DIRECTED MODELS FOR STATISTICAL RELATIONAL LEARNING

by

Hassan Khosravi

M.Sc., AmirKabir University of Technology, 2007

B.Sc, Shahid Bahonar University, 2005

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2012

APPROVAL

Name: Hassan Khosravi

Degree: Doctor of Philosophy

Title of Thesis: Directed Models for Statistical Relational Learning

Examining Committee: Dr. Anoop Sarkar,Associate Professor, Computer Science
Chair

Dr. Oliver Schulte, Associate Professor, Computing Science
Senior Supervisor

Dr. Martin Ester, Professor, Computing Science
Supervisor

Dr. Lise Getoor, Associate Professor, Computer Science
University of Maryland
External Examiner

Dr. James Delgrande Professor, Computing Science
Internal Examiner

Date Approved: 12 October 2012

Partial Copyright Licence

Abstract

Statistical Relational Learning is a new branch of machine learning that aims to model a joint distribution over relational data. Relational data consists of different types of objects where each object is characterized with a different set of attributes. The structure of relational data presents an opportunity for objects to carry additional information via their links and enables the model to show correlations among objects and their relationships. This dissertation focuses on learning graphical models for such data. Learning graphical models for relational data is much more challenging than learning graphical models for propositional data. One of the challenges of learning graphical models for relational data is that relational data, unlike propositional data, is non independent and identically distributed and cannot be viewed in a single table. Relational data can be modeled using a graph, where objects are the nodes and relationships between the objects are the edges. In this graph, there may be multiple edges between two nodes because objects may have different types of relationships with each other. The existence of multiple paths of different length among objects makes the learning procedure much harder than learning from a single table. We use a lattice search approach with lifted learning to deal with the multiple path problem. We focus on learning the structure of Markov Logic Networks, which are a first order extension of Markov Random Fields. Markov Logic Networks are a prominent undirected statical relational model that have achieved impressive performance on a variety of statistical relational learning tasks. Our approach combines the scalability and efficiency of learning in directed relational models, and the inference power and theoretical foundations of undirected relational models. We utilize an extension of Bayesian networks based on first order logic for learning class-level or first-order dependencies, which model the general database statistics over attributes of linked objects and their links. We then convert this model to a Markov Logic Network using the standard moralization procedure. Experimental

results indicate that our methods are two orders of magnitude faster than, and predictive metrics are superior or competitive with, state-of-the-art Markov Logic Network learners.

To my parents for their love and support.

Life is a process of becoming, a combination of states we have to go through. Where people fail is that they wish to elect a state and remain in it. –Anais Nin

Acknowledgments

Doing a Ph.D. is a long journey and is only possible with the supervision of my committee members and moral and physical support of family, my partner, and friends.

First of all I would like to express my deepest gratitude to my senior supervisor, Dr. Oliver Schulte, for his excellent guidance, kindness, patience throughout my Ph.D. He has helped me develop many skilled during our collaboration. I would never have been able to finish my dissertation without his guidance and support. I would also like to thank my supervisor, Professor Martin Ester, for his guidance, support and willingness to help. I am grateful to Dr. Lise Getoor, Dr. Jim Delgrande, and Dr. Anoop Sarkar for willing to participate in my final defense as external examiner, internal examiner, and chair.

I would like to thank my wife Rozita for all her love and support. All of the nights we spend together studying, and her encouragements was in the end what made this dissertation possible. I would also like to thank my parents, two sisters Sara and Rana, and brother in law Reza. They were always very supportive and encouraged me with their best wishes.

Special thanks go to all of the collaborators, other students, that I had the honor of working with during my Ph.D. Bahareh Bina, Ali Bozorgkhan, Tong Man (Mike), Xianoyouan Xu(Vivian), Jianfeng Hu, Tianxiang Gao, and Yuke Zhu have all been super collaborators.

Finally, I would like to thank all the friends that have been there for me during my stay at Simon Fraser University. My roommates, Mohammad, Ehsan, Alireza, and Bamdad for putting up with me, and of all other friends Majid, Maryam, Nima, Bardia, Arina, Maryam, Esmaeil, Kouhyar, Hossein, Shahab, Hamed, Kamyar, and Amir for the coffees, swimming, tennis, racketball, shelem and xbox time that we spend together.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many databases store data in a relational format, with different types of entities and information about links among the entities. The field of statistical-relational learning (SRL) has developed a number of new statistical models for relational databases [33]. Markov Logic Networks (MLNs) form one of the most prominent SRL model classes; they generalize both first-order logic and Markov network models [15]. MLNs have achieved impressive performance on a variety of SRL tasks. Because they are based on undirected graphical models, they avoid the difficulties with cycles that arise in directed SRL models [79, 15, 105]. Essentially, an MLN is a set of weighted first-order formulas that compactly defines a Markov network comprising of ground instances of logical predicates. The formulas are the structure or qualitative component of the Markov network; they represent associations between ground facts. The weights are the parameters or quantitative component; they assign a likelihood to a given relational database by using the log-linear formalism of Markov networks. An open-source benchmark system for MLNs is the Alchemy package [61].

This dissertation addresses structure learning for MLNs in relational schemas that feature a significant number of descriptive attributes compared to the number of relationships. Previous MLN learning algorithms do not scale well with such datasets. We introduce a new *moralization approach* to learning MLNs: first we learn a directed Bayes net graphical model for relational data, then we convert the directed model to an undirected MLN model using the standard moralization procedure (marry spouses, omit edge directions)[15]. The main motivation for performing inference with undirected models is that converting a Bayes net to an undirected model avoids the cyclicity problem [15, 105, 53]. Thus our

approach combines advantages of both directed and undirected SRL models: learning efficiency and interpretability from directed models, and the solutions to the combining and cyclicity problems together with the inference power of undirected models.

1.1 Approach

We address three closely related research problems in this dissertation.

(1) In Chapter 3, we present a new algorithm for learning a non-recursive Bayes net from relational data, the *learn-and-join* algorithm. This algorithm performs a level-wise model search through the table join lattice associated with a relational database, where the results of learning on subjoins are propagated as constraints for learning on larger joins. This propagation mechanism leads to longer and more informative MLN clauses. A single-table Bayes net learner, which can be chosen by the user, is applied to each join table to learn a Bayes net for the dependencies represented in the table. For joins of relationship tables, this Bayes net represents dependencies among attributes conditional on the existence of the relationships represented by the relationship tables. The Bayes net is then converted to an MLN for inference using moralization. In our experiments on small datasets, the run-time of the learn-and-join algorithm is 200-1000 times faster than benchmark programs in the Alchemy framework [59] for learning MLN structure. On medium-size datasets, almost none of the Alchemy systems return a result given our system resources, whereas the learn-and-join algorithm produces an MLN structure within a few minutes. The structure of MLNs that are learned through moralization are more complex and use a combination of variables and constants, whereas other state-of-the-art MLN methods tend to learn a much sparser structure that only has variables. A preliminary version of this research was published in the proceedings of the Association for the Advancement of Artificial Intelligence (AAAI2010) conference [53]. Later a more detailed version was published in the Journal of Machine Learning [94]

(2) Chapter 4 focuses on recursive dependencies in relational data. A key phenomenon that distinguishes relational data from single-population data is that the value of an attribute for an entity can be predicted by the value of the same attribute for related entities; these are called recursive dependencies. For example, whether individual a smokes may be predicted by the smoking habits of a's friends. Conversion of Bayes nets to undirected models avoids cyclicity while performing inference; however, it still remains a problem for model selection

during the learning phase. The cycles make it difficult to define a model likelihood function for observed ground facts in the data, which is an essential component of model selection techniques. To define a model likelihood function for a Bayes net search, we utilize a recent relational Bayes net pseudo likelihood [93] that measures the fit of a Bayes net to a relational database and is well-defined even in the presence of recursive dependencies. To show recursive dependencies using this pseudo likelihood, additional copies of variables participating in the recursive dependencies are added to the graphical model. For example a second variable to show the smoking habits of users is added to the graphical model. This pattern can be represented by a clausal notation such as $Smokes(X) \leftarrow Smokes(Y), Friend(X, Y)$. If each replicated predicate is treated as a separate random variable, then multiple redundant edges showing the same dependency will be added, as the logical variables are interchangeable placeholders for the same domain of entities. In this chapter we propose a normal form for Bayes nets that are learned from relational data to eliminate such redundancies and prove that this constraint incurs no loss of expressive power. A preliminary version of this research was published in the proceedings of the Inductive Logic program (ILP2011) conference [97]. Later a more detailed version was published in the Journal of Machine Learning [98]

(3) In Chapter 5 we learn compact Markov Logic Networks using decision trees. A weakness of the moralization approach is that it leads to an unnecessarily large number of clauses. The Bayes net method learns dependencies among predicates, not literals, which fail to capture local or context-sensitive independencies. MLNs have one weight parameter for each clause, which decreases the accuracy of parameter estimates, and slows down inference. In this chapter, we show that using decision trees to represent conditional probabilities in the Bayes net is an effective remedy that leads to much more compact MLN structures. The decision trees can be learned using standard propositional decision tree learners. In experiments on benchmark datasets, the decision trees reduce the number of clauses in the moralized MLN by a factor of 5 to 25, depending on the dataset. The accuracy of predictions is competitive with the unpruned model and in many cases superior. A preliminary version of this research was published in the proceedings of the Inductive Logic program (ILP2011) conference [51]. Later a more detailed version was published in the Journal of Machine Learning [52]

1.2 Contributions

The main contributions of this dissertation are the following:

1. A new structure learning algorithm, learn-and-join, for Bayes nets that models the distribution of descriptive attributes given the link structure in a relational database with discussion and justification for some relational constraints that speed up structure learning. The algorithm is a level-wise lattice search through the space of join tables.
2. A new normal form theorem for Bayes nets that addresses redundancies in modelling recursive dependencies.
3. An extension of the learn-and-join algorithm for learning Bayes nets from relational data that include autocorrelations.
4. A new structure learning algorithm augmenting Bayes net learning with decision tree learning to learn a compact set of clauses, a Markov Logic Network, for relational data.
5. A comparison of all of our proposed methods with other state-of-the-art methods.

1.3 Limitations

The main limitations of the work presented in this dissertation are the following:

1. Our current algorithms do not find associations among relationships. For instance, if a professor advises a student, then they are likely to be coauthors. In the terminology of Probabilistic Relational Models [30], our model addresses attribute uncertainty, but not existence uncertainty (concerning the existence of links).
2. In this dissertation, we only focus on learning the structure, and we use the default Markov Logic Network method provided in the Alchemy package for parameter learning. While this parameter learning method finds suitable parameter settings, it is slow and constitutes the main computational bottleneck for our approach. In our experiments, 98% of the time for learning is spent optimizing the parameters for the learned structure. We propose a moralization approach to parameter estimation where MLN weights are directly inferred from Bayes net parameters. Empirical evaluation

indicates that parameter estimation via moralization is orders of magnitude faster than parameter optimization, while performing as well or better on prediction metrics. This research is not included in this dissertation, and appears in the proceedings of the Inductive Logic Programming conference [49].

3. In this dissertation, we do not discuss any industrial applications for the learn-and-join algorithm; the learned Bayes nets are always converted to Markov Logic Networks for inference on the ground model. The standard grounding semantics is appropriate for answering queries about individual ground facts ("What is the probability that Tweety can fly?") but not appropriate for answering frequency queries ("What is the probability that an arbitrary bird can fly?"), because frequency queries concern generic events. Frequency queries may be used for query optimization, policy making and strategic planning, and finding statistical first-order patterns. To apply Bayes nets to such queries we propose a frequency semantics for Bayes nets, based on Halpern's classic domain frequency semantics for probabilistic first-order logic [36]. Learning the parameters for such a frequency semantics can be done using our learn-and-join algorithm which like Halpern's semantics is based on random instantiations of first-order variables. A naive computation of the empirical frequencies of the relations is intractable due to the complexity imposed by negated relational links. We render this computation tractable by using the fast Moebius transform. This research is not included in this dissertation and appears in the proceedings of the Inductive Logic Programming conference [96].

Chapter 2

Background and Literature Review

In Section 2.1, we provide the necessary background on fields related to this dissertation. The rest of this chapter provides a literature review and a running example on some of the state-of-the-art models in statistical relational learning; the methods of representing, learning, and inference are discussed. In Section 2.2, we discuss Probabilistic Relational Models; their graphical model is very similar to that of ours. In Section 2.3 we discuss Markov Logic Networks; the main contributions of this dissertation are learning methods for learning Markov Logic Networks. In Section 2.4 we discuss Relational Dependency Networks and in Section 2.5 we discuss Bayesian Logic Programs. We conclude this chapter with Section 2.6 which addresses the limitations of current models and provides a comparison among them.

2.1 Background and Notation

In this section we introduce some notation and terminology that is referred to throughout this dissertation. Please consider reading the citations in case you need more familiarity with any of the topics discussed. We presents a brief discussion on probability, logic, machine learning, Bayesian networks, decision trees, relational data, and relational models in this section.

2.1.1 Probability Distribution

We assume general familiarity with probability theory. We denote **random variables** by capital letters (X,Y,...) and the values of a random variable by lower case letters (x,y,...). We denote sets of random variables by boldface capital letters $\mathbf{X}=\{X_1, \ldots, X_n\}$.

The set of values of a variable is the **range** of the variable. In this dissertation we only consider random variables with a *finite range*. We write $P(X_1 = x_1, ..., X_n = x_n) = p$, sometimes abbreviated as $P(x_1, ..., x_n) = p$, or $P(\mathbf{X} = \mathbf{x}) = p$, to denote that the **joint probability** of random variables $X_1, \ldots, X_n$ taking on values $x_1, \ldots, x_n$ is p. The sum of the joint probability distribution of a set of random variables $\mathbf{X}$ over all possible values $\mathbf{x}$ is 1, that is,

$$\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) = 1$$

The joint probability distribution of a subset $\mathbf{Y}$ of $\mathbf{X}$ can be obtained by summing out the remaining variables $\mathbf{Z} = \mathbf{X} \backslash \mathbf{Y}$. The distribution is called the **marginal probability** distribution of $\mathbf{Y}$.

$$P(\mathbf{Y}) = \sum_{\mathbf{z}} P(\mathbf{Y}, \mathbf{Z} = \mathbf{z})$$

The conditional probability of a subset $(Y) \in (X)$ given a subset $\mathbf{Z} \in (Z)$ is the probability of $\mathbf{Y}$ occurring when we know $\mathbf{Z}$ has occurred. The **conditional probability** can be obtained by the following formula:

$$P(\mathbf{Y}|\mathbf{Z}) = \frac{P(\mathbf{Y}, \mathbf{Z})}{P(\mathbf{Z})}$$

Two events $\mathbf{X}$ and $\mathbf{Y}$ are *independent* if occurrence of $\mathbf{X}$ does not effect the probability of $\mathbf{Y}$ occurring. In such a event, the joint probability of the two events co-occurring is calculated by the product of each of their occurrence:

$$P(\mathbf{X}, \mathbf{Y}) = P(\mathbf{X}) \times P(\mathbf{Y})$$

2.1.2 Logic

We assume general familiarity with propositional and first order logic. In this dissertation, a **term** τ is a constant or a variable. A **functor** is a function symbol. Each functor has a set

of values (constants) called the **range** of the functor. A functor whose range is $\{T, F\}$ is a **predicate**, usually written with uppercase letters like P, R. A **functor random variable** is of the form $f(\tau_1, \ldots, \tau_k)$ where f is a functor and each τ_i is a term. We also refer to functor random variables as **functor nodes**, or for short **fnodes**.[1] Unless the functor structure matters, we refer to a functor node simply as a node. If functor node $f(\boldsymbol{\tau})$ contains no variable, it is **ground**, or a **gnode**. If functor node contains a variable, it is a **vnode**.

An assignment of the form $f(\boldsymbol{\tau}) = x$, where x is a constant in the range of f, is a **literal**; if $f(\boldsymbol{\tau})$ is ground, the assignment is a **ground literal**. A **population** is a set of individuals, corresponding to a domain or type in logic. Each first-order variable X is associated with a population $\mathcal{P}_X$ of size $|\mathcal{P}_X|$; in the context of Functor Bayes nets, we refer to **population variables** [84]. An **instantiation** or **grounding** γ for a set of variables $X_1, \ldots, X_k$ assigns a constant $\gamma(X_i) = x_i$ from the population of X_i to each variable X_i.

2.1.3 Machine Learning

Machine learning has become a crucial part of computer science; as a broad subfield of artificial intelligence, machine learning is concerned with the design and development of algorithms and techniques that, given a dataset $\mathcal{D}$ and an objective function $f(.)$, search through the hypothesis space or models $\mathcal{M}$ for the most suitable candidate $m \in \mathcal{M}$ such that $f(m, \mathcal{D})$ is maximized.

A variety of machine learning tasks may be distinguished by the characteristics of the hypothesis space (models) considered and the objective function. In this dissertation we address the following.

- *Discriminative Learning* is a class of learning in which the objective function $f(.)$ is defined as to maximize the predictive performance of the model on a target attribute T, for unseen data. The output is usually a conditional probability distribution $P(T|\mathbf{E})$ over the range of the values of T where $\mathbf{E}$ is the evidence information from other attributes. Decision Trees, discussed in Sec 2.1.5, are a well known model for discriminative learning.

- *Generative Learning* is a class of learning that aims to model and describes the patterns

[1] The term "functor" is used as in Prolog [7]. In Prolog, the equivalent of a functor random variable is called a "structure". Poole [84] refers to a functor random variable or fnode as a "parametrized random variable". We use the term fnode for brevity.

and regularities in the data. The output is usually a probability distribution over the random variables in the model. Bayesian Networks, discussed in Section 2.1.4, are a well known model for generative learning.

2.1.4 Bayesian Networks

We employ notation and terminology from [83, 101] for a Bayesian Network. We consider Bayes Nets for a set of variables $\mathbf{V} = \{X_1 \dots X_n\}$ where each X_i has a finite range. **A Bayes net structure** $G = (\mathbf{V}, \mathbf{E})$ for a set of variables $\mathbf{V}$ is directed accylic graph (DAG). A **Bayes net** (BN) is a pair $\langle G, \boldsymbol{\theta}_G \rangle$ where $\boldsymbol{\theta}_G$ is a set of parameter values that specify the probability distributions of children conditional on assignments of values to their parents. The conditional probabilities are specified in **conditional probability tables**.

If X, Y are two variables and S is a set of variables disjoint from $\{X, Y\}$, then S d-separates X and Y if along every path between X and Y there is a node w satisfying one of the following conditions: (1) w is a collider on the path and none of w nor any of its descendants are in S, or (2) w is not a collider on the path and w is in S. We write $(X \perp\!\!\!\perp Y|S)_G$ if X and Y are d-separated by S in graph G. If two nodes X and Y are not d-separated by S in graph G, then X and Y are **d-connected** by S in G, written $(X \not\perp\!\!\!\perp Y|S)_G$. If $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ are three disjoint sets of variables, then $\mathbf{Z}$ d-separates $\mathbf{X}$ and $\mathbf{Y}$ if for all variables $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$, the set $\mathbf{Z}$ d-separates X and Y.

For a node X, we refer to the set of its parents, children and co-parents (i.e. other parents of its children) as **the Markov blanket** of X, $MB(X)$. Given its Markov blanket, each node X in G is d-separated from all other nodes outside of the Markov blanket. We refer to the set of independences $\{X \perp\!\!\!\perp Y|MB(X) : Y \notin MB(X)\}$ as the **set of Markov blanket independences** for graph G.

Figure 2.1 is a well known example of a Bayesian network from [92]. In the example, the event of grass being wet has two possible causes: either the water sprinkler is on, or it is raining. Cloudy weather causes rain and reduces the probability of sprinkler being on. The strength of the relationships are given in the conditional probability tables.

Learning Bayesian Networks

Given a set of variables V and a dataset $\mathcal{D}$ containing independent and identically distributed (i.i.d) examples from an unknown distribution P, the goal of structure learning is

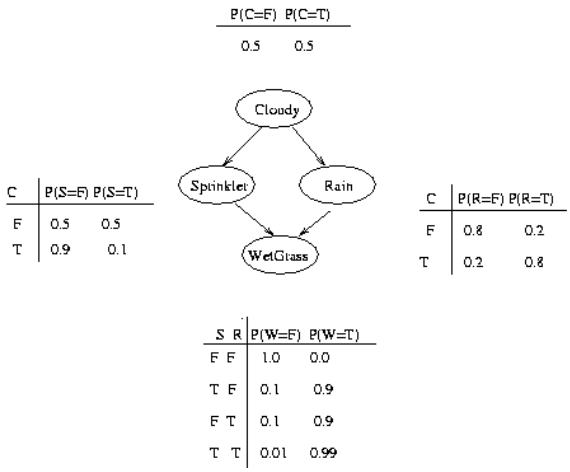


Figure 2.1: An example of a Bayesian network taken from [92].

to identify a directed acyclic graph G that represents P. Finding an optimal structure is a computationally intractable problem; structure learning algorithms determine for every possible edge in the network whether or not to include the edge in the final network and which direction to orient the edge. The total possible number of graphs is super exponential in $|V|$. Even a restricted form of structure learning where variables are constrained to have at most k parents has been proven to be NP-Complete. Two broad classes of structure learning are well-known in the literature [76, 37].

