee, J. Kang, P. Mitra, C. L. Giles, and B. On, *Are your citations clean?*, Communications of the ACM (CACM) **50** (2007), no. 12. 66

[MTB10] Wannes Meert, Nima Taghipour, and Hendrik Blockeel, *First-order Bayes-ball*, Machine Learning and Knowledge Discovery in Databases, European Conference (ECML PKDD), Springer, 2010, pp. 369–384. 20

[MZK+08] B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L.P. Kaelbling, *Lifted probabilistic inference with counting formulas*, Association for the Advancement of Artificial Intelligence (AAAI), 2008.

[O+09] T. Oliphant et al., *Numpy, a python library for numerical computations*, 2009. 53

[Pea88] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann, 1988. 9, 26

[PK00] A.J. Pfeffer and D. Koller, *Semantics and Inference for Recursive Probability Models*, Association for the Advancement of Artificial Intelligence (AAAI), 2000, pp. 538–544. 22

[PR01] H. Pasula and S. Russell, *Approximate inference for first-order probabilistic languages*, International Joint Conferences on Artificial Intelligence (IJCAI), 2001. ix, 39, 40, 41

[RD06] M. Richardson and P. Domingos, *Markov logic networks*, Machine Learning **62** (2006), no. 1, 107–136. 22

[Sha98] R.D. Shachter, *Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams)*, Uncertainty in Artificial Intelligence (UAI), vol. 14, 1998, pp. 480–487. 20

[SLD05] W. Shen, X. Li, and A. Doan, *Constraint-based entity matching*, Association for the Advancement of Artificial Intelligence (AAAI), vol. 20, 2005, p. 862. 51

[TAK02] B. Taskar, P. Abbeel, and D. Koller, *Discriminative probabilistic models for relational data*, Uncertainty in Artificial Intelligence (UAI), vol. 18, 2002, pp. 895–902. 22

[XD05] J. Madhavan X. Dong, A. Y. Halevy, *Reference reconciliation in complex information spaces*, Special Interest Group on Management Of Data (SIGMOD), 2005, pp. 85–96. 51

[ZP96] N.L. Zhang and D. Poole, *Exploiting causal independence in bayesian network inference*, Journal of Artificial Intelligence Research (JAIR), no. 5, 1996, pp. 301–328. 17, 22