- Score Based methods search over possible Bayesian network structures for the most suitable factorization of the joint probability based on $\mathcal{D}$. The model selection criterion is usually defined as a score that the methods are trying to maximize. BDeu [38] and BIC [11] are examples of well known scores for Bayesian network learning.

- Constraint Based methods employ a statistical tests to detect conditional (in)dependencies given a sample $\mathcal{D}$, and then compute a BN structure G that fits the (in)dependencies [68, 10]. The (in)dependencies test can be chosen to suit the type of available data and application domain. One of the traditional test for categorical data is the χ^2 test.

To form a complete Bayes net, an additional step of parameter estimation is required to determine θ_G from $\mathcal{D}$. Since the focus of this dissertation is on structure learning, techniques for parameter estimation will not be addressed here. (See e.g. [37] for details).

Parametrized Bayes Nets

Parametrized Bayes nets(PBNs) were introduced by Poole to study first-order probabilistic inference on directed models. They are a comparatively simple adaptation of the Bayes net format for relational data. The syntax of PBNs is similar to that of other directed graphical SRL models, such as Bayes Logic Programs [46] and Probabilistic Graphical Models [26]. A Parametrized Bayes Net structure consists of: (1) a directed acyclic graph whose nodes are functor nodes. (2) a population for each first-order variable. (3) an assignment of a range to each functor. A Parametrized Bayes Net is a Bayes net whose graph is a parametrized Bayes Net structure. Because each of the nodes in a parametrized Bayes net is a functor, We also use the name Functor Bayes Nets (FBNs) for them.

2.1.5 Decision Trees

A decision tree is a discriminative method for predicting a target attribute using a tree. We consider decision trees for a set of variables $\mathbf{V} = \{X_1 \dots X_n, T\}$ where T is the target attribute and each X_i and T have a finite range. An instance $\{x_1 \dots x_n, t\}$ is classified by starting at the root node of the tree. Each X_i nodes in the tree specifies a test on the value x_i; a branch is selected based on x_i. This process is then repeated for the subtree rooted at the next node. Each branch is terminated with a value t in the range of T which is the classification value. Figure 2.2 is a well known example of a decision tree from [71]. The model decides if the weather is amenable for playing tennis. It illustrates that in case of sunny outlook and normal humidity tennis will be played and in case of rainy outlook and strong wind, tennis will not be played.

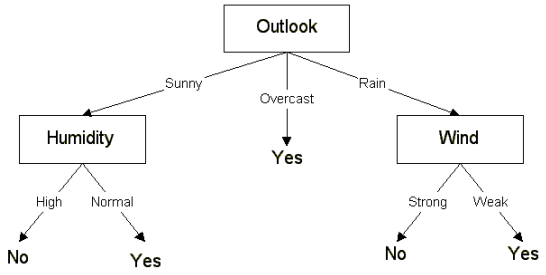


Figure 2.2: An example of a decision tree taken from [71].

Learning Decision Trees Most algorithms that have been developed for learning decision trees are variations of the ID3 algorithm [89] by Quinlan that employ a top down greedy search through the space of possible trees. As in any greedy algorithm, a definition for the best choice is required; each of the attributes, on its own, is used to build a classifier of depth one, and is evaluated on the training. This is equivalent to selecting the attribute with the smallest entropy measure on the training set. The best attribute X_i is selected and used as the root node of the tree. A descendant node of the root is generated for each x_i. This process is repeated until an ending criterion is met.

2.1.6 Relational Data and Databases

The majority of work in learning has focused on data which consists of identically structured entities that are assumed to be independent. However, many real world datasets are relational and most real world applications are characterized by the presence of uncertainty and complex relational structure; statistical learning is based on the former and relational learning is based on the latter.

Statistical relational learning is a new branch of machine learning that aims to model a joint distribution over relational data. Relational data are more complex and better suited where examples are given as multiple related tables. The structure of relational data presents an opportunity for objects to carry additional information via their links and enables the model to show correlations among entities and their relationships [33].

We assume general familiarity with relational databases [107]. We use standard relational schema containing a set of tables, each with key fields, descriptive attributes, and foreign keys. A **table join** of two or more tables contains the rows in the Cartesian products of the tables whose values match on common fields. If the schema is derived from an entity-relationship model (ER model) [107, Ch.2.2], the tables in the relational schema can be divided into *entity tables* and *relationship tables*. Table 2.1 shows a example of a university relational schema. The entity types of schema of Table 2.1 are students, courses and professors. There are two relationship tables: *Registration* records courses taken by each student and the grade and satisfaction achieved, and *RA* records research assistantship contracts between students and professors. Relationships refer to their related entities using *reference slots*. Each table in the relational database is considered as a class with some descriptive attributes.

A **database instance** specifies the tuples contained in the tables of a given database

$Student(\underline{student\_id}, intelligence, ranking)$
$Course(\underline{course\_id}, difficulty, rating)$
$Professor\ (\underline{professor\_id}, teaching\_ability, popularity)$
$Registered\ (\underline{student\_id}, \underline{course\_id}, grade, satisfaction)$
$Teaches(\underline{professor\_id}, \underline{course\_id})$

Table 2.1: A relational schema for a university domain. Key fields are underlined.The schema has three entities and two relationships

schema. Figure 2.3 illustrates a database instance for schema in Table 2.1

Student

s-id	Intelligence	Ranking
Jack	3	1
Kim	2	1
Paul	1	2

Professor

p-id	Popularity	Teaching-ability
Oliver	3	1
Jim	2	1

Course

c-id	Rating	Difficulty
101	3	1
102	2	2

Registration

s-id	C.nr	Grade	Satisfaction
Jack	101	A	1
Jack	102	B	2
Kim	102	A	1
Paul	101	B	1

Teaches

P-id	C-id
Oliver	101
Jim	102

Figure 2.3: A database instance for schema in Table 2.1.

The functor syntax is rich enough to represent an entity-relation schema [107, Ch.2.2] via the following translation: Entity sets correspond to populations, descriptive attributes to function symbols, relationship tables to predicate symbols, and foreign key constraints to type constraints on the first-order variables.

We assume that a database instance assigns a unique constant value to each gnode $f(\mathbf{a})$. The value of descriptive relationship attributes is well defined only for tuples that are linked by the relationship. For example, the value of $grade(jack, 101)$ is not well defined in a university database if $Registered(jack, 101)$ is false. In this case, we follow the approach of Schulte *et al.* [95] and assign the descriptive attribute the special value $\perp$ for "undefined". Thus the atom $grade(jack, 101) = \perp$ implies $Registered(jack, 101) = F$. Fierens *et al.* [20] discuss other approaches to this issue. The results in this paper extend to functors built

with nested functors, aggregate functions [55], and quantifiers; for the sake of notational simplicity we do not consider more complex functors explicitly.

It is possible to represent relational data using a graph $G_D(V_D, E_D)$ where the nodes of G_D are the entities of the database and the edges between them represent the relationships between the entities. A vector is assigned to each of the nodes and edges to keep the information of the attributes of the entity and the links. Figure 2.4 shows $G_D(V_D, E_D)$ for the instance in Figure 2.3

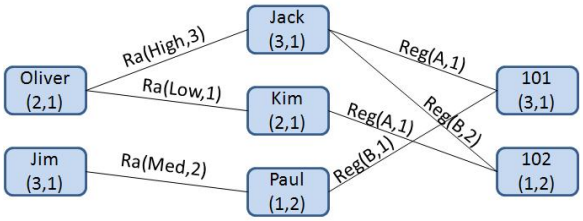


Figure 2.4: Graph $G_D(V_D, E_D)$ for the instance given in Figure 2.3. The nodes represent the entities of the schema and the links represent the relationships. An array is assigned to nodes and edges that carries the information of the objects and the relationships.

Graphical models [64, 45] have become a popular tool for modeling statistical relational models, and provide a principled approach to dealing with uncertainty and relational data through probability theory. The goal of graphical models is to represent a joint distribution. It encodes probabilistic relationships over a set of random variables. The graphical structure of the model is used to present the independence that exist among variables. The two most common classes of graphical models are Bayesian networks, which are directed graphs and Markov Networks, which are undirected graphs [83]. In this section we use $G_M(V_M, E_M)$ to represent the first-order class template of the methods discussed. Figure 2.5 shows an example of a $G_M(V_M, E_M)$ for the instance in Figure 2.3.

Learning graphical models for relational data is very slow due to model selection. To perform model selection in structure learning for a model $m \in \mathcal{M}$, a new graph $G_I(V_I, E_I)$ is produced, by rolling out G_D over m. $G_I(V_I, E_I)$ takes the template from m and the relations in $G_D(V_D, E_D)$ constrains the way it is rolled out. G_i is called the ground model and is generated with the property that there is a node for every attribute of each object and the parameters of the models for attributes of the objects are inherited from m. G_I evaluates how well m models G_D. Figure 2.6 shows an example. With this model selection technique, a graphical model for a given database can show not only the correlations between attributes

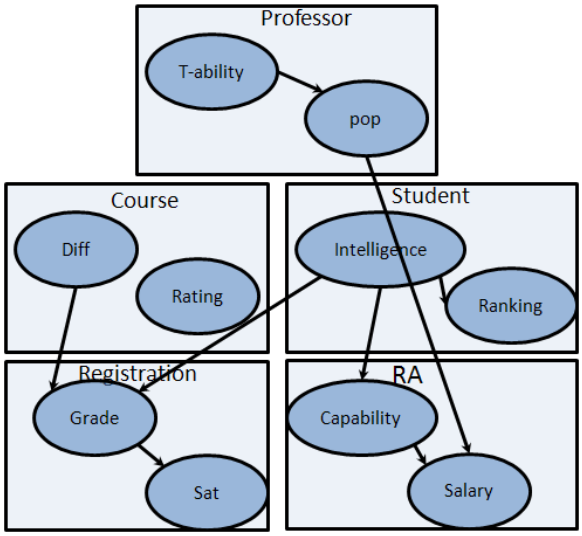


Figure 2.5: An example of a graph $G_M(V_M, E_M)$ for G_D given in Figure 2.4. The variables represent descriptive attributes of tables of the schema and edges show dependencies between attributes.

of each table, but also dependencies among attributes of different tables. For example, a statistical relational model may show that intelligence of a student and difficulty of a course are related to the grades achieved by the student.

Learning graphical models for relational data is much more challenging than learning graphical models for propositional data. The following are some of major differences of relational data amd propositional data.

- Propositional data consists of identically structured entities, typically assumed to be independently and identically distributed (iid); however relational data consists of different entities of different types which may be related to each other.

- In propositional data, learners model a fixed set of attributes intrinsic to each object, whereas in relational data, learners decide on what information to use in their model as the links of the linked objects may also carry helpful information. Collective inference [44] consider the links of the linked objects as well as just the links of the object.

- In modeling propositional data, the number of states is exponential in the number of attributes $O(2^n)$. For relational data, the G_I graph has $n \times m$ variables where n is the number of attributes and m is the number of objects. Modelling the joint distribution

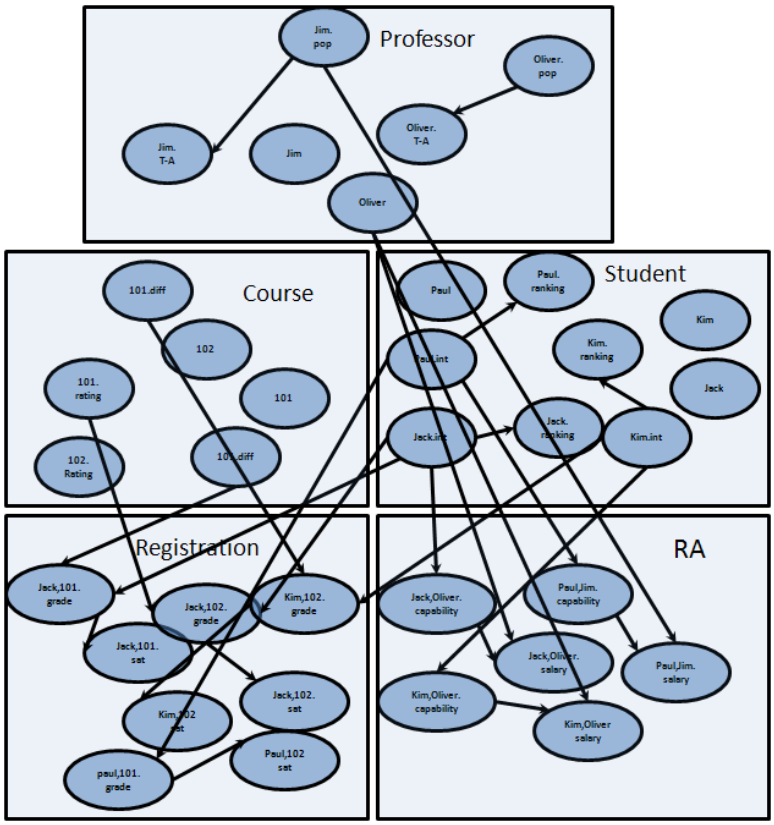


Figure 2.6: The graph G_I for G_M in Figure 2.5 rolled out over the schema in Figure 2.4. The variables of the graph show descriptive attributes of each object and edges connect attributes of objects using the template from G_M

of G_I is exponential in the product of the attributes and objects $O(2^{nm})$.

- Correlations between values of attributes of objects of the same type were dubbed autocorrelations by Jensen and Neville [12]. Let A be an attribute of an entity E and e_1 and e_2 are instances of type E. A relationship between objects of the same entity is required to have autocorrelation. Assume R(E,E) is a relationship on entity E where $R(e_1, e_2) = T$. There well may be a correlation between the attribute value $A(e_1)$ and the attribute value $A(e_2)$. A well known example for autocorrelation it that, the gene type of a child is correlated to the genes of his parents. The presence of autocorrelation is a feature of relational data which increases the complexity of relational learning.

- We call two tables T_1 and T_2 *related* if T_2 is a relationship table with foreign key

pointer to entity table T_1, or T_1 and T_2 are both relationship tables with foreign key pointers to a common entity table E_1. Let $X_1, Y_1, .., Y_k, , X_2$ for $k > 0$ be attributes associated with a database schema. The variables form a *slot chain* if neighboring variables are related; for example, $Y_1 =$ *grade* in the *Registered* table forms a slot chain for $X_1 =$ *intelligence* and $X_2 =$ *difficulty*. Correlations through a slot chain are also a feature of relational data and are challenging to discover.

- A difficulty with modeling relational data is that the database may contain information about many links related to an entity that need to be combined. For instance, a Bayes net may encode the probability that a student is highly intelligent given the properties of a single course they have taken. But the database may contain information about many courses that the student has taken, which needs to be combined; we may refer to this as the *combining problem*.

The presence of relational data not only leads to more accurate results on traditional tasks like classification and prediction, but also introduces some new tasks that have attracted the interest of many researchers. Popular tasks defined for SRL models include the following.

- Collective classification [44] is an extension of relational classification. Relational classification is the task of predicting the class of an object given its attributes, links, and other objects that are related through its links. In collective classification, the links of related objects are also considered for classification.
- Linked-based clustering [110] groups together objects that have similar characteristics both in their own attributes [23] and more importantly in the attributes of their links
- Link prediction [104] determines whether a relation exists between two objects from the properties of the objects and their links.
- Entity resolution is the problem of determining which records in datasets refer to the same objects in the real world [113]

In the remainder of the chapter we review four of the proposed models in statistical relational learning in details; the methods of representing, learning, and inference are discussed on a running example.

2.2 Probabilistic Relational Models(PRMs)

Probabilistic Relational Models (PRMs) [26] are a rich representation language for statistical models. PRMs were one of the first successful methods proposed for statistical relational learning. They use a Bayesian network as their framework and combine logical representation with probabilistic semantics. PRMs extend Bayesian networks with the concept of objects. A PRM and a database instance define a probability distribution over the descriptive attributes of the objects and the relationships. In this section we will review the representation, learning, and inference methods for PRMs.

2.2.1 Representation in PRMs

PRMs define a probability distribution for a *relational skeleton*, a partial specification of an instance of the schema where the relations are defined but the descriptive attributes of the entities and relationships are undefined. A PRM, like a Bayesian network, has two components: the dependency structure and the parameters associated with it. Random variables in PRMs can have two types of dependencies, (1) both variables correspond to attributes from the same table, and (2) variables correspond to attributes from different tables but there is a slot chain between them. The second type of dependency is usually between a value and a set which is an example of the combining problem. PRMs use the notion of aggregation from database theory to return a summary in one value. There are many functions used as aggregation of a set: *mode, mean, median, maximum, and minimum* are among the most used aggregation functions. Figure 2.5 shows a class level $G_M(V_M, E_M)$ for the instance in Figure 2.3.

A massive Bayesian network, G_I, serves as the ground model for PRMs. Figure 2.6 shows a ground Bayes net for the instance given in Figure 2.3. As with a normal Bayesian network, an acyclic network guarantees coherence of the probability model.

PRMs with structural uncertainty

The previous section described PRMs with uncertainty over just the descriptive attributes of entities and relationships. However, PRMs can be extended to allow uncertainty over the relationships between objects. They can define a probability distribution for an *object skeleton*, a relational skeleton where the relationship tables exist, but the reference slots are not assigned. For example the *Student*, *Course* and the *Registered* table are specified but

the student id and course id in the registration table are unassigned. Thus they need to specify a probability distribution over all objects in the entity tables for the reference slots. Since there are many objects, assigning a probability over each object is neither useful nor practical. To achieve a general and compact representation, they use the values of some descriptive attributes of entities to partition the distribution of objects. Acyclic PRMs with reference uncertainty relative to an object skeleton define a coherent probability distribution over instantiations extending the object skeleton.

The second form of structural uncertainty introduced by PRMs is existence uncertainty in which they further limit the background information. In reference uncertainty the number of tuples in both entity and relationship tables is known. In existence uncertainty the total number of links in relationships is unknown, so each potential link can be present or absent. The only input is an *entity skeleton*, which specifies the set of objects of the domain only for the entity classes. In existence uncertainty for each relationship, a binary attribute with values true and false is added to the list of descriptive attributes. The existence node, like any other node, can have parents and children, and there is a conditional probability table assigned to it. To make the model coherent and consistent, PRMs add the following conditions to the model.

1. Suppose that relation R has slots $p_1, ..., p_k$. The conditional probability table for the added node is filled regularly if all the slots are true and 0 otherwise.

2. If the binary attribute is a parent of a descriptive attribute of R, the conditional probability table of the descriptive attribute is filled regularly when the binary attribute is true and is a uniform distribution on all possible values in the domain of A if $E(R)$= False. For example, the conditional probability table for attribute *grade* given that it has three parents *intelligence*, *difficulty*, and $B(Registered)$ is given in Table 2.2. The probability for rows where B(*Registered*)= False is $\frac{1}{2}$ in this example. Rows where E(*Registered*)= True is calculated using look ups in *Registered*, *Student*, and *Course* tables in Figure 2.3.

2.2.2 Inference in PRMS

PRMs in few cases, when either the skeleton is small or the tree width is low, can use exact inference on the $G_I(V_I, E_I)$ graph. Unfortunately exact inference is usually not applicable

intelligence	*difficulty*	E(*Registered*)		*grade* = A	*grade* = *B*
1	1	True		0	1
1	1	False		$\frac{1}{2}$	$\frac{1}{2}$
1	2	true		$\frac{1}{2}$	$\frac{1}{2}$
1	2	False		$\frac{1}{2}$	$\frac{1}{2}$
2	1	True		$\frac{1}{2}$	$\frac{1}{2}$
2	1	False		$\frac{1}{2}$	$\frac{1}{2}$
2	2	true		1	0
2	2	False		$\frac{1}{2}$	$\frac{1}{2}$
3	1	True		1	0
3	1	False		$\frac{1}{2}$	$\frac{1}{2}$
3	2	true		0	1
3	2	False		$\frac{1}{2}$	$\frac{1}{2}$

Table 2.2: The conditional probability table for attribute grade when a binary node B(registration) is in the set of parents of grade.

with real world data. Inference in PRMs requires inference over the ground network defined by an instantiated PRM for a specific skeleton. Because $G_I(V_I, E_I)$ is usually very large, efficient inference is very complicated. The approximate algorithm used for inference in PRMs is a variant of belief propagation. We briefly describe the algorithm but for more information see [73, 111]. A Bayesian network can be converted into a family graph where a node is added for the set of every node and its parents, and an edge is added between nodes if they have common variables. If some new evidence is observed in the Bayesian network it may effect all the other nodes that are dependent on the observed node. The family graph is structured in a way that all of the dependent nodes in the Bayesian network are d-connected. So, a parameter passing algorithm can update all effected nodes by passing the new information through its edges. While the algorithm is not guaranteed to converge, it typically does so after several iterations.

2.2.3 Learning In PRMs

We will review parameter learning and structure learning in PRMs briefly.

Parameter Learning

The key feature in parameter learning is the likelihood function which is defined as the probability of the data given the model. Let $G_D^{\mathbf{X}}(a)$ be an assignment of values of object a to a set of variables $\mathbf{X}$. For example, $G_D^{intelligence,ranking}(Jack) = \{3, 1\}$. Formula 2.1 shows

the probability of the data given the model for PRMs.

$$l(\theta_{G_M}|G_D, G_M) = \sum_{X \in V_M} \sum_{a \in \mathbf{a}} \log p(G_D^{\mathbf{X}}(a)|G_D^{\mathbf{pa(X)}}(a)) \tag{2.1}$$

Where θ_{G_M} is the parameters for G_M, $\mathbf{a}$ is the set of objects. An important component of learning is the score function. A score is decomposable if it can be calculated by independent summations over the score of the nodes of the structure [37]. Decomposability of the score has significant impact on the efficiency of the algorithm. To perform parameter learning using a decomposable score, PRMs use the well established maximum likelihood parameter estimation. Using Maximum Likelihood, Formula 2.1 can be decomposed into summation terms that each may be maximized separately,

$$l(\theta_{G_M}|G_D, G_M) = \sum_{X \in V_M} \sum_{x \in X} \sum_{u \in \mathbf{pa(X)}} C_{[x,u]} \times \log \theta_{G_M(x|u)} \tag{2.2}$$

where $C_{[x,u]}$ is the number of times $G_D^{\mathbf{X}}(a) = x$ and $G_D^{\mathbf{pa(X)}}(a) = u$. $v|u$ is the conditional probability of $G_D^X(a) = v$ given $G_D^{pa(V)}(a) = u$.

An important advantage of using a relational database is that SQL queries may be used to calculate local probabilistic distributions and sufficient statistics. For example the statistics for intelligence of students, their grades and capabilities from the *Registered* and the *RA* table can be computed respectively, **SELECT** capability, intelligence, grade, count(*)
FROM *Registered*, *Student*, *Course*, *Professor*, *RA*
WHERE $Registered.student\_id = Student.student\_id$ and $Registered.course\_id = Course.course\_id$ and $RA.student\_id = Student.student\_id$ and $RA.professor\_id = Professor.professor\_id$
GROUP BY $capability, intelligence, grade$

In cases where an attribute in the set of the parents is from a different tables, a view is used to reduce the computation cost. To compute the correct statistics, the projection and the group by commands in SQL may be used on the created view. SQL also supports most of the aggregation functions used by PRMs.

Structure Learning

Structure learning is more challenging than parameter learning in graphical models. The general task of structure learning is to find the set of edges E_M in G_M. The "goodness" of different structures must be comparable to allow preference of a model over another.

For evaluating different structures, score functions like BIC [37] as described in Section 2.1.4 are used. As BIC and most other score functions are only defined for (iid) data, a new definition for these scores must be specified for use of relational data. Generally for Bayesian networks, the task of finding the best structure is NP hard. PRMs use greedy algorithms that iteratively modify the structure to increase the score. The operations used in each step are adding, removing or reversing an edge. Adding or reversing an edge may introduce cycles but it is possible to check a PRM for cycles in $O(|V_M| + |E_M|)$. In each step, all possible transformations using these operations are considered and the transformation with the best score is selected. Greedy algorithms often get stuck in local maximums. PRMs consider a number of random operations and then restart the greedy search process to avoid this problem. Greedy methods become more efficient if a tabulist is used to keep tack of visited states, so the search algorithm does not return to a recently visited state. Also, operations used in PRMs structure learning only change the parents set of maximum of two nodes on the sides of the modified edge. By using a decomposable score, only the components of the score associated with these two nodes need to be reevaluated.

Learning With Reference Uncertainty To cover learning with reference uncertainty PRMs need to define the partitions which involves expanding search operators to allow addition and deletion of attributes in the partition function. PRMs with reference uncertainty need to introduce two new operators **refine** and **abstract**. The partitions can be defined using a decision tree where *refine* adds a split to one of the leaves to make the result more specific and *abstract* removes a split to make the partitions more general.

Learning With Existence Uncertainty Extending PRMs with existence uncertainty is straightforward. The existence attribute is treated the same as descriptive attributes. The new concept is how to compute sufficient statistics that include existence attributes without explicitly enumerating the non existent entities. To do so PRMs count the number of potential objects with a given instantiation from entity tables and subtract the actual number of true objects that have been instantiated from relationship tables. No new operations are added to handle existence uncertainty but PRMs force additional edges in the class dependency graph.

2.3 Markov Logic Networks(MLN)

Markov Logic Networks (MLNs) [15] are among the most well known methods proposed for statistical relational learning. They have been introduced as a unifying framework for SRL to facilitate transfer of knowledge across approaches, make comparison and understanding of different models easier, and help establish structure to the field. The authors mention several criteria for the unifying framework. First, the framework must be consistent with first-order logic and probabilistic graphical models as many current approaches rely on them. Second, the framework must be clear and simple. Finally, the framework must facilitate the use of domain knowledge in statistical relational models.

Essentially, an MLN is a set of weighted first-order formulas that compactly defines a Markov network comprising ground instances of logical predicates. The formulas are the structure or qualitative component of the Markov network; they represent associations between ground facts. The weights are the parameters or quantative component; they assign a likelihood to a given relational database by using the log-linear formalism of Markov networks. An open-source benchmark system for MLNs is the Alchemy package [61].

2.3.1 Representation in MLNs

MLNs combine first order logic and Markov networks. A Markov network is a model for the joint distribution of a set of variables $X = (X_1, X_2, \dots X_n)$. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{2.3}$$

where $(x_{\{k\}})$ is the state of the kth clique. Z, known as the partition function, is given by $Z = \sum_{x \in X} \prod_k \phi_k(x_{\{k\}})$

A first order knowledge base is a set of hard constraints where if a grounding violates just a single formula, the probability of its happening is zero. This hard constraint in first order logic makes them inappropriate for modeling noisy data or datasets with contradicting rules. Markov logics extend first-order logic so that the hard constraints are replaced with soft

First-order logic	Weight	English
$\forall x(intelligent(x) \Rightarrow highrank(x))$	2.3	If a student is intelligent then his rank is high
$\forall x \forall y(intelligent(x) \wedge friend(x,y) \Rightarrow intelligent(y))$	0.7	If a student has an intelligent friend then he is intelligent

Table 2.3: Example of a small MLN

constraints using formulas that have weights based on the importance of the constraint. In MLNs, if a grounding violates some formulas, it is just less probable. The fewer groundings a formula violates, the more probable the formula is.

Formally, a Markov Logic Network is a set of pairs of formulas and their corresponding weights (F_i, w_i) where formulas are in first order logic and the weights are real numbers. The set of formulas in MLNs correspond to the class model G_M. An MLN with a finite set of objects in G_D defines a ground Markov network G_I which has a binary node for each ground predicate in the MLN. The value of the node is 1 if the ground atom is true and 0 otherwise. An MLN is a template for the ground Markov network and the size of the model is a function of the number of objects. The structure and the parameters of the ground network are forced by the MLN model. The edges are cliques between ground atoms that appear together in a formula and all groundings of a the same formula have the same weight as specified by the MLN.

Table 2.3 presents an MLN for the university example. The MLN has two formulas for the students. One of the formulas connects *intelligent* and *highrank* and the other one connects $friend$ and *intelligent*. Assuming there are only two students Jack (j) and Kim (k) in our university, Figure 2.7 shows the ground Markov network G_I for the MLN in Table 2.3. Edges between $friend$ predicates and *intelligent* predicates are due to the second formula and edges between *intelligent* predicates and *highrank* predicates are due to the first formula.

Each state of the Markov network represents a database instance. The probability distribution over database instances x specified by the ground network is calculated by

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp(\sum_i w_i n_i(x)) \tag{2.4}$$

where $n_i(\mathbf{x})$ is the number of true groundings for F_i in $\mathbf{x}$ and Z is the partition function that is used to make sure the summation of all possible groundings add up to one. Leaving

out the denominator, the likelihood is calculated using Formula 2.5

$$P(\mathbf{X} = \mathbf{x}) \propto \exp(\sum_i w_i n_i(\mathbf{x})) \tag{2.5}$$

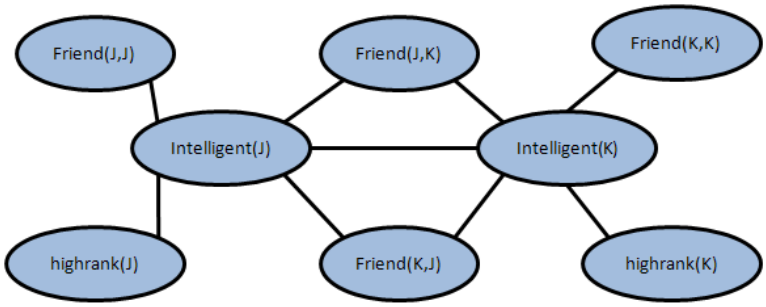


Figure 2.7: Ground Markov network obtained by applying MLN of Table 2.3 to Jack and Kim

To show an example of calculating probabilities for possible database instances, we further simplify the example. Suppose that the MLN has just the first formula in Table 2.3 and Jack is the only student. This leads to four database instances:

1. $\{intelligent(J), highrank(J)\}$
2. $\{intelligent(J), \neg highrank(J)\}$
3. $\{\neg intelligent(J), highrank(J)\}$
4. $\{\neg intelligent(J), \neg highrank(J)\}$

Using Formula 2.5, the likelihood for $P(X = \{\neg intelligent(J), highrank(J)\}) = 1$ and the likelihood for the other three instances is $e^{2.3}$. As the probability of all database instances needs to add up to 1, $\frac{1}{Z}$ can be calculated. Formula 2.6 computes the inverse of the potential function of our example.

$$\frac{1}{Z} = \frac{1}{3e^{2.3} + 1} \tag{2.6}$$

so $P(X = \neg intelligent(J), highrank(J)) = \frac{1}{3e^{2.3}+1}$ and probability of the other three instances is $\frac{e^{2.3}}{3e^{2.3}+1}$. The probabilities indicate that instances that are inconsistent with the formulas in the MLN are less probable than the consistent database instances.

2.3.2 Inference in MLN

In this section we discuss how inference is performed in MLNs. The sort of queries that may be answered using an MLN are of the form "What is the probability that F_{i_1} holds knowing that F_{i_2} holds, given the ground Markov network". This may be formalized as $P(F_{i_1}|F_{i_2}, G_I(V_I, D_I))$. Inference in MLNs is done on the ground Markov network; a brute force way of calculating this probability is by using formula 2.7.

$$P(F_{i_1}|F_{i_2}, M) = \frac{P(F_{i_1}, F_{i_2}|M)}{P(F_{i2}|M)} = \frac{\sum_{\mathbf{x}\in(\mathbf{x}_{F_{i\,1}}\cap\mathbf{x}_{F_{i\,2}})} P(\mathbf{X}=\mathbf{x}|M)}{\sum_{\mathbf{x}\in\mathbf{x}_{F_{i\,2}}} P(\mathbf{X}=\mathbf{x}|M)} \tag{2.7}$$

$\mathbf{x}_{Fi}$ is the set of instances where F_i holds. This direct inference method is highly intractable and is only applicable to very small domains. Inference in Markov networks itself is generally NP complete and cannot be carried out on networks with many nodes. So, a different method of inference is used in MLNs. The approach has two main phases.

In the first phase a minimal subset of G_I, the ground Markov network, that is required for computing $P(F_{i_1}|F_{i_2}, M)$ is obtained. Many predicates that are irrelevant to F_{i_1} and F_{i_2} may be filtered in this phase. As a result, a smaller Markov network is used for inference.

In the second phase, we perform inference on the Markov network using Gibbs sampling [8] where the nodes of F_{i_2} are observed and are set to their values. In Gibbs sampling the unobserved variables in the network are randomly initialized and ordered to $\{X_1 = x_1, \ldots, X_n = x_n\}$. In step 1 a new value x_1' for variable X_1 is sampled conditional on all the other variables $P(X_1|X_2 = x_2, \ldots, X_n = x_n)$, and in step i a new value x_i' for variable x_i is sampled conditional on all the other variables $P(X_i|X_1 = x_1' \ldots X_{i-1} = x_{i-1}', X_{i+1} = x_{i+1} \ldots X_n = x_n)$. We can take advantage of the conditional independence to ease the computation. Conditional on its immediate neighbors, Markov Blanket, a node is independent of all of the other variables. Assuming $MB(X_k) = \mathbf{b} = \{b_1 \ldots b_i\}$ shows the different states of the Markov blanket of the node X_k, the probability of a ground atom $X = x$ given its Markov blanket is in state b_l is

$$\frac{\exp(\sum_{f_i\in F_l} w_i F_i(\mathbf{x}, b_l))}{\exp(\sum_{f_i\in F_l} w_i F_i(X=0, b_l)) + \exp(\sum_{f_i\in F_l} w_i f_i(X=1, b_l))} \tag{2.8}$$

where F_l is the set of ground formulas that X appears in, and $f_i(X = 1, MB(X) = b_l)$ is the value corresponding to F_i when it has been grounded by $X = 1, MB(X) = b_l$. To make the process less effected by the initialization, the Markov chain is ran multiple times.

While transferring an MLN to a Markov network, some undesired cliques may be generated that do not correspond to groundings of F_is; the Markov network contains some spurious cliques that should not be involved in computing probability distributions of database instances. Therefore, the MLN is required for guidance and the Markov network can not be used on its own for inference.

2.3.3 Learning in MLN

Having efficient structure and parameter learning algorithms is crucial for any graphical model. Without efficient learning algorithms, the model is unsuitable for realistic size domains and problems. Learning is a hard task in Markov networks; in this section, parameter learning for MLNs is discussed and an outline of structure learning for MLNs is explained.

Parameter learning in MLNs

Finding the weights of formulas in an MLN is equivalent to computing parameters in other models. The weights are learned from the relational database. Assuming the network has n ground atoms, a database has up to n facts. MLNs use the closed-world assumption [18]; if a ground atom is absent in the database, it is assumed to be false. A database may be represented as a vector $\mathbf{x} = \{x_1 \dots x_n\}$. In MLNs, the weight of a formula is the derivative of the log-likelihood of Formula 2.4 with respect to its weight.

$$\frac{\partial}{\partial w_i} \log p_w(\mathbf{x}) = n_i(\mathbf{x}) - \sum_{x'} P_w(\mathbf{x'}) n_i(\mathbf{x'}) \tag{2.9}$$

where the sum is over all database instances x', and $P_w(\mathbf{X} = \mathbf{x'})$ is $P(\mathbf{X} = \mathbf{x'})$ computed using the current weight vector. The derivative of the log-likelihood has an intuitive form which is the difference between the number of true groundings of the formula from the database x and the expected number of true groundings for the formula over all database instances x'. The summation over all database instances is the derivative of the partition function which requires inferencing over the model.

Formula 2.9 is difficult to compute because of two main reasons. First, counting the true groundings of a first order clause in a database is #p-complete [108] which causes problems in large domains. The second problem is with computing the expected number of true groundings over all database instances which requires inference over the model. Having the partition function as part of the formula makes the computation intractable.

One method to overcome the first problem is to uniformly sample the groundings and approximate $n_i(x)$. For the second problem, an approximation for calculating probability of a world using pseudo-likelihood [3] is used that omits the use of the partition function. Pseudo-likelihood is a measure in statistics that serves as an approximation of the distribution of **x** based on its Markov blanket instead of all other **x'**. Formula 2.10 shows an approximation for $p(x)$ using pseudo-likelihood,

$$P_w^*(x) = \prod_{l=1}^{n} P_w(x_l|\mathbf{b}) \tag{2.10}$$

where **b** is the state of the Markov blanket of X_l in x and the whole formula is computed using Formula 2.8. Formula 2.11 takes the derivative of the log-likelihood of Formula 2.10 to find the gradients,

$$\frac{\partial}{\partial w_i} \log P_w^*(\mathbf{x}) = \sum_{l=1}^{n} [n_i(\mathbf{x}) - P_w(X_l = 0|\mathbf{b}) n_i(\mathbf{x}_{[X_l=0]}) - P_w(X_l = 1|\mathbf{b}) n_i(\mathbf{x}_{[X_l=1]})] \tag{2.11}$$

where $n_i(\mathbf{x}_{[X_l=0]})$ is the number of true groundings of the ith formula where $X_l = 0$. Consequently, using Formula 2.11 boosts the efficiency without requirement of inference over the model. To avoid over-fitting, a Gaussian prior is used on the weights to penalize the pseudo-likelihood.

Though pseudo-likelihood is usually a good approximation, it fails when querying distant variables in the model [15]. Pseudo-likelihood uses local measures, Markov blanket, to calculate probabilities, so it is unable to achieve reasonable results for queries that ask about the network globally. Discriminative methods are used in such situations which model the conditional probability distribution $P(y|x)$. One of the limitations of discriminative models is that they cannot sample the joint distribution. While using discriminative learning, you need to know the evidence and the query nodes in advance. Once you know the observed variables, the domain of database instances is decreased in orders of magnitude making learning a much simpler task.

Structure learning in MLNs

Structure learning in MLNs is the problem of discovering what formulas hold from the data; it is similar to *Inductive Logic Programming* approaches. The field of inductive logic programming is concerned with finding Prolog programs or Horn clauses from data. MLNs

extend the approach to finding arbitrary first order formulas instead of just Horn clauses from the data. First MLN structure learning used the CLAUDIEN [13] system which is able to learn first order formulas and not just Horn clauses. In this approach, a set of candidates for formulas are extracted by adding, removing, and negating literals. The formulas are then evaluated based on the data. MLN methods evaluate their structure differently compared to inductive logic programming approaches. In inductive logic programming methods, the search is usually guided by a criterion like accuracy or information gain, while in MLNs the structure is evaluated using maximum likelihood.

To evaluate the structure using maximum likelihood, the weights for each set of candidates are required to be calculated, which is very costly. In order to make evaluation of structures more feasible, several techniques are used. First, the weights are learned less accurately and faster for candidates in the structure learning process; once the final structure is obtained, the parameters are relearned more accurately. Secondly, when a formula is modified, it is possible to initialize the weight of the formula with its previous weight instead of 0 as it is probable that the change has has no effect on the weight of the formulas. Finally, the use of sub-sampling to approximate the number of true groundings of a formula in a model reduces the cost of structure learning. It is possible to sub-sample thousands of groundings for a domain with millions of groundings and extrapolate the results to the total.

2.4 Relational Dependency Networks(RDNs)

Relational Dependency Networks (RDNs)[78], an extension of Dependency Networks (DNs)[41], are a class of graphical models that approximate a joint distribution using a bi-directed graph with conditional probability tables for variables. They inherit their undirected graph from Markov networks and their conditional probability tables from Bayesian networks. RDNs have several characteristics that makes them favorable for relational data: (1) unique representation which provides the ability to represent cyclic dependencies, (2) simple methods for parameter estimation, and (3) efficient structure learning techniques. The strength of RDNs is mostly due to the use of pseudo-likelihood [3] for estimating an acceptable approximation of the joint distribution.

The presence of autocorrelation in relational data grants a strong motivation for using RDNs; autocorrelation will be discussed in Chapter 4. Although domain knowledge may

sometimes be used in directed models to structure autocorrelation in an acyclic manner [26], often such knowledge does not exist. This limitation makes directed models unsuitable for presenting autocorrelations. For undirected models, in principle there is no restriction against having cycles, but inefficient parameter learning due to the existence of the partition function makes undirected models impractical for presenting autocorrelation. RDNs can both represent and reason with cyclic dependencies that are required to express autocorrelation. Because RDNs approximate the joint distribution, they can avoid dealing with the partition function and thus have relatively easy parameter and structure learning algorithms.

RDNs extend the graphical model of DNs to the relational setting. The model encodes probabilistic relationships among a set of random variables with a bi-directed graph $G_M(V_M, E_M)$ where conditional independence is interpreted using graph separation as in undirected models. Although conditional independence is inferred using an undirected view of the graph, bi-directed edges are used to define the set of neighbors of a node used in their CPT. Each node has a probability distribution conditional on its neighbors as in directed models. Figure 2.8 shows an instance of a database on papers and authors. The example contains autocorrelation to emphasize the ability of RDNs for handling recursive dependencies.

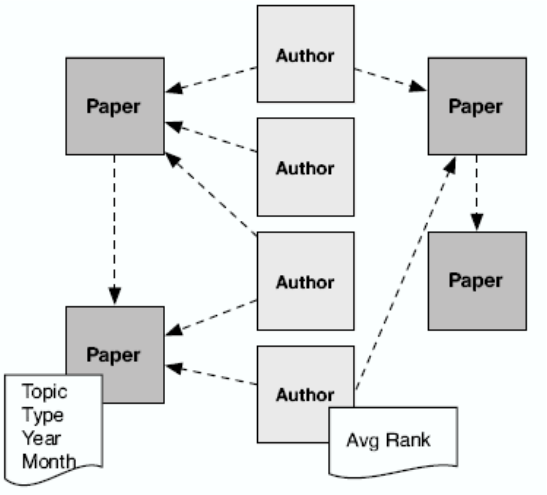


Figure 2.8: A graph G_D representing the schema and the instance of a dataset for RDNs taken from [78].

Figure 2.9 shows an example of an RDN for the instance given in Figure 2.8. For this model, the CPT for year contains topic, but the CPT of topic does not contain year.

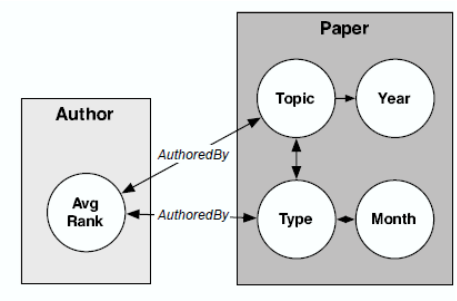


Figure 2.9: An example of a G_M for the G_D in Figure 2.8 taken from [78]

The nodes of the model are similar to the variables in PRMs discussed in Section 2.2.1. Each node $X \in \mathbf{X}$ corresponds to a descriptive attribute from an entity or a relationship table. An edge between two nodes correspond to correlations between attributes of the dataset. A limitation of the RDN model is that, the conditional probability distributions do not factor the joint probability, so its impossible to calculate the joint probability directly as in directed models.

2.4.1 Learning in RDNs

RDNs extend the learning of DNs to a relational setting. The set of the conditional probability tables CPTs describe both the structure and the parameters of the model. RDNs use pseudo-likelihood techniques [3] to avoid the complexities of estimating the partition function. Instead of optimizing the log-likelihood of the joint distribution, RDNs optimize the pseudo-likelihood for each node separately conditioned on all its neighbors. Formula 2.12 computes the pseudo-likelihood for each node in G_M seperately,

$$Pl(\theta_{G_M}|G_D, G_M) = \sum_{X \in V_M} \sum_{x \in X} \sum_{u \in (\mathbf{pa(X)})} P(x|u) \tag{2.12}$$

where θ_{G_M} is the parameters for G_M. RDNs use conditional relational learners in their learning procedure.

Conditional Relational Learners:

RDNs use a conditional relational learner at the heart of both parameter and structure learning. The variables selected by conditional relational learners are reflected as edges in G_M. We outline two well known conditional relational learners used by RDNs; *relational Bayesian classifiers* [80] and *relational probability trees* [77]

A Relational Bayesian classifier is a non selective model that treats heterogeneous relational subgraphs as a homogeneous set of attribute multisets. The classifier assumes that each value in the multiset is drawn independently from the same multinomial distribution. For example, when considering the year of the publication of references of a paper, the publications' years form a multiset { 1995, 1995, 1995, 1996 } where the classifier selects values independently from the multiset distribution. A Relational Bayesian classifier, as a Naive Bayesian classifier, further assumes independence between the nodes given the class label. As the model is non-selective, all descriptive attributes are included in the set of the parents of a node.

A Relational probability tree is a selective model that extends traditional classification trees to a relational settings. As in a relational Bayesian classifier, they treat heterogeneous relational subgraphs as a set of attribute multisets; however, instead of treating the values like an independent set, relational probability trees use aggregation functions to map the set of values into a single value. The model considers four classes of aggregation functions: Mode, Count, Proportion, and Degree. Features build using aggregation functions are evaluated using χ^2 to measure the correlation between the function and the class.

2.4.2 Inference in RDNs

Figure 2.10 provides an example of the G_M graph of RDN in Figure 2.9 rolled out over a test data of four authors and four papers where P_1 is authored by $\{A_1, A_2, A_3\}$, P_2 is authored by $\{A_2, A_4\}$, P_3 is authored by $\{A_3, A_4\}$, and P_4 is authored by $\{A_4\}$.

RDNs use Gibbs sampling as explained in Section 2.3.2 for inference on G_I. The values of the unobserved variables are initialized with their prior distribution and are iteratively relabeled using the current state on the model and the CPT of the node. Gibbs sampling is generally inefficient approach to estimate the joint probability of the model, however it is reasonably fast to estimate conditional probabilities for each node given its parents.

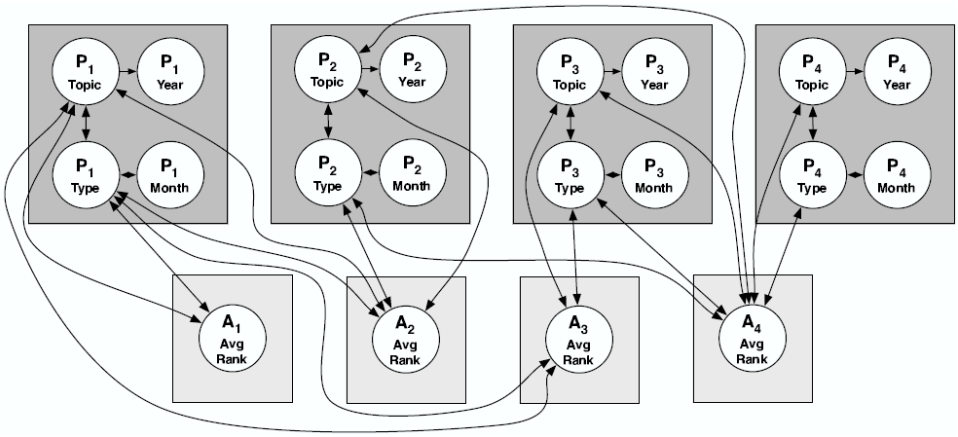


Figure 2.10: An example of an inference graph G_I for the G_M in Figure 2.9 rolled out over a graph G'_D with four authors and four papers.

2.5 Baysian Logic Programs (BLPs)

Bayesian Logic programs [46] are a model based on Bayesian networks. BLPs unify Bayesian networks with logic programming [72] to overcome the propositional character of Bayesian networks and the purely logical nature of logical programs. BLPs use Bayesian clauses with conditional probability tables to represent the distribution of the head of the clause conditional on its body, and use combining rules to combine the information on a single literal that is the head of several clauses. BLPs are implemented in a package called BALIOS [46] and are considered as one of the successful models of Statistical relational learning.

2.5.1 Representation and Learning in BLPs

BLPs are produced from logical programs. A logical program is a set of clauses of the form $A : B_1, B_2, \ldots B_n$ where A and B_i are universally quantified atoms. We call A the head and B_is the body of the clause. The head of the clause is considered true in the model if the body of the of the clause is true. BLPs use Bayesian clauses which differ from logical clauses. Bayesian clauses use a conditional probability table to keep the probability of the head of the clause conditioned on its body, whereas logical clauses have a deterministic value. It is possible to have several clauses with the same variable in the head of the clause. Since each clause has its own local probability distribution, a variable may have several local probability distributions with possibly different sets of parents. Here are two clauses

for defining an intelligent person.

$$intelligent(X) : -highrank(X)$$

$$intelligent(X) : -friend(X, Y), intelligent(Y)$$

To obtain a single conditional probability distribution for the variable that includes the union of all parents, both *combining rules* and aggregation functions are used. A combining rule is a function that maps finite sets of conditional probability distributions $\{ P(A|A_{i1} \dots A_{in_i}), i = 1 \dots m \}$ on to one combined conditional probability distribution $P(A|B_1 \dots B_k)$ with $P(A|B_1 \dots B_k) \subseteq \bigcup_{i=1}^{m} \{A_{i1} \dots A_{in_i}\}$. BLPs can also use aggregation functions, as in PRMs. In some domains it makes more sense to use aggregation functions instead of combination rules. For example, to summarize the grades of a student it is more appropriate to use aggregation functions instead of combining rules.

BLPs use a unique and complicated graphical model which is an extension to Bayesian networks. The model uses a bipartite directed acyclic graph with two set of nodes: Gray ovals denote random variables and black boxes denote local probability models. Incoming edges to black boxes represent parents of the variable x_i that is connected through a single outgoing edge of the black box. The black box specifies a conditional probability distribution $p(x_i|pa(x_i))$. Variables are given as white ovals and constants are represented as white boxes. Arguments of variables are represented as white circles on the boundary of the ovals.

Consider Figure 2.11, which is the BLP for part of our university example. BLPs add aggregation functions to their model using octagonal nodes. R5 computes the average grade for a student on the set of subjects that he is registered in deterministically, and R6 shows that rank of a student probabilistically depends on his average value.

2.5.2 Learning in BLPs

Learning in BLPs is a probabilistic extension of learning in inductive logic programing [72] as in MLNs [15] and is formulated as follows: Given a set of Bayesian logic programs G_M, G_D, and a scoring function $f(.)$; find a acyclic candidate G_M^* such that G_M^* matches G_D best according to $f(.)$. The score function $f(.)$ is used to evaluate how good the clauses are.

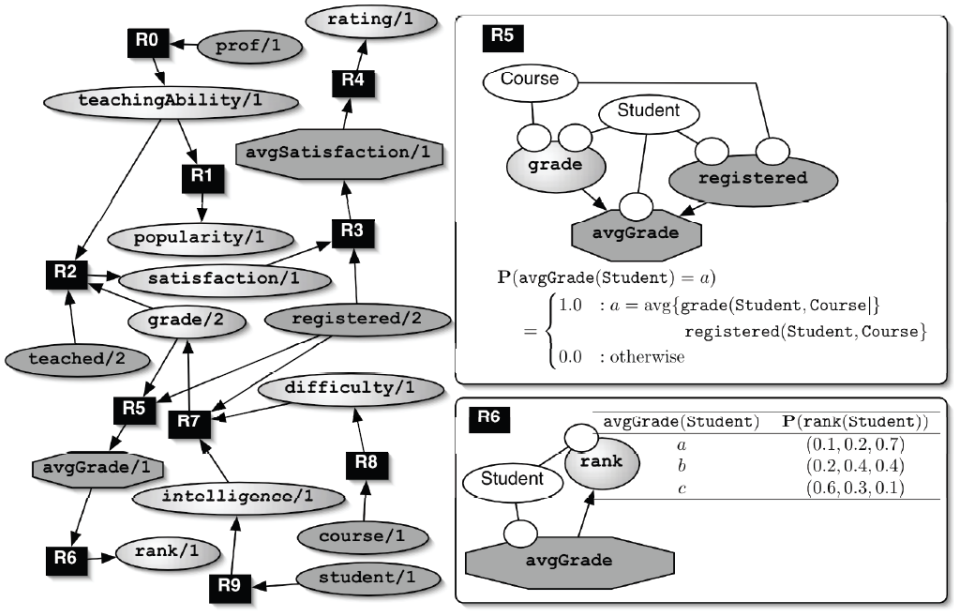


Figure 2.11: A Bayesian logic program for part of the university domain. Octagonal nodes show aggregate predicates. The Figure is taken from [46]

To adapt traditional techniques used for parameter estimation of Bayesian networks such as *expectation maximization* algorithm [14], combining rules are required to be decomposable [40]; most common combining rules for Bayesian networks such as "noisy or" [102] are decomposable. The best match refers to those parameters of the associated conditional probability distributions that maximize the scoring function where the score function is based on maximum likelihood [17].

Structure learning in BLPs follows the procedure of rule learning in ILP systems [82] which have operators such as adding and deleting literals, flipping, instantiating variables, unifying variables on literals or clauses. BLPs speed up the learning procedure executing several operations simultaneously. A simple hill climbing algorithm for BLPs learning can be sketched as follows: (1) start with a Bayesian logic programs G_M, (2) compute $F(G_M)$ based on G_D, (3) carry out all operators to find all the neighbors G'_M of G_M that do not introduce cycles, (4)compute $f(G'_M)$ based on G_D, (5) if $f(G_M) < f(G'_M)$ then $G_M = G'_M$, (5) repeat until G'_M such that $f(G_M) < f(G'_M)$ does not exist.

2.5.3 Inference in BLPs

Inference, as in other SRL methods, is intractable in BLPs and is proceeded via grounding the clauses of Bayesian logic program. Each Bayesian logic program specifies a propositional Bayes net, where inference is done using standard Bayes net learning algorithms. The set of the variables in G_I of BLPs is similar to those of MLNs: A variable is required for each grounding of each predicate. The set of predicates grounding a clause are connected to each other; the literals of the body of the clause are the parents of the literal in the head of the clause. Figure 2.12 is the G_I for the first order formulas in Table 2.3 with constants k and j. The second formula is illegal for BLPs because it introduces cycles. The formula is changed into $\forall x \forall y(intelligent(x) \wedge friend(x, y) \Rightarrow highrank(y))$

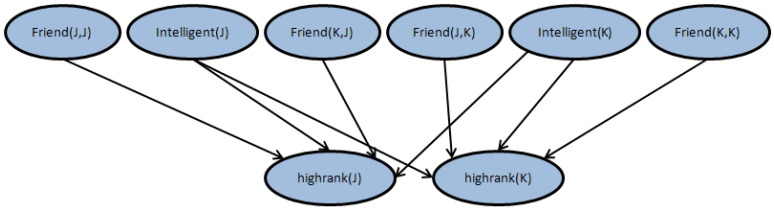


Figure 2.12: An example of inference graph for two formulas $(intelligent(x) \Rightarrow highrank(x))$ and $(intelligent(x) \wedge friend(x, y) \Rightarrow highrank(y))$.The set of predicates grounding a clause are connected to each other; the literals of the body of the clause are the parents of the literal in the head of the clause.

Combining rules are required for predicates that are head of several clauses. The parameters of the two clauses need to be combined to achieve correct conditional distribution for *highrank*. A well known combining rule that is frequently used is "noisy or".

2.6 Comparison with previous work

In this section we compare Probabilistic Relational Models (PRMs), Markov Logic Networks (MLNs), Relational Dependency Netoworks (RDNs), and Bayes Logic Networks (BLPs) along various dimensions of importance.

Class Level Model. The models considered for the literature review have a variety of different class level models. PRMs are a first order extension of Bayes nets that can have self loops. MLNs extend logical knowledge bases with weights to introduce weighted first order clauses as a class model. RDNs combine the graphical structure of undirected models

with directed models to introduce a bi-directed model. RDNs, as in directed models, use conditional tables for storing the parameters and as in undirected models, obtain the Markov blanket of a node to be its immediate neighbors. BLPs introduce their own graphical model; they extend Bayesian networks with a bipartite graph with various nodes. Table 2.6 presents the structure of class level models of various models.

SRL Method	Class Level Model
PRM	First order extension of Bayesian networks
MLN	Weighted logical clauses
RDN	Bi-direceted model
BLP	bipartite directed model

Table 2.4: A comparison of class level models for various SRL methods.

Learning. Maximum Likelihood is the dominant parameter learning method when the structure is known and the data is fully observable. Structure learning is more difficult and a variety of methods are used for it. PRMs and BLPs use score based learning as discussed in 2.1.4; score based methods lead to local maxima solutions. MLNs use inductive logic programming based methods, discovering what formulas hold from the data, for learning. Such methods are usually not scalable and very inefficient for large datasets. RDNs use two relational learners as part of their model, Relational Bayesian Classifiers and Relational Probability Trees. Using either relational learner has its own set of problems. Relational Bayesian Classifiers are non selective and choose independently a single value, as a representer, from the multinomial distribution of values of the link which weakens the model. Relational Probability Trees use aggregation functions. Choosing an informative aggregation function exponentially increases the complexity of the learning procedure. Table 2.6 illustrates the learning methods for various SRL methods.

SRL Method	Parameter Learning	Structure Learning
PRM	Maximum likelihood to fill cpts	Score based
MLN	Maximum likelihood to learn weights	Inductive logic programming methods
RDN	Maximum likelihood	Relational naive Bayes and Relational decision tree
BLP	Maximum likelihood to fill cpts	Score based

Table 2.5: A comparison of learning methods for various SRL methods.

Inference. Performing inference in a reasonable time frame is probably the biggest limitation for SRL methods. The size of the graph G_I is proportional to the number of descriptive attributes and objects, which limits the scalability for many realistic datasets. PRMs and

BLPs have a slow inference procedure as they use standard complex Bayesian network inference algorithms on their ground G_I network. MLNs share the same problem as they use an undirected model as their ground network. Exact inference on large datasets for MLNs is also impractical because of the computations on the partition function. MLNs are forced into using approximation techniques for inference. Pseudo-likelihood fails to give significant results when querying on variables that are distant in the model [15]. Inference is quicker in RDNS, because they approximate a joint distribution using a bi-directed graph with conditional probability tables for variables; however, the conditional probability distributions do not factor the joint probability so it is impossible to calculate the joint probability directly, a common task in most other statistical relational models. Extensive research is currently being performed on lifted inference [47], which aims to do exact inference without materializing the ground inference graph. Table 2.6 summarizes the learning methods for various SRL methods.

SRL Method	**Inference Graph**	**Inference Method**
PRM	Ground Bayesian network	Belief propagation
MLN	Ground Markov network to learn weights	Pseudo-likelihood, sampling
RDN	Ground relational dependency network	Pseudo-likelihood
BLP	Ground Bayesian network	Any standard BN inference

Table 2.6: A comparison of inference methods for various SRL methods.

Autocorrelation. Autocorrelation causes significant representation and computational difficulties for most SRL models. PRMs add a single random variable to the class model for every descriptive attribute in the dataset. Due to this, autocorrelation is shown with self loops in their class model. It is possible to achieve acyclic ground models even with the existence of self loops in the class model, however additional information on the dataset is required which complicates the model and is usually unknown or does not exist. A well known example is the correlation between blood type of parents and their children. Although the descriptive attribute of blood type requires a self loop in the class level model, the order on the ground atoms avoids introducing loops in the ground model. Neville and Jensen conclude that the acyclicity constraints of directed PRMs precludes the learning of arbitrary autocorrelation dependencies and thus severely limits the applicability of these models in relational domains [44]. MLNs require additional predicates in a clause to illustrate autocorrelation. For example,

$$BloodType(X) \leftarrow BloodType(Y) \wedge Parent(X, Y)$$

shows, in MLN format, that blood type of a person is correlated with the blood type of his parents. RDNs can easily handle autocorrelation. Table 2.6 summarizes how various SRL models handle autocorrelation.

SRL Method	**Autocorrelation**
PRM	Self-loops in class-level model
MLN	Additional variables needed
RDN	No requirements
BLP	Not discussed

Table 2.7: A comparison of how various SRL models handle autocorrelation.

Combining Problem. Combining problem is a challenging feature of learning relational data. PRMs add aggregation functions to their models and BLPs add both combining rules and aggregation functions. MLNs and RDNs do not need any additional requirements for dealing with combining problem. Table 2.6 summarizes how various SRL models deal with combining problem.

SRL Method	**Combining Problem**
PRM	Require aggregation
MLN	No requirements
RDN	Not discussed
BLN	Require combination rules

Table 2.8: A comparison of how various SRL models handle the combining problem.

Statistical relational learning has matured as a research field and some outstanding results are already achieved and many methods have been proposed. We reviewed four of the proposed models in details. The methods of representing, learning, and inference are mostly discussed. In Chapter 3, we propose a new structure learning method for MLNs using directed models.

Chapter 3

Learning Graphical Models for Relational Data via Lattice Search

In this chapter we present an efficient new algorithm for learning a Bayes Net that performs a level-wise search through the table join lattice for relational dependencies. From the Bayes net we obtain an MLN structure via a standard moralization procedure for converting directed models to undirected models. Learning MLN structure by moralization is 200-1000 times faster and scores substantially higher in predictive accuracy than benchmark MLN algorithms on six relational databases.

In Section 3.1 we discuss the main approach of the learn-and-join algorithm. In Section 3.2 we review related work, then in Section 3.3 we review lattice search for attribute dependencies. The learn-and-join algorithm is introduced in Section 3.4, and finally in section 3.5 we evaluate the performance of the learn-and-join algorithm to state-of-the-art structure learning methods in MLN and Inductive Logic Programming (ILP) methods, both in terms of processing speed and in terms of model predictive accuracy. We also perform lesion study that asses the effects of using only parts of the components of our main algorithm.

3.1 Introduction

We introduce a new *moralization approach* to learning MLNs: first we learn a directed Bayes net graphical model for relational data, then we convert the directed model to an undirected MLN model using the standard moralization procedure (marry spouses, omit

edge directions). The main motivation for performing inference with undirected models is that they do not suffer from the problem of cyclic dependencies in relational data [15, 105, 53]. Thus our approach combines the scalability and of directed model search, and the inference power and theoretical foundations of undirected relational models.

3.1.1 Approach

We present a new algorithm for learning a Bayes net from relational data, the *learn-and-join* algorithm. While our algorithm is applicable to learning directed relational models in general, we base it on the Parametrized Bayes Net formalism of Poole [84]. The learn-and-join algorithm performs a level-wise model search through the table join lattice associated with a relational database, where the results of learning on subjoins constrain learning on larger joins. The join tables in the lattice are (i) the original tables in the database, and (ii) joins of relationship tables, with information about descriptive entity attributes added by joining entity tables. A single-table Bayes net learner, which can be chosen by the user, is applied to each join table to learn a Bayes net for the dependencies represented in the table. For joins of relationship tables, this Bayes net represents dependencies among attributes conditional on the existence of the relationships represented by the relationship tables.

Single-table Bayes net learning is a well-studied problem with fast search algorithms. Moreover, the speed of single-table Bayes net learning is significantly increased by providing the Bayes net learner with constraints regarding which edges are required and which are prohibited. Key among these constraints are the *join constraints*: the Bayes net model for a larger table join inherits the presence or absence of edges, and their orientation, from the Bayes nets for subjoins. We present a theoretical analysis that shows that, even though the number of potential edges increases with the number of join tables, the join constraint reduces the Bayes net model search space to keep it approximately constant size throughout the join lattice. In addition to the join constraints, the relational structure leads to several others that reduce search complexity; we discuss the motivation for these constraints in detail. The graph of Figure 3.1 summarizes the system architecture.

We evaluated the structures obtained by moralization using two synthetic dataset and five public domain datasets. In our experiments on small datasets, the run-time of the learn-and-join algorithm is 200-1000 times faster than benchmark programs in the Alchemy framework [59] for learning MLN structure. On medium-size datasets, such as the MovieLens database, almost none of the Alchemy systems returns a result given our system resources,

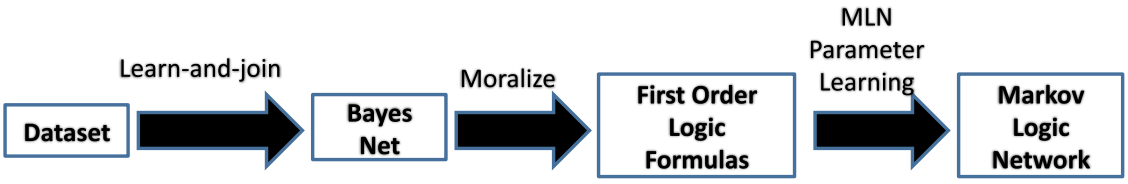


Figure 3.1: System architecture for learning a Markov Logic Network from an input relational database.

whereas the learn-and-join algorithm produces an MLN with parameters within 2 hours; most of this time (98%) is spent optimizing the parameters for the learned structure. To evaluate the predictive performance of the learned MLN structures, we used the parameter estimation routines in the Alchemy package. Using standard prediction metrics for MLNs, we found in empirical tests that the predictive accuracy of the moralized BN structures was substantially greater than that of the MLNs found by Alchemy. Our code and datasets are available for ftp download [48].

3.2 Related Work

The syntax of other directed SRL models, such as Probabilistic Relational Models (PRMs) [30], Bayes Logic Programs (BLPs) [46] and Logical Bayesian Networks [20], is similar to that of Parametrized Bayes Nets [84]. Our approach applies to directed SRL models generally.

3.2.1 Lattice Search Methods

The idea of organizing model/pattern search through a partial order is widely used in data mining, for instance in the well-known Apriori algorithm, in statistical-relational learning [86] and in Inductive Logic Programming (ILP) [63]. Search in ILP is based on the θ-subsumption or specialization lattice over clauses. Basically, a clause c specializes another clause c' if c adds a condition or if c replaces a 1st-order variable by a constant. The main similarity to the lattice of relationship joins is that extending a chain of relationships by another relationship is a special case of clause specialization. The main differences are as follows. (1) Our algorithm uses only a lattice over chains of relationships, not over conditions that combine relationships with attributes. Statistical patterns that involve attributes are

learned using Bayes net techniques, not by a lattice search. (2) ILP methods typically stop specializing a clause when local extensions do not improve classification/prediction accuracy. Our algorithm considers all points in the relationship lattice. This is feasible because there are usually only a small number of different relationship chains, due to foreign key constraints.

Lattice search is related to iterative deepening methods for statistical-relational learning [79, Sec.8.3.1], [9]. The main differences are as follows. (1) Current statistical-relational learning methods do not treat dependencies learned on smaller relational neighborhoods as constraining those learned on larger ones. Thus dependencies learned for smaller neighborhoods are revisited when considering larger neighborhoods. In principle, it appears that other statistical-relational learning methods could be adapted to use the relationship join lattice with inheritance constraints as in our approach. (2) To assess the relevance of information from linked entities, statistical-relational learning methods use aggregate functions (e.g., the average grade of a student in the courses they have taken), or combining rules (e.g., noisy-or) [46, 75]. In Probabilistic Relational Models, Bayes Logic Programs, and related models, the aggregate functions/combining rules add complexity to structure learning. In contrast, our statistical analysis is based on table joins rather than aggregation. Like Markov Logic Networks, our algorithm does not require aggregate functions or combining rules, although it can incorporate them if required.

3.2.2 MLN structure learning methods

Current methods [59, 70, 42, 4] successfully learn MLN models for binary predicates (e.g., link prediction), but do not scale well to larger datasets with descriptive attributes that are numerous and/or have a large domain of values. Mihalkova and Mooney [70] distinguish between top-down approaches, that follow a generate-and-test strategy of evaluating clauses against the data, and bottom-up approaches that use the training data and relational path finding to construct candidate conjunctions for clauses. In principle, the BN learning module may follow a top-down or a bottom-up approach; in practice, most BN learners use a top-down approach. The BUSL algorithm [70] employs a single-table Markov network learner as a subroutine. The Markov network learner is applied once after a candidate set of conjunctions and a data matrix has been constructed. In contrast, we apply the single-table BN learner repeatedly, where results from earlier applications constrain results of later applications.

We briefly describe the key high-level differences between our algorithm and previous MLN structure learning methods, focusing on those that lead to highly efficient relational learning.

Search Space. Previous Markov Logic Network approaches have so far followed Inductive Logic Programming techniques that search the space of clauses. Clauses define connections between *atoms* (e.g. $intelligence = hi, gpa = low$). Descriptive attributes introduce a large number of atoms, one for each combination of attribute and value, and therefore define a large search space of clauses. We utilize Bayes net learning methods that search the space of links between *predicates/functions* (e.g., $intelligence, gpa$), rather than atoms. Associations between predicates constitute a smaller model space than clauses that can be searched more efficiently. The advantages of searching the predicate space rather than the clause space are discussed by Kersting and deRaedt [46, 10.7].

We employ a number of constraints that are motivated by the relational semantics of the data. These further reduce the search space, mainly by requiring or forbidding edges in the Bayes net model. A key type of constraint is based on the lattice of relationship chains: These state that edges learned when analyzing shorter chains are inherited by longer chains. This allows the learn-and-join algorithm to perform a local statistical analysis for a single point in the relationship chain lattice, while connecting the results of the local analyses with each other.

Data Representation and Lifted Learning. The data format used by Markov Logic Networks is a list of ground atoms, whereas the learn-and-join algorithm analyzes data tables. This allows us to apply directly propositional Bayes net learners which take single tables as input. From a statistical point of view, the learn-and-join algorithm requires only the specification of the frequency of events in the database (the sufficient statistics in the database) [93]. The data tables provide these statistics. In the case of join tables the statistics are frequencies conditional on the existence of a relationship (e.g., the percentage of pairs of friends who both smoke). The learn-and-join algorithm can be seen as performing *lifted learning*, in analogy to lifted probabilistic inference [84]. Lifted inference uses as much as possible frequency information defined at the class level in terms of 1st-order variables, rather than facts about specific individuals. Likewise, the learn-and-join algorithm uses frequency information defined in terms of 1st-order variables (namely the number of satisfying groundings of a 1st-order formula).

3.3 Lattice Search for Attribute Dependencies

We describe the learn-and-join method for learning a Functor Bayes net that models correlations among descriptive attributes given the link structure. We begin with the data structures that the algorithm uses to represent relational objects. Then we give pseudocode for the algorithm and illustrate it with an example.

3.3.1 Overview

The components of the algorithm address the following main issues: (1) the representation of relationship sets, (2) bounding the complexity of relational contexts and (3) propagating constraints from smaller relationship sets in the multinet lattice to larger ones. We define and discuss the constraints on the graph structure that are used in the algorithm, separately from the description of the model search procedure. We describe the model search procedure as a lattice search, where the lattice points are chains of relationships. Conceptually, the lattice view makes the description simpler and more general without losing rigor. Computationally, the lattice diagram facilitates the implementation of the model search.

3.3.2 The Multinet Lattice

With each point in the relationship lattice, we associate a Bayes net model and a join data table. Thus the lattice structure defines a *multinet* rather than a single Bayes net. Multinets are a classic Bayes net formalism for modelling context-sensitive dependencies among variables. They have been applied for modelling diverse domains, such as sleep disorders, eye diseases, and turbines that generate electricity. Geiger and Heckerman contributed a standard reference article for the multinet formalism [28]. In their illustrative example, a building guard watches for three different types of people, visitors, spies, and workers. The existence of a dependency between the gender of a person and whether or not they wear an identification badge depends on the type of person. This scenario is modelled with three multinets, one for each type of person. The type of person is the context for the corresponding multinet.

Going back to the classic work of Ngo and Haddaway on context-sensitive dependencies in relational data [81], directed relational models usually include resources for representing the influence of context on dependencies [81, 20, 31, 75, 39, 92]. A common approach is to use a logical language for stating context conditions as well as dependencies between

random variables. In this chapter we employ multinets rather than context-sensitive rules for two reasons: (1) To stay close to standard graphical approaches for non relational Bayes nets. (2) To ensure that the dependencies found for a given context define a valid acyclic Bayes net structure.

In the learn-and-join algorithm, the context of a multinet is defined by a chain of relationship functor nodes. Distinguishing these different contexts allows us to represent that the existence of certain dependencies among attributes of entities depend on which kind of links exist between the entities. The final output of the learn-and-join algorithm is a single Bayes net derived from the multinets. The output of the algorithm can also be converted to other formalisms, including those based on rules for context-sensitive dependencies.

The Functor Nodes

Throughout the discussion we assume that a set of functor random variables F is fixed. The random variables F are partitioned into the following:

1. Functor nodes representing descriptive attributes of entities. Descriptive attribute functor nodes take a single population variable as argument.

2. Functor nodes representing descriptive attributes of links. Relational descriptive attributes take two or more population variables as arguments.

3. Boolean relationship functor nodes that indicate whether a relationship holds. Boolean relationship functor nodes also take take two or more population variables as arguments.

We make the assumption that functor nodes only contain variables and no functor node contains the same variable twice. These assumptions are met in typical SRL models. They do not actually involve a loss of modelling power because a functor node with a constant or a repeated variable can be rewritten using a new functor symbol (provided the functor node contains at least one variable). For instance, a functor node $\mathit{Friend}(X, \mathit{jack})$ can be replaced by introducing a new unary functor symbol $\mathit{Friend}_{\mathit{jack}}(X)$. Similarly, $\mathit{Friend}(X, X)$ can be replaced by the unary functor symbol $\mathit{Friend}_{\mathit{self}}(X)$.

Examples. The nodes of the Bayes net of Figure 3.2 are the functor nodes generated from the DB schema of Table 2.1 with one population variable per entity type (e.g., S for *Student*).

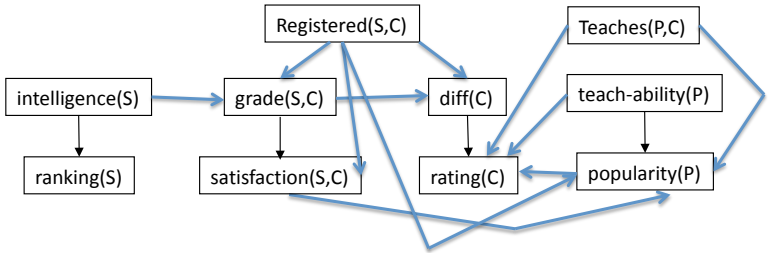


Figure 3.2: A Functor Bayes net graph for the relational schema of Table 2.1.

Relationship Chains

A relationship set is a *chain* if it can be ordered as a list $[R_1(\boldsymbol{\tau}_1), \ldots, R_k(\boldsymbol{\tau}_k)]$ such that each functor $R_{i+1}(\boldsymbol{\tau}_{i+1})$ shares at least one population variable with the preceding terms $R_1(\boldsymbol{\tau}_1), \ldots, R_i(\boldsymbol{\tau}_i)$. All sets in the lattice are constrained to form a chain. For instance, in the University schema of Table 2.1, a chain is formed by the two relationship nodes

$$RA(P, S), Registered(S, C).$$

If relationship node $TA(C, S)$ is added, we may have a three-element chain

$$RA(P, S), Registered(S, C), TA(C, S).$$

The subset relation defines a lattice on relationship sets, that is, each point in the lattice is a set of relationship functors. The lattice join and meet are just union and intersection as usual for a powerset lattice. Only relationship sets that define a chain are suitable contexts for statistical analysis, so we show only the relationship chains in lattice diagrams. The empty relationship set corresponds to a context in which we do not consider links. This means that each entity table is analyzed independently. Instead of including the empty relationship set in diagrams, we include the entity tables at the bottom, and connect them to relationships that link these entities. Propagating associations from entity tables to relationships is intuitively how the learn-and-join algorithm enforces hierarchical constraints. Figure 3.3 illustrates the lattice for the relationship nodes in the University schema of Figure 3.2. For reasons that we explain below, entity tables are also included in the lattice and linked to relationships that involve the entity in question.

The concept of a relationship is related to but different from the notion of a slot chain as used in Probabilistic Relational Models [30]. The main difference is that a slot chain can

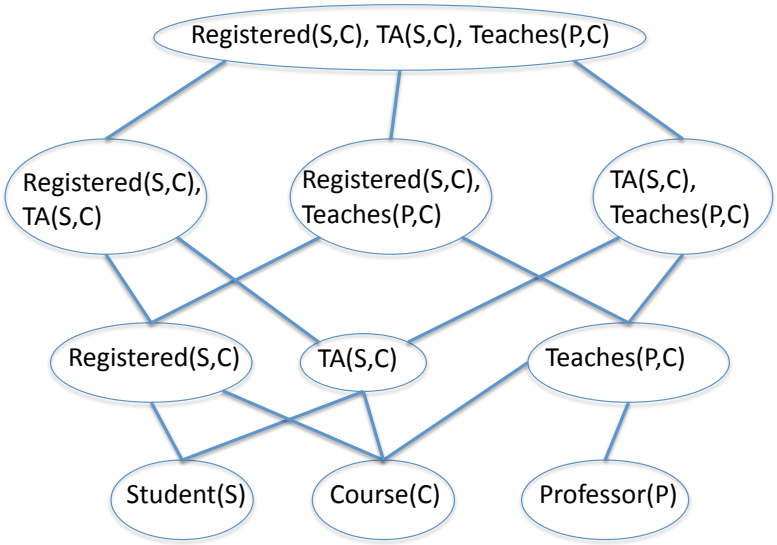


Figure 3.3: A lattice of relationship sets for the University schema of Table 2.1 (without the RA relation). Links from entity tables to relationship tables correspond to foreign key pointers.

connect entity tables as well as relationship tables. Thus a path from the Student table to the Registered table to the Course table constitutes a slot chain of length 3, but contains only a single relationship (relationship chain of length 1).

The Join Data Table

With each relationship set $\mathbf{R}$ is associated a join data table $\bowtie_{\mathbf{R}}$. The table represents the frequencies (sufficient statistics) with which combinations of attribute values occur, conditional on all relationships in $\mathbf{R}$ being true. Let R_i denote the data table associated with relationship functor $R_i(\boldsymbol{\tau}_i)$. For relationship functors $\mathbf{R} = \{R_1(\boldsymbol{\tau}_1), \ldots, R_k(\boldsymbol{\tau}_k)\}$ let $X_1, \ldots, X_l$ be the population variables that occur in the k relationship functors, and write E_j for the entity data table associated with the population variable X_j. Then the join table for the relationship set, or **relationship join**, is given by

$$\bowtie_{\mathbf{R}} \equiv \bowtie_{i=1}^{k} R_i \bowtie_{j=1}^{l} E_j.$$

If two or more variables are associated with the same population, then the same descriptive attribute will appear at least twice in the relationship join. In this case we disambiguate the names of the descriptive attributes by adding the variable as their argument. Similarly, we add variables to disambiguate repeated occurrences of descriptive link attributes. Thus each column label in the relationship join corresponds to exactly one functor node. For each relationship set $\mathbf{R} = \{R_1(\boldsymbol{\tau}_1), \ldots, R_k(\boldsymbol{\tau}_k)\}$, the nodes in the associated Bayes net $B_{\mathbf{R}}$ are the column labels in $\bowtie_{\mathbf{R}}$, plus Boolean relationship indicator nodes $R_1(\boldsymbol{\tau}_1), \ldots, R_k(\boldsymbol{\tau}_k)$.

Examples. For the relationship chain $[RA(P, C), Registered(S, C)]$, the join data table is given by

$$RA \bowtie Registered \bowtie Professor \bowtie Student \bowtie Course.$$

3.3.3 Model Conversion

The output of the lattice search is the Bayes net associated with the largest relationship chain that forms the apex of the relationship lattice. The Bayes net of Figure 3.2 is associated with the relationship set $Registered(S, C)$, $Teaches(P, C)$, which is the maximal conjunction of both relationship functors in this functor set. In Figure 3.3 the maximally large relationship set has three members. For Markov Logic Networks, we convert the maximal Bayes net to an MLN using **Moralization**. Moralization converts a directed acyclic graph into an undirected model. To convert a Functor Bayes Net into an MLN using moralization, add a clause to the MLN for each assignment of values to a child and its parents. The MLN for a moralized Bayes net B thus contains a clause for *each* conditional probability in B [15].

The Bayes net can similarly be converted to other clausal formalisms like Bayesian Logic programs, since a Functor Bayes net defines a set of directed clauses of the form $child \leftarrow parents$ [46].

3.4 The Learn-And-Join Algorithm

This section presents the Learn-and-Join Algorithm (LAJ) that takes as input a relational database and constructs a Functor Bayes multinet for each relationship set in the multinet lattice. The algorithm enumerates relationship lists. This can be done using any standard technique, such as those developed for enumerating item sets in association rule mining [1]. It proceeds level-wise by considering relationship sets of length $s = 1, 2, \ldots$. After Bayes nets have been learned for sets of length s, the learned edges are propagated to sets of length

$s+1$. In the initial case of single relationship tables where $s = 1$, the edges are propagated from Bayes nets learned for entity tables. In addition to the edge constraints, the algorithm enforces a number of constraints that are motivated by the relational structure of the functor nodes.

We next provide a compact description of the constraints used, including their definition, an intuitive interpretation and examples. Then we show by examples how the constraints operate. Finally, we summarize the algorithm with pseudocode as Algorithm 1. Later sections discuss the constraints in detail, including motivation, mathematical analysis and references to related concepts that have appeared in the literature.

3.4.1 Constraints Used in the Learn-And-Join Algorithm

The constraints fall into two types: relational constraints that capture the semantics of the relational schema, and lattice constraints on the presence/absence of edges that connect the results of learning from different points in the relationship set lattice.

Edge Inheritance from Entity Tables

This type of constraints state the presence or absence of edges in graphs associated with join tables lower in the lattice is inherited by graphs associated with join tables higher in the lattice. The intuition behind these constraints is that dependencies should be assessed in the most specific context possible; edges from an entity table are inherited by relationship tables that involve the entity in question.

Constraint 1 *Let X be the population variable for an entity type associated with entity table E. Let $\mathbf{R}$ be any relationship set that contains the variable X. Then the Bayes net associated with $\mathbf{R}$ contains an edge $f(X) \rightarrow g(X)$ connecting two descriptive attributes of X if and only if the Bayes net associated with E contains the edge $f(X) \rightarrow g(X)$.*

Example. If the *Student* Bayes net contains an edge $\mathit{Intelligene}(Y) \rightarrow \mathit{Ranking}(Y)$, then the Bayes net associated with the relationship *Registered* must also contain this edge. If the edge is absent in the *Student* Bayes net, it must also be absent in the Bayes net associated with the relationship *Registered*.

Edge Inheritance in the Relationship Lattice

The next constraint states that edges learned on smaller relationship sets are inherited by larger relationship sets. If the smaller sets are ambiguous with regard to the direction of an adjacency, the larger relationship set must contain the adjacency; the direction is then resolved by applying Bayes net learning to the larger relationship set.

Constraint 2 *Suppose that nodes $f(\boldsymbol{\tau}), g(\boldsymbol{\tau}')$ appear in the join table $\bowtie_{\mathbf{R}}$. Then*

1. *If $f(\boldsymbol{\tau})$ and $g(\boldsymbol{\tau}')$ are not adjacent in any graph $B_{\mathbf{R}^*}$ associated with a relationship subset $\mathbf{R}^* \subset \mathbf{R}$, then $f(\boldsymbol{\tau})$ and $g(\boldsymbol{\tau}')$ are not adjacent in the graph associated with the relationship set $\mathbf{R}$.*

2. *Else if all subset graphs agree on the orientation of the adjacency $f(\boldsymbol{\tau}) - g(\boldsymbol{\tau}')$, the graph associated with the relationship set $\mathbf{R}$ inherits this orientation.*

3. *Else the graph associated with the relationship set $\mathbf{R}$ must contain the edge $f(\boldsymbol{\tau}) \rightarrow g(\boldsymbol{\tau}')$ or the edge $f(\boldsymbol{\tau}) \leftarrow g(\boldsymbol{\tau}')$.*

Examples. Consider the lattice shown in Figure 3.3. Suppose that the graph associated with the relationship *Registered*(S, C) contains an edge *difficulty*$(C) \rightarrow$ *intelligence*(S), and that the graph associated with the relationship *TA*(S, C) does not contain the edge *difficulty*$(C) \rightarrow$ *intelligence*(S). Then the edge *difficulty*$(C) \rightarrow$ *intelligence*(S) must be present in the Bayes net associated with the larger relationship set *Registered*(S, C), *TA*(S, C). If the edge is contained in neither of the graphs associated with *Registered*(S, C), and *TA*(S, C), it must not be present in the graph associated with the *Registered*(S, C), *TA*(S, C).

Population Variable Bound

We allow the user to specify a bound on the number of population variables that occur in a family (child + parent nodes). Intuitively, this bounds the number of distinct (generic) objects that can be considered in a single child-parent configuration. For instance, if the bound is 1, the family expresses patterns only about a single entity. With 2 population variables, patterns involving pairs can be expressed, with 3 triples can be modelled, etc.

Examples. For the node *ranking*(S) of Figure 3.2, its family contains a single population variable Y, so only patterns involving a generic student can be represented. For the node

diff(C), its family contains two population variables S and C, so patterns involving pairs of students and courses may be represented.

We emphasize that a variable number bound does not imply any bound on the length of clauses: even with a single population variable like S associated with students, we can have an arbitrary number of attributes of students in a single clause. Kok and Domingos [60] highlight the importance of learning long clauses for relational models.

Link Attributes

There is a deterministic dependency between a Boolean relationship indicator node and a descriptive attribute associated with the relationship: If the relationship does not exist between two entities, then the value of the descriptive attribute is undefined. In our representation, this means that the descriptive attribute takes on the value $\perp$ for undefined (cf. Section 2.1.6). This deterministic connection can be enforced given the following graphical constraint.

Constraint 3 *Suppose that f_R denotes a descriptive attribute of relationship R and that $f_R(\boldsymbol{\tau})$ is a functor node for f_R and $R(\boldsymbol{\tau})$ is a main functor node for R. Then there is an edge $R(\boldsymbol{\tau}) \rightarrow f_R(\boldsymbol{\tau})$ in any Bayes net that contains $f_R(\boldsymbol{\tau})$.*

Examples. In the Bayes net of Figure 3.2, the functors *satisfaction* and *grade* denote descriptive attributes of the *Registered* relationship. So the Bayes net must contain edges $\mathit{Registered}(S, C) \rightarrow \mathit{satisfaction}(S, C)$ and $\mathit{Registered}(S, C) \rightarrow \mathit{grade}(S, C)$, which are the main nodes for their respective functors.

Relationship Parents

This constraint states that descriptive attributes of different populations are independent of each other and can only be conditionally dependent thorough a common relationship.

Constraint 4 *Let X and Y be population variables for entity types associated with entity tables E_1 and E_2. Let R be any relationship set that contains both variables X and Y. If the Bayes net associated with with R contains an edge $f(X) \rightarrow g(Y)$ connecting a descriptive attributes of X to a descriptive attribute of Y, then the edge $R(\boldsymbol{\tau}) \rightarrow g(Y)$ is forced to be added too.*

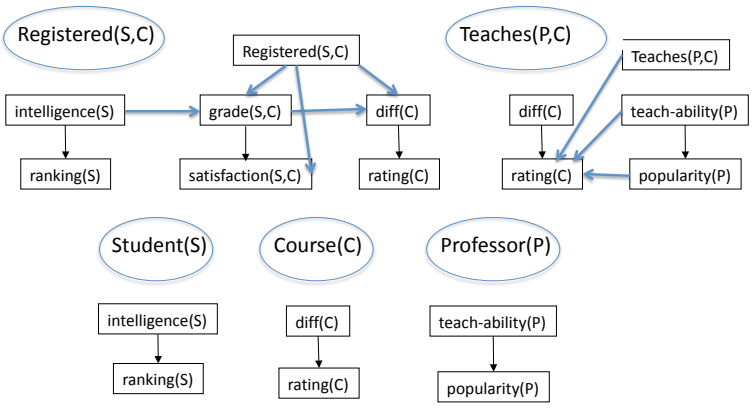


Figure 3.4: The multinet lattice for the University Schema, restricted to entity and relationship functors. Join data tables are not shown. Figure 3.2 shows the final Bayes net after the multinets have been combined.

Example. A course and a professor are two independent entities, so popularity of a professor and rating of a course are independent. However, if there is a relationship between the two, $Teaches(P, C)$, the two may conditional on the relationship be dependent. It may be true that all of the courses that a popular professor teaches are well rated. In the Bayes net of Figure 3.2, there is an edge between $popularity(P) \rightarrow rating(C)$. As a result the edge $Teaches(P, C) \rightarrow rating(C)$ is also added to the Bayes net.

We discuss and analyze the constraints further in later sections. The next subsection presents examples of how the constraints operate in the learn-and-join algorithm.

3.4.2 Example of Learn-and-join Algorithm

We illustrate the learn-and-join algorithm on the example database of Figure 2.3; with Schema 2.1; Figure 3.4 and our example demonstrate the multinets for the University schema up to level 1. Continuing the construction up to the highest level 2 produces a single Bayes net for the maximal relationship set $Registered(S, C)$, $Teaches(P, C)$ that is shown in Figure 3.2.

1. Applying the single-table Bayes net learner to the *Student*, *Course*, and *Prof* table produces three corresponding Bayes nets (Figure 3.4).

2. Then form the join data table $J = \mathit{Student} \bowtie \mathit{Course} \bowtie \mathit{Registered}$ The Bayes net learner is applied to J, with the following constraints.

   (a) From the *Student* Bayes net, there must be an edge $\mathit{intelligence}(S) \rightarrow \mathit{ranking}(S)$, where S is the population variable associated with *Student*. (Constraint(1)).

   (b) From the *Course* Bayes net, there must be an edge $\mathit{diff}(C) \rightarrow \mathit{rating}(C)$, where C is the population variable associated with *Course*. (Constraint(1)).

   (c) The edges $\mathit{Registered}(S, C) \rightarrow \mathit{grade}(S, C)$ and $\mathit{Registered}(S, C) \rightarrow \mathit{satisfaction}(S, C)$ must be present. (Constraint 3)

3. Then form the join data table

$$J = \mathit{Course} \bowtie \mathit{prof} \bowtie \mathit{Teaches}$$

   (a) From the *Prof* Bayes net, there must be an edge $\mathit{teach} - \mathit{ability}(P) \rightarrow \mathit{popularity}(P)$, where P is the population variable associated with *Student*. (Constraint(1)).

   (b) From the C Bayes net, there must be an edge $\mathit{diff}(C) \rightarrow \mathit{rating}(C)$, where C is the population variable associated with *Course*. (Constraint(1)).

   (c) The edge $\mathit{Registered}(S, C) \rightarrow \mathit{rating}(C)$ must be present. (Constraint 4)

Algorithm 1 combines all the algorithm's components in pseudocode.

We next discuss the constraints in more detail, including a mathematical analysis of the most important ones.

3.4.3 Discussion: Lattice Constraints

A key feature of the learn-and-join algorithm is the lattice inheritance constraints that a Bayes net for a table join must respect the links found for the joined tables. We describe a computational and a statistical motivation for them.

Computational Complexity. We discuss the complexity of the learn-and-join algorithm relative to the complexity of applying the single Bayes net table learner. The edge-inheritance constraint reduces the search complexity considerably. To illustrate, consider the impact of Clause 1 for two entity tables that contain k descriptive attributes each. Then in an unconstrained join with a relationship table, the search space of possible adjacencies

Algorithm 1: Pseudocode for structure learning with lattice search

Input: Database $\mathcal{D}$ with entity tables $E_1, ..E_e$; functors F; #variable bound $maxVar$.
Output: MLN formulas for $\mathcal{D}$; a Bayes multi-net $B_{\mathbf{R}}$ for relationship subsets of F.

Calls BNL: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.
Notation: $BNL(T, Econstraints)$ is the output DAG of the Bayes net learner given data table T and constraints $Econstraints$.

1: Econstraints := Edge constraints from Constraint(4) and Constraint(3).
2: **for** $i \leftarrow 1$ **to** e **do**
3: $B_{E_i} := BNL(E_i, Econstraints)$.
4: **end for**
5: Econstraints += Lattice Constraints from entity tables (Constraint(1)).
6: **for** list size $s \leftarrow 1, 2 \ldots$ **do**
7: Enumerate relationship lists $[\mathbf{R}]_1, \ldots, [\mathbf{R}]_{s_k}$ of size s, such that for each i:
(1) $[\mathbf{R}]_i$ is a chain.
(2) the number of population variables in $[\mathbf{R}]_i$ is no greater than $maxVar$.
8: **if** there is no such list of size s **then**
9: terminate computation
10: **else**
11: **for** $i \leftarrow 1$ **to** s_k **do**
12: $B_{[\mathbf{R}_i]} := BNL(\bowtie_{\mathbf{R}_i}, Econstraints)$ + edges from the $\mathbf{R}_i$ functors.
13: **end for**
14: **end if**
15: Econstraints += Lattice Constraints from relationship joins of size s (Constraint(2)).
16: **end for**
17: Let B_{max} be the Bayes net associated with the maximal relationship set.
18: Add all family formulas of $B_{[\mathbf{R}]}$ to MLN.

has size $\binom{2k}{2}$, whereas with the constraint, the search space size is $k^2/2$, which is smaller than $\binom{2k}{2}$ because the quadratic k^2 factor has a smaller coefficient. For example, with $k = 6$, we have $\binom{2k}{2} = 66$ and $k^2/2 = 18$. Therefore we may assume that the model search for each join table is of roughly similar complexity. This means that *the overall runtime of the algorithm is the cost of learning a Bayes net on a single table, times the number of join tables.* Because of foreign key constraints, the number of valid relationship chains in the lattice that need to be analyzed is generally much smaller than the powerset of all possible relationship sets. For the learn-and-join algorithm, the main computational challenge in scaling to larger table joins is therefore not the increasing number of columns (attributes) in the join, nor the number of join tables, but only the increasing number of rows (tuples) in more complex joins.

Statistical Motivation. In addition to , a statistical motivation for the edge-inheritance constraint is that the marginal distribution of descriptive attributes may be different in an entity table than in a relationship table. For instance, if a highly intelligent student s has taken 10 courses, there will be at least ten satisfying groundings of the conjunction $\mathit{Registered}(S, C), \mathit{intelligence}(S) = \mathit{hi}$. If highly intelligent students tend to take more courses than less intelligent ones, then in the *Registered* table, the frequency of tuples with intelligent students is higher than in the general student population. In general, the distribution of database frequencies conditional on a relationship being true may be different from its unconditional distribution. The edge inheritance constraint ensures that the subgraph of the final parametrized Bayes net whose nodes correspond to the attributes of the E table is exactly the same as the graph that the single-table Bayes net learner constructs for the E table.

The motivation for Clause 2 is similar: a dependency ought to be evaluated on a minimal context. For instance, the presence of an edge $\mathit{intelligence}(S) \rightarrow \mathit{difficulty}(C)$ given that $\mathit{Registered}(S, C) = T$ ought to depend only on the *Registered* relationship and not on a relationship that involves another object, such as a Teaches for the course (i.e., the edge is inherited in the larger context $\mathit{Registered}(S, C) = T, \mathit{Teaches}(P, C)$).

3.4.4 Discussion: Population Variable Bound.

As both the statistical and the computational complexity of Functor Bayes nets can be high, it is desirable to allow a user to specify a complexity bound to control the trade-off between

expressive power and computational difficulty. A key issue is the length of the relationship chains that the algorithm needs to consider. The number of relationship chains grows exponentially with this parameter. The computational complexity of computing sufficient statistics for a relationship set depends on the length of slot chains as well: with no bound on the length, the problem is #P-complete [15, Prop.12.4]. We expect that more distantly related entities carry less information, so many SRL algorithms assume a small bound on the length of possible slot chains, on the order of 3 or so. A less restrictive way to bound the complexity of relationship chains is to allow the user to specify a bound on the number of population variables that occur in a family, as proposed by Vardi [109].

Intuitively, this bounds the number of distinct (generic) objects that can be considered in a single child-parent configuration. The computational complexity of computing sufficient statistics for a relationship set depends on the number of population variables as well: with no bound, the problem is #P-complete [15, Prop.12.4]. With a constant bound, sufficient statistics can be computed in polynomial time [109, 50].

The next section presents empirical evidence about the performance of the learn-and-join algorithm on benchmark datasets.

3.5 Evaluation: Experimental Design

All experiments were done on a QUAD CPU Q6700 with a 2.66GHz CPU and 8GB of RAM. Our code and datasets are available on the world-wide web [48]. We made use of the following existing implementations.

Single Table Bayes Net Search GES search [11] with the BDeu score as implemented in version 4.3.9-0 of CMU's Tetrad package (structure prior uniform, ESS=10; [106]).

MLN Parameter Learning The default weight training procedure [67] of the Alchemy package [61], Version 30.

MLN Inference The MC-SAT inference algorithm [85] to compute a probability estimate for each possible value of a descriptive attribute for a given object or tuple of objects.

Join Data Tables The join data tables for a given relationship chain are computed using a straightforward SQL inner join over the required relationship and entity tables. Our database management system is MySQL Version 1.2.15.

The computation of the join data tables could be optimized in a number of ways. For instance, like most Bayes net scores, the BDeu score requires only the sufficient statistics, or database frequencies of events. Rather than materializing the entire join data table, we could use indices or the SQL *Count* aggregate function to compute these summary statistics only. We did not include optimizations of the database computations because data management is not the focus of our work, and we already achieve very fast learning without them. Thus the database is used only to find the set of tuples that satisfy a given joint condition (e.g., find the set of (x, y) pairs such that $Friend(X, Y) = T, Smokes(X) = T, Smokes(Y) = F$).

3.5.1 Datasets

We used two synthetic and five benchmark real-world datasets for our evaluation.

University Database. We manually created a small dataset, based on the schema given in Table 2.1.

MovieLens Database. The MovieLens dataset is from the UC Irvine machine learning repository. It contains two entity tables: *User* with 941 tuples and *Item* with 1,682 tuples, and one relationship table *Rated* with 80,000 ratings. The *User* table has 3 descriptive attributes $Age, gender, occupation$. We discretized the attribute age into three bins with equal frequency. The table *Item* represents information about the movies. It has 17 Boolean attributes that indicate the genres of a given movie. We performed a preliminary data analysis and omitted genres that have only weak correlations with the rating or user attributes, leaving a total of three genres.

Mutagenesis Database. This dataset is widely used in ILP research [103]. Mutagenesis has two entity tables, *Atom* with 3 descriptive attributes, and *Mole*, with 5 descriptive attributes, including two attributes that are discretized into ten values each (logp and lumo). It features two relationships *MoleAtom* indicating which atoms are parts of which molecules, and *Bond* which relates two atoms and has 1 descriptive attribute.

Hepatitis Database. This dataset is a modified version of the PKDD02 Discovery Challenge database. We adopted the modifications of Frank *et al.* [25], which includes removing tests with null values. It contains data on the laboratory examinations of hepatitis B and C infected patients. The examinations were realized between 1982 and 2001 on 771 patients.

The data are organized in 7 tables (4 entity tables, 3 relationship tables and 16 descriptive attributes). They contain basic information about the patients, results of biopsy, information on interferon therapy, results of out-hospital examinations, results of in-hospital examinations.

Mondial Database. This dataset contains data from multiple geographical web data sources [69]. We follow the modification of [100], and use a subset of the tables and features. Our dataset includes a self-relationship table *Borders* that relates two countries.

UW-CSE database. This dataset lists facts about the Department of Computer Science and Engineering at the University of Washington (UW-CSE) (e.g., Student, Professor) and their relationships (i.e. AdvisedBy, Publication). The dataset was obtained by crawling pages in the department's Web site (www.cs.washington.edu). Publications and AuthorOf relations were extracted from the BibServ database (www.bibserv.org).

Table 3.1 lists the databases and their sizes in terms of total number of tuples and number of ground atoms, which is the input format for Alchemy.

Dataset	**#tuples**	**#Ground atoms**
University	171	513
Movielens	82623	170143
Mutagenesis	15218	35973
Hepatitis	12447	71597
Mondial	814	3366
UW-CSE	2099	3380

Table 3.1: Size of datasets in total number of table tuples and ground atoms. Each descriptive attribute is represented as a separate function, so the number of ground atoms is larger than that of tuples.

Because several of the Alchemy systems returned no result on some of the real-world datasets, we formed two sub databases for each by randomly selecting entities for each dataset. We restricted the relationship tuples in each subdatabase to those that involve only the selected entities. Table 3.2 lists the resulting sub databases and their sizes in terms of total number of tuples and number of ground atoms.

Dataset	#tuples	#Ground atoms
MovieLens1 (subsample)	1468	3485
MovieLens2 (subsample)	12714	27134
Mutagenesis1 (subsample)	3375	5868
Mutagenesis2 (subsample)	5675	9058
Hepatitis1	6162	41335

Table 3.2: Size of subdatasets in total number of table tuples and ground atoms. Each descriptive attribute is represented as a separate function, so the number of ground atoms is larger than that of tuples.

3.5.2 Graph Structures Learned by the Learn-and-Join algorithm

Figure 3.5 shows the learned Bayes nets for the Hepatitis and MovieLens datasets. The graphs illustrate how the learn-and-join algorithm learns models with complex dependency patterns.

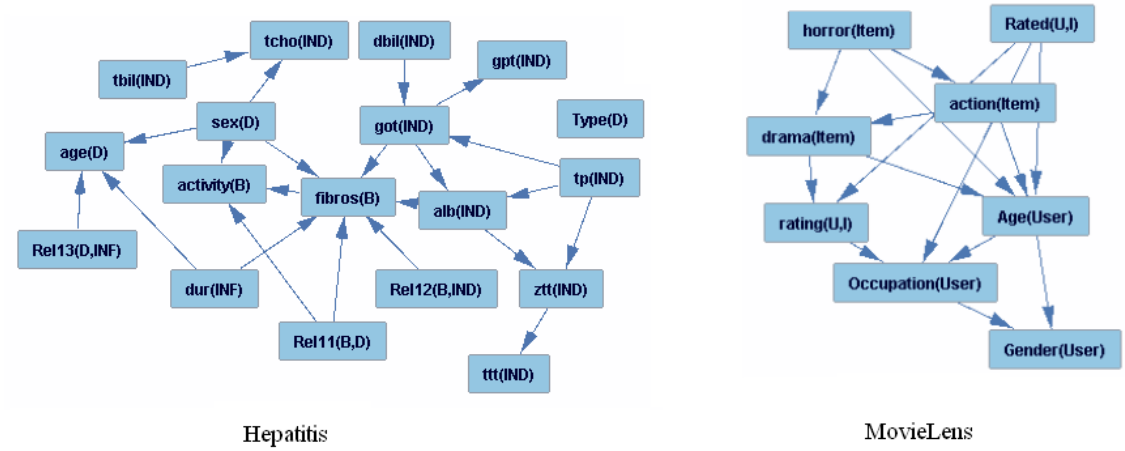


Figure 3.5: The Bayes net structures learned by the learn-and-join algorithm for Hepatitis and MovieLens datasets. We omitted edges from Boolean relationship nodes to descriptive attributes of the relationship to make the figure more easily readable.

3.6 Moralization vs. Other Structure Learning Methods: Basic Comparisons

We begin with a set of comparisons on standard benchmarks that follows the design and performance metrics of previous MLN structure learning studies [53, 70, 59]. Then we focus on experiments that assess specific components of the learn-and-join algorithm.

3.6.1 Comparison Systems and Performance Metrics

We compared the learn-and-join algorithm with 4 previous Markov Logic Network structure learning methods implemented in different versions of Alchemy.

MBN uses the learn-and-join algorithm to learn a Functor Bayes net. To perform inference, the Functor Bayes net is converted to a Markov Logic Network using moralization, which we refer to as the Moralized Bayes Net (see Section 2.1). Weights for the moralized Bayes net are learned using Alchemy's weight learning routine.

MSL uses beam search which begins by adding unit clauses to an Markov Logic Network. MSL then considers all clauses of length two and always maintains the n highest-scoring ones in the set. MSL terminates when it cannot find a clause that improves upon the current Markov Logic Network's score [57].

LHL Lifted Hypergraph Learning [59] uses relational path finding to induce a more compact representation of data, in the form of a hypergraph over clusters of constants. Clauses represent associations among the clusters.

BUSL Applies relational path finding to ground atoms and variabilizes each ground atom in a path. It then constructs a Markov network for the nodes in the path, then computes a single data table for the path and learns edges between the nodes using a single-table Markov network learner [70].

LSM Learning Structural Motifs [60] uses random walks to identify densely connected objects in data, and groups them and their associated relations into a motif.

We use 3 performance metrics: Runtime, Accuracy, and Conditional log likelihood. These measures have been used in previous studies of Markov Logic Network learning [70, 59, 53].

Runtime The time taken to learn the model from the training dataset.

Accuracy (ACC) To define accuracy, we apply MLN inference to predict the probability of an attribute value, and score the prediction as correct if the most probable value is the true one. For example, to predict the gender of person Bob, we apply MLN inference to the atoms $gender(Bob, male)$ and $gender(Bob, female)$ (cf. Section 3.5.1). The result is correct if the predicted probability of $gender(Bob, male)$ is greater than that of $gender(Bob, female)$.

Conditional Log-Likelihood (CLL) The conditional log-likelihood of a ground atom in a database $\mathcal{D}$ given an Markov Logic Network is its log-probability given the Markov Logic Network and $\mathcal{D}$. The CLL directly measures how precise the estimated probabilities are.

For ACC and CLL the values we report are averages over all attribute predicates. Khosravi *et al.* [53] also report the AUC (area-under-curve) for predicates that correspond to descriptive attributes with binary values (e.g. gender), for the databases MovieLens and Mutagenesis [53]. As there are only few binary descriptive attributes, we omit AUC from this study. For the existing binary predicates, the AUC improvement of the MBN approach over previous Markov Logic Network methods is similar to that for ACC and CLL.

3.6.2 Runtime Comparison

Table 3.3 shows the time taken in minutes for learning in each dataset. The Alchemy times include both structure and parameter learning. For the MBN approach, we report both the Bayes net structure learning time and the time required for the subsequent parameter learning carried out by Alchemy.

Structure learning is very fast for both the MBN and the LSM method, orders of magnitude faster than for the other methods. The total runtime for MBN is dominated by the time required by Alchemy to find a parametrization for the moralized Bayes net. On the smaller databases, this takes between 5-12 minutes. On MovieLens, parametrization takes two hours, and on Mutagenesis, over 1.5 hours. While finding optimal parameters for the MBN structures remains challenging, the combined structure+weight learning system is much faster than the overall structure + weight learning time of most of the Alchemy

	Alchemy Methods				LAJ
Dataset	**MSL**	**LHL**	**BUSL**	**LSM**	**MBN**
University	5.02	3.54	0.5	**0.01**	**0.03 + 0.032**
MovieLens	NT	NT	NT	**0.45**	**1.2 +120**
MovieLens1	44	34.52	50	**0.03**	**0.05 + 0.33**
MovieLens2	2760	3349	NT	**0.06**	**0.12 + 5.10**
Mutagenesis	NT	NT	NT	**0.53**	0.5 + 106
Mutagenesis1	3360	3960	150	**0.12**	**0.1 + 5**
Mutagenesis2	NT	NT	NT	**0.17**	**0.2 +12**
Hepatitis	NT	NT	NT	**0.15**	**0.4 + 95.6**
Hepatitis1	NT	NT	NT	**0.1**	**0.2 + 4.8**

Table 3.3: Runtime to produce a parametrized Markov Logic Network, in minutes. The MBN column shows structure learning time + weight learning time. NT indicates non-termination.

methods: They do not scale to the complete datasets, and for the sub databases, the MBN approach is faster by a factor ranging from 200 to 1000.

These results are strong evidence that the MBN approach leverages the scalability of Bayes net learning to achieve scalable Markov Logic Network learning on databases of realistic size. The LSM method is very fast for all datasets. Inspection of the learned clauses by LSM shows that the rules are mostly just the unit clauses that model marginal probabilities. This indicates under fitting the data, as the following measurements of accuracy and conditional log-likelihood confirm.

3.6.3 Predictive Accuracy and Data Fit

Previous work on Markov Logic Network evaluation has used a "leave-one-out" approach that learns Markov Logic Networks for a number of sub databases with a small subset omitted [70]. This is not feasible in our setting because even on a training set of size about 15% of the original, finding a Markov Logic Network structure using the slower Alchemy methods is barely possible. Given these computational constraints, we investigated the predictive performance by learning an Markov Logic Network on one randomly selected 2/3 of the sub databases as a training set, testing predictions on the remaining 1/3. While this does not provide strong evidence about the generalization performance in absolute terms, it gives information about the relative performance of the methods. In the next section we give further cross validation results using the faster Alchemy methods LSM and LHL. Tables 3.4

and 3.5 report the average accuracy and conditional log-likelihood of each real-world dataset. (The synthetic dataset University was too small for learning on a subset).

Higher numbers indicate better performance and NT indicates that the system was not able to return an Markov Logic Network for the dataset, either crashing or timing out after 4 days of running. *MBN achieved substantially better predictions on all test sets, in the range of 10-20% for accuracy.*

The CLL performance of LSM is acceptable on average. The parameter estimates are biased towards uniform values, which leads to predictions whose magnitudes are not extreme. Because the average accuracy is low, this means that when mistaken predictions are made, they are not made with great confidence. The LSM pattern of low accuracy and acceptable log-likelihood is found in our other datasets as well.

Accuracy	Alchemy Methods				LAJ
Dataset	**MSL**	**LHL**	**BUSL**	**LSM**	**MBN**
Movielens11	0.40	0.42	0.34	0.37	**0.63**
Movielens12	0.41	0.44	NT	0.49	**0.62**
Movielens	NT	NT	NT	0.30	**0.69**
Mutagenesis1	0.34	0.31	0.37	0.30	**0.69**
Mutagenesis2	NT	0.35	NT	0.28	**0.65**
Hepatitis	NT	NT	NT	0.32	**0.54**
Hepatitis1	NT	NT	NT	0.29	**0.53**

Table 3.4: The table compares accuracy performance of the moralization approach (MBN) vs. previous Markov Logic Network learning methods. The data are obtained by training on 2/3 of the database and testing on the other 1/3. ACC is reported as an average over all attribute predicates of the datasets.

Where the learning methods return a result on a database, we also measured the predictions of the different Markov Logic Network models for the facts in the training database. This indicates how well the Markov Logic Network summarizes the statistical patterns in the data. These measurements test the log-linear as a solution to the combining problem for inference. While a small improvement in predictive accuracy may be due to over fitting, the very large improvements we observe are evidence that the Markov Logic Network models produced by the Alchemy methods underfit and fail to represent statistically significant dependencies in the data. Tables 3.6 and 3.7 show the results for Accuracy and Conditional Log-likelihood. *MBN achieved substantially better predictions on all test sets, at least 20% for accuracy.*

Conditional Log-likelihood	Alchemy Methods				LAJ
Dataset	**MSL**	**LHL**	**BUSL**	**LSM**	**MBN**
Movielens11	-4.22	-4.60	-2.80	-1.21	**-1.15**
Movielens12	-3.55	-3.38	NT	**-1.06**	-1.33
Movielens	NT	NT	NT	-1.1	**-0.98**
Mutagenesis1	-4.45	-4.33	-2.54	**-1.12**	-1.85
Mutagenesis2	NT	NT	NT	**-1.18**	-1.48
Hepatitis	NT	NT	NT	-1.26	**-1.18**
Hepatitis1	NT	NT	NT	-1.34	**-1.09**

Table 3.5: The table compares conditional log likelihood performance of the moralization approach (MBN) vs. previous Markov Logic Network learning methods. The data are obtained by training on 2/3 of the database and testing on the other 1/3. CLL is reported as an average over all attribute predicates of the datasets.

Accuracy	Alchemy Methods				LAJ
Dataset	**MSL**	**LHL**	**BUSL**	**LSM**	**MBN**
University	0.37	0.37	0.38	0.40	**0.84**
Movielens11	0.43	0.42	0.36	0.39	**0.69**
Movielens12	0.42	0.48	NT	0.53	**0.68**
Movielens	NT	NT	NT	0.34	**0.74**
Mutagenesis1	0.36	0.33	0.37	0.33	**0.80**
Mutagenesis2	NT	NT	NT	0.31	**0.65**
Hepatitis	NT	NT	NT	0.33	**0.57**
Hepatitis1	NT	NT	NT	0.30	**0.57**

Table 3.6: The table compares accuracy performance of the moralization approach (MBN) vs. previous Markov Logic Network learning methods. The data report the training error where inference is performed over the training dataset. ACC is reported as an average over all attribute predicates of the datasets.

Conditional Log-likelihood	Alchemy Methods				LAJ
Dataset	**MSL**	**LHL**	**BUSL**	**LSM**	**MBN**
University	-5.79	-5.91	-2.92	-0.82	**-0.47**
Movielens11	-4.09	-4.09	-2.44	-1.18	**-1.06**
Movielens12	-3.55	-3.38	NT	-1.10	**-1.09**
Movielens	NT	NT	NT	-1.02	**-0.6**
Mutagenesis1	-4.33	-4.33	-2.54	-1.17	**-0.62**
Mutagenesis2	NT	-4.65	NT	-1.15	**-0.7**
Hepatitis	NT	NT	NT	-1.22	**-1**
Hepatitis1	NT	NT	NT	-1.28	**-1.03**

Table 3.7: The table compares conditional log-likelihood performance of the moralization approach (MBN) vs. previous Markov Logic Network learning methods. The data report the training error where inference is performed over the training dataset. CLL is reported as an average over all attribute predicates of the datasets.

3.6.4 UW-CSE Dataset

The UW-CSE dataset is naturally divided into 5 folds according to the subarea of computer science, so learning studies have used a cross-validation approach [59, 70], which we follow. This dataset has been used extensively in previous Markov Logic Network experiments, and it differs from the others in that it features a relatively small set of 4 attributes relative to the set of 5 relationships. We therefore provide a more detailed set of measurements that compare predictive performance for each attribute separately. As with the other datasets, the speed and predictive accuracy of the learn-and-join algorithm is a substantive improvement. The breakdown by attribute in Figures 3.6 and 3.7 shows that while the extent of the improvement varies with the predicates, the moralized Bayes net approach performs uniformly well on all predicates.

Table 3.8: Cross-validation averages for the UW-CSE dataset

	MSL	**LHL**	**LSM**	**BUSL**	**MBN**
Time(min)	2160	2413	2	276	**0.6**
Accuracy	0.29	0.25	0.26	0.27	**0.41**
Conditional log-likelihood	-3.85	-3.98	**-1.37**	-1.42	-1.43

Our results indicate that the two most recent Markov Logic Network structure learning methods—-Lifted Hypergraph Learning and Learning Structural Motifs—show the best performance. This is confirmed in independent empirical studies by other researchers [60].

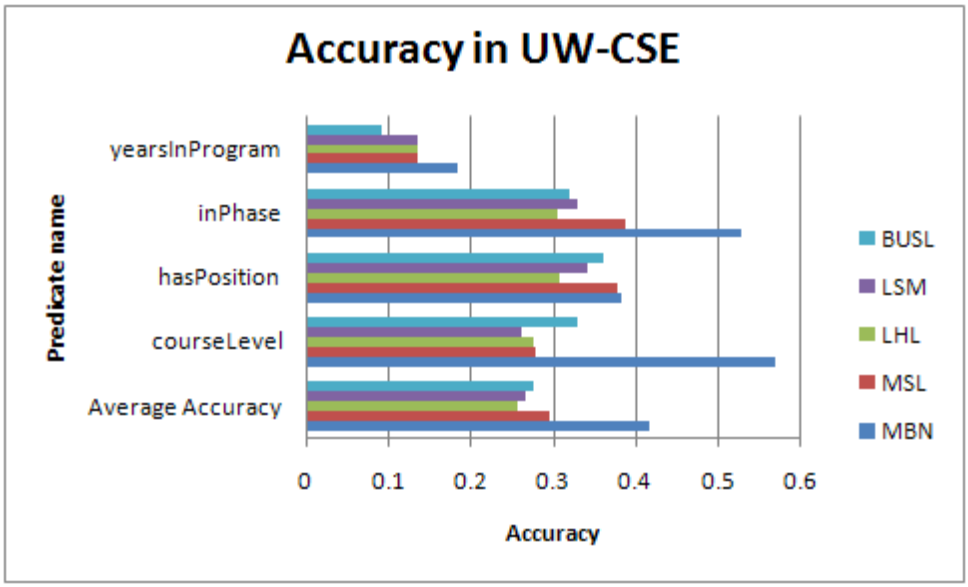


Figure 3.6: Predictive Accuracy by attribute, measured by 5-fold cross-validation. The methods are ordered as shown, with MBN at the bottom.

The remaining sections of the chapter therefore focuses on comparing LHL and LSM with the moralization approach.

3.6.5 Comparison with Inductive Logic Programming on Mutagenesis

We compare the performance of the learn-and-join algorithm for a classification task, predicting the mutagenicity of chemical compounds. This problem has been extensively studied in Inductive Logic Programming (ILP). The purpose of this comparison is to benchmark the predictive performance of the moralization approach against discriminative learning by methods that are different from Markov Logic Network learners.

The class attribute is the mutagenicity (log p). Compounds recorded as having positive mutagenicity are labeled active (positive examples) and compounds recorded as having 0 or negative mutagenicity are labeled inactive (negative examples). The database contains a total of 188 compounds. Whereas Inductive Logic Programming methods are discriminative, the moralization method performs generative learning over all attributes, a significantly more difficult task. We compare the predictive accuracy of the Moralized Bayes net with

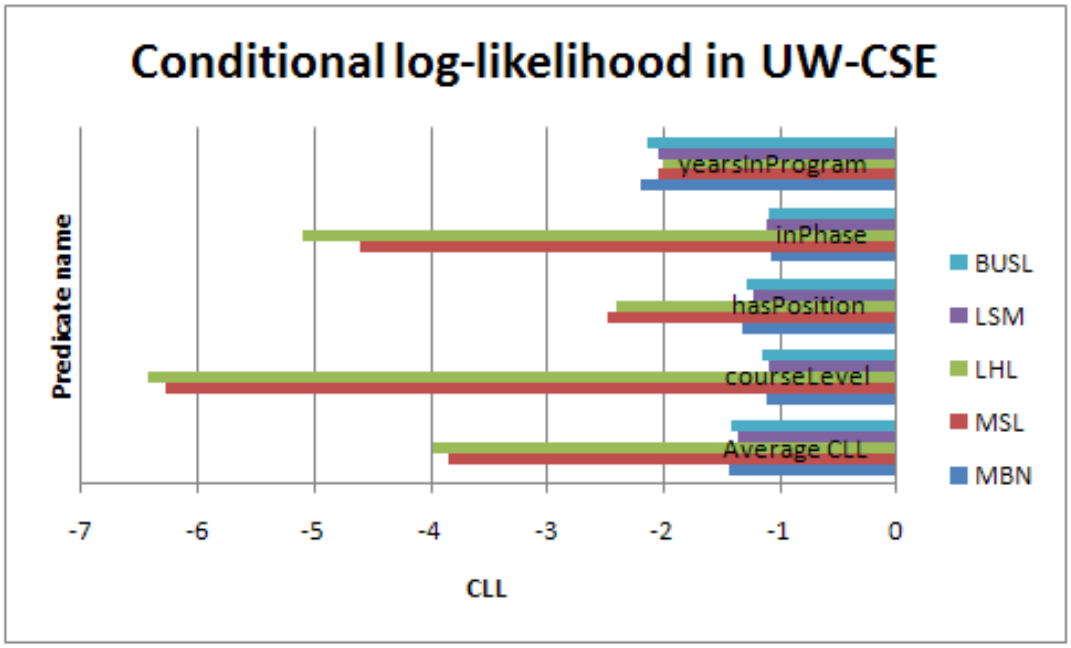


Figure 3.7: Conditional Log-likelihood by attribute, measured by 5-fold cross-validation. The methods are ordered as shown, with MBN at the bottom.

well-known ILP methods. Table 3.9 presents the results of Lodhi and Muggleton [66]. For the STILL system, we followed the creators' evaluation methodology of using a randomly chosen training and test set. The other systems are evaluated using 10-fold cross-validation. The table shows that the classification performance of the generative Moralized Bayes net model matches that of the discriminative Inductive Logic Programming models.

3.6.6 Lesion Studies-Lattice Constraints

In this section we study the effects of relaxing lattice constraints used in the learn-and-join algorithm. To achieve a high resolution, all results are based on 5-fold cross validation. We report a number of quantities for comparing the learned structures.

SLtime(s) Structure learning time in seconds

Numrules Number of clauses in the Markov Logic Network excluding rules with weight 0.

Method	Evaluation	Accuracy	Reference
MBN	10-fold	0.87	
P-progol	10-fold	0.88	[103]
FOIL	10-fold	0.867	[88]
STILL	90%train-10%test	0.936	[99]
MBN	90%train-10%test	0.944	

Table 3.9: A comparison of the Moralized Bayes net method with standard Inductive Logic Programming systems trained to predict mutagenicity. Although Bayes net learning produces a generative model, its performance is competitive with discriminative learners.

AvgLength The average number of atoms per clause.

AvgAbWt The average absolute weight value.

Because constraints curtail the search space, we expect the constrained method to have the following effects compared to the unconstrained method: (1) faster run-time, (2) a simpler model with fewer rules, and (3) If the constraints are chosen well, they should allow the system to identify important rules and not impair predictive performance.

A key feature of the learn-and-join algorithm is the use of lattice constraints. In terms of the join lattice, Bayes nets for join tables higher in the lattice inherit the edges from Bayes nets for join tables lower in the lattice. We remove this constraint to assess its effect on learning. If the Bayes nets learned for different join tables are no longer constrained to be consistent with each other, the question arises how to merge them into a single Bayes net. One possibility is to learn a Bayes net for the maximum join table (e.g., for the relationship set $\mathit{Registered}(S, C)$, $\mathit{Teaches}(P, C)$ in Figure 3.2). However, in experiments we found that for our benchmark datasets, the maximum join table is too small to yield meaningful results, because too few tuples meet the join conditions. We therefore investigated an intermediate approach where different Bayes nets are learned for single relationship tables joined with the associated entity tables (e.g., $\mathit{Registered}(S, C) \bowtie \mathit{Student} \bowtie \mathit{Course}$). In our benchmark datasets, there were very few conflicts between the edges in different Bayes nets, which we broke randomly. So we could obtain a single acyclic graph for the database by merging the graphs of the intermediate joins; we refer to this approach as the *intermediate* method. Figure 3.8 shows, that the algorithm can find long chains, but they are rare. There seems to be no important difference between the constrained and unconstrained methods with

	Constraint	Intermediate
SLtime(s)	**3.1**	3.3
# Rules	**289**	443
AvgAbWt	**2.08**	1.86
AvgLength	4.26	4.94
ACC	**0.86**	0.84
CLL	**-0.89**	-0.9

Table 3.10: Comparison to study the effects of removing Lattice Constraints on the University+ dataset.

	Constraint	Intermediate
SLtime(s)	**5.2**	6.1
# Rules	**818**	848
AvgAbWt	**1.68**	1.53
AvgLength	4.15	4.11
ACC	**0.47**	0.41
CLL	**-1.31**	-1.34

Table 3.11: Comparison to study the effects of removing Lattice Constraints on the Hepatitis dataset.

respect to chain length.

Tables 3.10, 3.11, and 3.12 shows the results for the University+, Hepatitis, and Mondial datasets. The numbers are averages from 5-fold cross validation. Folds are formed by randomly selecting entities as described in Section 3.5.1. The lattice constraints speed up the learning time spent on join tables, which is the dominant factor. The constrained model features fewer rules with comparatively higher weights. The average predictive accuracy was somewhat higher than with the unconstrained model, whereas the conditional log-likelihood performance was very similar. This is evidence that the constraints helped identify predictively relevant clauses.

	Constraint	Intermediate
SLtime(s)	**8.9**	13.2
# Rules	**739**	1053
AvgAbWt	**0.22**	0.24
AvgLength	3.98	4.66
ACC	**0.43**	0.37
CLL	**-1.39**	**-1.39**

Table 3.12: Comparison to study the effects of removing Lattice Constraints on Mondial dataset

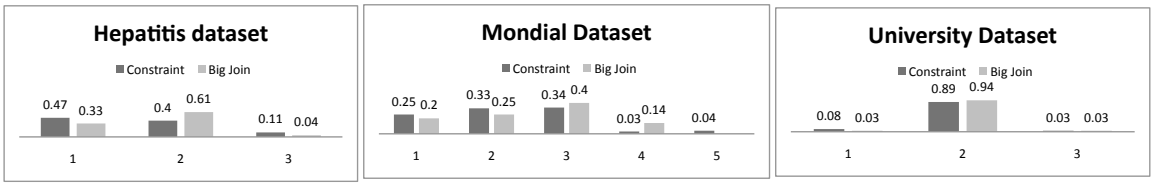


Figure 3.8: The figure illustrates the percentage of rules of a given chain length for Hepatitis, Mondial, and University dataset in the Lattice constraint lesion study.

Chapter 4

Learning Graphical Models for Relational Data with Recursive Dependencies

A key characteristic of relational data is that the value of a predicate often depends on values of the same predicate for related entities. For directed graphical models, such *recursive dependencies* lead to cycles, which violates the acyclicity constraint of Bayes nets. In this chapter we present a new approach to learning directed relational models which utilizes two key concepts: a pseudo likelihood measure that is well defined for recursive dependencies, and the notion of stratification from logic programming. We address three main problems in learning Bayes nets for relational data with recursive dependencies. The main focus of the chapter is that modelling recursive dependencies with Bayes nets introduces redundant edges that increase the complexity of learning. We propose a new normal form format that removes the redundancy, and prove that under mild assumptions, the normal form constraints involve no loss of modelling power. Empirical evaluation compares our approach to learning recursive dependencies with undirected models (Markov Logic Networks). The Bayes net approach is orders of magnitude faster, and learns more recursive dependencies, which lead to more accurate predictions. This work has been published in the journal of Machine learning [98].

In Section 4.1 we introduce the problems with modelling recursive dependencies with directed models and outline our approach. In Section 4.2 we illustrate a small database

with autocorrelation that is used throughout this chapter. Section 4.3 discusses related work on autocorrelation on directed models and stratification. Section 4.4 and 4.5 address the challenges to learning directed models with recursive rules and Section 4.6 extends the learn-and-join algorithm for recursive rules. Finally, in Section 4.7 we compare the model to other state-of-the art models.

4.1 Introduction

A key phenomenon that distinguishes relational data from single-population data is that the value of an attribute for an entity can be predicted by the value of the same attribute for related entities. This expressive power allows the model to elegantly represent *recursive dependencies* using Horn clauses. For example, whether individual a smokes may be predicted by the smoking habits of a's friends. This pattern can be represented by clausal notation such as $Smokes(X) \leftarrow Smokes(Y), Friend(X, Y)$.

Different subfields concerned with relational data have introduced different terms for this phenomenon. From a logic programming perspective, it is natural to speak of a *recursive dependency*, where a predicate depends on itself. In statistical-relational learning, Jensen and Neville introduced the term *relational autocorrelation* in analogy with temporal autocorrelation [43, 79]. In multi-relational data mining, such dependencies are found by considering *self-joins* where a table is joined to itself. We will use both the terms recursive dependency and autocorrelation. The former emphasizes the format of the rules we consider, whereas the latter distinguishes the probabilistic dependencies we model from deterministic logical entailment.

In this chapter we investigate a new approach to learning recursive dependencies with Bayes nets, specifically Poole's Parametrized Bayes Nets (PBNs) [84]; however, our results apply to other directed relational models as well, such as Probabilistic Relational Models (PRMs) [34] and Bayes Logic Programs (BLPs) [46]. We address three difficulties for learning recursive dependencies using directed models.

1. Recursive dependencies lead to cyclic dependencies among ground facts [91, 15, 105]. The cycles make it difficult to define a model likelihood function for observed ground facts in the data, which is an essential component of model selection techniques. To define a model likelihood function for Bayes net search, we utilize a recent relational Bayes net pseudo likelihood [93] that measures the fit of a PBN to a relational database

and is well-defined even in the presence of recursive dependencies. The learn-and-join algorithm discussed in Chapter 3 searches for models that maximize the pseudo likelihood. In this Chapter we evaluate the pseudo likelihood approach on datasets with strong autocorrelations.

2. A related problem is that defining valid probabilistic inferences in cyclic models is difficult. To avoid cycles in the ground model while doing inference, we propose converting learned Bayes nets to an undirected model using the standard moralization procedure. Inference with recursive dependencies can then be carried out using Markov Logic Networks (MLNs), a prominent relational model class that combines the syntax of logical clauses with the semantics of Markov random fields [15]. The moralization approach combines the efficiency and scalability of Bayes net learning with the high-quality inference procedures of MLNs.

3. A third problem which is the main focus of this chapter is the following; To show recursive dependencies using this pseudo likelihood, additional copies of variables participating in the recursive dependencies are added to the graphical model. For example a second variable to show the smoking habits of users are added to the graphical model. If each replicated predicate is teated as a separate random variable, then multiple redundant edges showing the same dependency will be added as the logical variables are interchangeable placeholders for the same domain of entities. In this chapter, we propose a normal form for Parametrized Bayes nets that eliminates such redundancies and prove that this constraint incurs no loss of expressive power.

We compared our learning algorithms with two state-of-the-art Markov Logic Network methods using public domain datasets The pseudo likelihood algorithm with main functor format is orders of magnitude faster, and learns more recursive dependencies, which lead to more accurate predictions.

4.2 Example

We illustrate Functor Bayes Nets and Markov Logic Networks with a relational schema that contains autocorrelation.

Figure 4.1 shows a simple database instance in the E-R format and its corresponding ground atoms in functor notation. As discussed in Section 2.1.6 on page 12, the functor

formalism is rich enough to represent the constraints of an entity-relationship (E-R) schema [107] via the following translation: Entity sets correspond to populations, descriptive attributes to functions, relationship tables to predicates, and foreign key constraints to type constraints on the arguments of relationship predicates.

People		
Name	Smokes	Cancer
Anna	T	T
Bob	T	F

Friend	
Name1	Name2
Anna	Bob
Bob	Anna

Smokes(Anna) =T	Friend(Anna, Bob) = T
Smokes(Bob) =T	Friend(Bob, Anna) = T
Cancer(Anna) = T	Friend(Anna, Anna) = F
Cancer(Bob) = F	Friend(Bob, Bob) = F

Figure 4.1: Left: A simple relational database instance. Right: The ground atoms for the database, and their values as specified by the database, using functor notation.

Figure 4.2 illustrates the Functor Bayes net and its grounding for the database in Figure 4.1. The double arrow corresponds to two directed edges which shows a cycle in the ground model. An example of a family formula with child node $Smokes(Y)$ is

$$Smokes(Y) = T, Smokes(X) = T, Friend(X, Y) = T.$$

4.3 Related Work

Bayes Net Learning for recursive dependencies. Adaptations of Bayes net learning methods for relational data are presented in [53, 22, 91, 26, 46]. Issues connected to learning Bayes nets with recursive dependencies are discussed in detail by Ramon *et al.* [91]. Early work on this topic required ground graphs to be acyclic [46, 26]. Probabilistic Relational Models allow dependencies that are cyclic at the predicate level as long as the user guarantees acyclicity at the ground level. For example whether a person has a gene depends on whether his/her parents have the gene. A recursive dependency of an attribute on itself is shown as a self loop in the model graph. If there is a natural ordering of the ground atoms in the domain (e.g., temporal), there may not be cycles in the ground graph. While this approach is sufficient for simple examples, it is restrictive for more general cases.

The generalized order-search of Ramon *et al.* instead resolves cycles by learning an ordering of ground atoms. A basic difference between our work and generalized order search

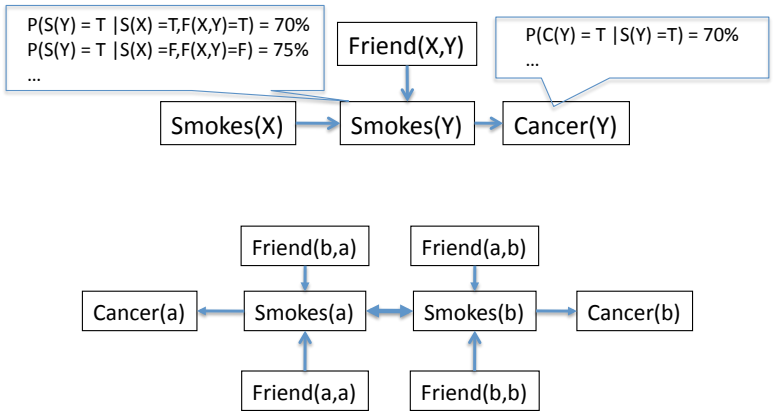


Figure 4.2: A Functor Bayes Net and its grounding for the database of Figure 4.1. The double arrow $\leftrightarrow$ is equivalent to two directed edges. Conditional probability parameters are chosen arbitrarily for illustration.

is that we focus on learning at the *predicate level*. Our algorithm can be combined with generalized order-search as follows: First use our algorithm to learn a Bayes net structure at the predicate/class level. Second carry out a search for a good ordering of the ground atoms. We leave integrating the two systems for future work.

Stratified Models. Stratification is a widely imposed condition on logic programs, because it increases the tractability of reasoning. Our definition is very similar to, but weaker than, the definition of stratification used by PRMs [Sec.4.1][30][29]. Related ordering constraints appear in the statistical-relational literature [19, 26].

4.4 Directed Models and Recursive Dependencies

Three main difficulties arise when directed models are used on relational data that contains autocorrelation. In this section we discuss the challenges on inference and model selection using examples and provide solutions for them. In the next chapter we discuss problems related to redundancy in directed models with recursive dependencies.

4.4.1 Inference in Directed Models with Recursive Dependencies

In statistical-relational learning, the usual approach to inference for relational data is to use the ground graphical model for defining a joint distribution over the attributes and links of entities. This approach is known as *knowledge-based model construction* [81, 62, 112] which faces the *cyclicity problem*: there may be cyclic dependencies between the properties of individual entities. For example, if there is generally a correlation between the smoking habits of friends, then we may have a situation where the smoking of Jane predicts the smoking of Jack, which predicts the smoking of Cecile, which predicts the smoking of Jane, where Jack, Jane, and Cecile are all friends with each other. In the presence of such cycles, neither aggregate functions nor combining rules lead to well-defined probabilistic predictions. Figure 4.2 shows a cycle of length 1 between the two gnodes $Smokes(a)$ and $Smokes(b)$. This model also illustrates how cycles arise in the presence of relationships that relate entities of the same type, as *Friend* relates two people. Such relationships are called rings in Entity-Relationship models [107] and are called **self-relationships** by Heckerman, Koller, Meek [39]. Self-relationships typically give rise to autocorrelations where the value of an attribute for an entity depends on the value of the same attribute among related entities. For instance, in the ground Bayes net of Figure 4.2, the value of $Smokes(a)$ depends on the value of *Smokes* for other people.

Because cycles are not allowed in a valid Bayes net graph, grounding Functor Bayes nets that include self-relationships does not lead to a valid distribution for carrying out probabilistic inference. The cyclicity problem has been difficult to solve, which has led Neville and Jensen to conclude that "the acyclicity constraints of directed models severely limit their applicability to relational data" [79, p.241]. Several researchers advocate the use of undirected models for relational data to avoid directed cycles. Markov random fields have therefore become a leading model class for learning and inference with relational data [15, 105].

Bayes net graphs can be converted to undirected Markov net graphs through the standard moralization method [15, 12.5.3]. Figure 4.3 illustrates the MLN structure obtained by moralization on Figure 4.2 and the corresponding ground Markov net for the database of Figure 4.1. For converting the Bayes net conditional probabilities to MLN clause weights, Domingos and Richardson suggest using the log of the conditional probabilities as the clause weight [15, 12.5.3], which is the standard conversion for propositional Bayes nets. Figure 4.3

illustrates moralization using log-probabilities as weights. In this chapter we apply moralization only to the model structure.

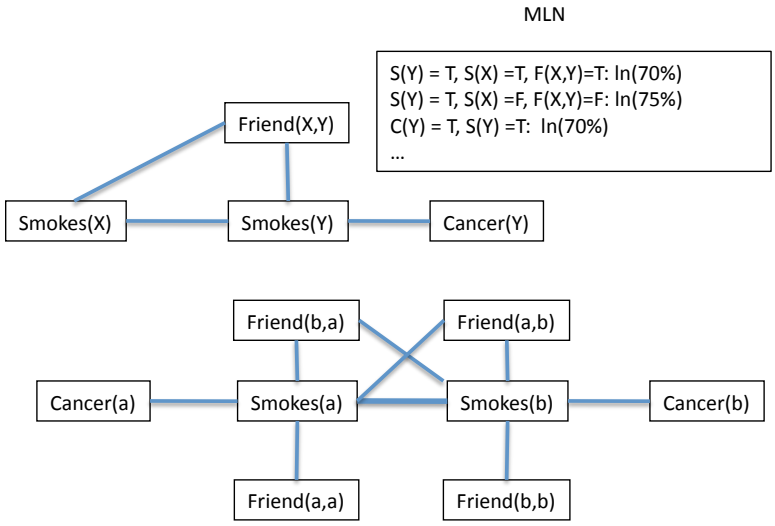


Figure 4.3: The moralized Functor Bayes net of Figure 4.2 and its ground Markov network for the database of Figure 4.1. The clauses are the family formulas in the Functor Bayes net. Each clause weight is the logarithm of the conditional probability that corresponds to the clause.

4.4.2 Model Selection in Directed Models with Recursive Dependencies

Score-based learning algorithms for Bayes nets require the specification of a numeric model selection score that measures how well a given Bayes net model fits observed data. A common approach to defining a score for a relational database is to consider the ground graph for a given database, illustrated in Figure 4.2. For example, suppose that the learning algorithm is in a step that it is considering adding the edge $Smokes(X) \rightarrow Smokes(Y)$, given a database like the one in Figure 4.1 which specifies a value for each node in the ground graph. Thus the likelihood of the new Parametrized Bayes net for the database can be defined as the likelihood assigned by the ground graph to the facts in the database following the usual Bayes net product formula. However, considering this edge introduces cycles in the ground model as illustrated in Figure 4.2.

Schulte [93] proposed a way to measure the fit of a Bayes net model to relational data

that does not require acylicity: the idea is to consider a *random* grounding of the 1st-order variables in the Parametrized Bayes net, rather than a complete grounding. The pseudo log-likelihood is defined as follows.

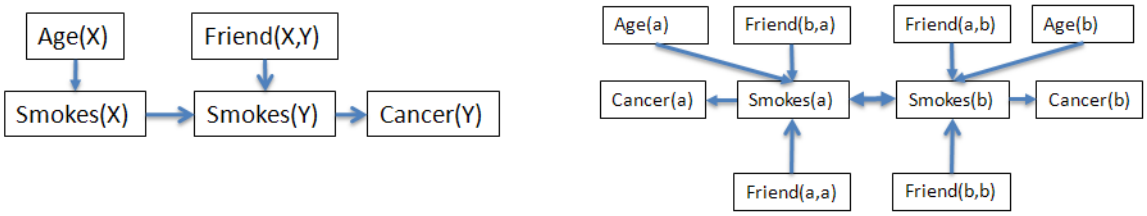
1. Let $X_1, \ldots, X_k$ be a list of *all* first-order variables that occur in the functor nodes of the parametrized Bayes net B.
2. Randomly select a grounding for *all* first-order variables that occur in the Bayes Net. The result is a ground graph with as many nodes as the original Bayes net.
3. Look up the value assigned to each ground node in the database. Compute the log-likelihood of this joint assignment using the usual product formula; this defines a log-likelihood for the random instantiation.
4. The expected value of this log-likelihood is the *pseudo log-likelihood* of the database given the Bayes net.

We utilize this pseudo likelihood measure in the learn-and-join algorithm to maximize the pseudo-likelihood [93].

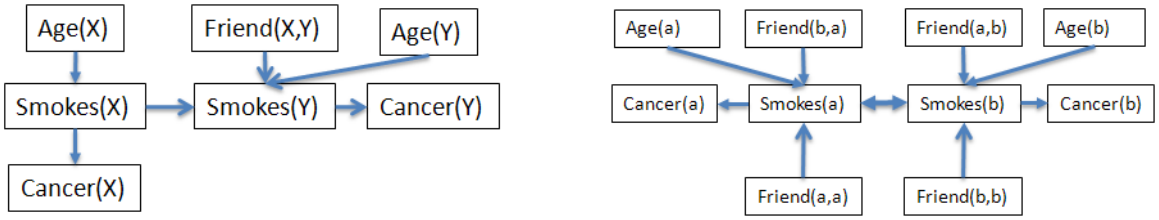
4.5 Redundancy in Directed Models with Recursive Dependencies

The repetition of predicates causes additional complexity in learning if each predicate instance is treated as a separate random variable. If there is in fact a statistical dependence of a repeated predicate and another predicate, then each of edges from the two predicates correctly represents this dependency, but one of them is redundant, as the logical variables are interchangeable placeholders for the same domain of entities. Consider the following example; Figure 4.4(a) shows a Bayes net without redundancy for the database given in Figure 4.1 with the addition of the age of the individuals to the database and the assumption that age of people is correlated with their smoking habits. If we treat $Smokes(X)$ and $Smokes(Y)$ as entirely separate variables, learning needs to consider additional edges similar to those already in the Bayes net, like $Smokes(X) \rightarrow Cancer(X)$ and $Age(Y) \rightarrow Smokes(Y)$. Figure 4.4(b) shows the Bayes net with these edges being added. However, such edges are redundant because the first-order variables X and Y are interchangeable as they refer to the same

entity set. In terms of ground instances, Figure 4.4 shows that both models have exactly the same ground instances.



(a) Left: A Bayes net without redundant edges. Right: The ground graph for two individuals a and b.



(b) Left: A Bayes net with redundant edges. Right: The ground graph for two individuals a and b.

Figure 4.4: An example to illustrate redundancy in directed models with recursive dependency. Note that both Bayes nets have the same ground model.
not

We propose a normal form for Parametrized Bayes nets that eliminates such redundancies using the notion of stratification.

4.5.1 Stratification and Recursive Dependencies

In this section we first consider analytically the relationship between cycles in a ground Bayes net and orderings of the functors that appear in the nonground Bayes net. It is common to characterize a Logic Program by the orderings of the functors that the logic program admits [65]; we adapt the ordering concepts for Bayes nets. The key ordering concept is the notion of a *level mapping*. We apply it to Bayes nets as follows.

Definition 1 *Let B be an Parametrized Bayes net. A **level mapping** assigns to each functor f in B a nonnegative integer level(f).*

- *A Bayes net is* ***strictly functor-stratified*** *if there is a level mapping such that for every edge $f(\tau) \rightarrow g(\tau)$, we have $level(f) < level(g)$.*
- *A Bayes net is* ***functor-stratified*** *if there is a level mapping such that for every edge $f(\tau) \rightarrow g(\tau)$, we have $level(f) \leq level(g)$.*

Left: A Bayes net with redundant edges. Right Strict stratification corresponds to the concept of a hierarchical rule set [65]. Since it implies that one fnode cannot be an ancestor of another fnode with the same functor, strict functor-stratification rules out recursive clauses. Functor-stratification with a weak inequality, by contrast, does allow the representation of autocorrelations. Functor stratification is a weaker condition than stratification in logic programs, because it requires only an ordering on predicates, and imposes no constraints on positive vs. negative literals. Stratification is a widely imposed condition on logic programs, because it increases the tractability of reasoning [65, Sec.3.5],[2]. The concept of stratification used by PRMs [Sec.4.1][30][29] pertains to predicates, like our definition, but makes a stronger requirement that guarantees that the ground graph is acyclic. Functor-stratification as we define it does not rule out cycles in the ground graph.

We next show that strict functor-stratification characterizes the absence of cycles in a ground Bayes net.[1]

Proposition 1 *Let B be a Parametrized Bayes net, and let $\mathcal{D}$ be a database instance such that every population (entity type) has at least two members. Then the ground graph $\overline{B}$ for $\mathcal{D}$ is acyclic if and only if the Bayes net B is strictly functor-stratified.*

Proof Outline. ($\Leftarrow$) If B is strictly functor-stratified, then so is the ground graph $\overline{B}$, using the same level mapping. Since each child node is ranked less than its parent, there can be no cycle in $\overline{B}$.

($\Rightarrow$) Suppose that B is not strictly functor-stratified. Then there are distinct fnodes $f(\boldsymbol{\tau}), f(\boldsymbol{\tau}')$ for the same functor such that $f(\boldsymbol{\tau})$ is an ancestor of $f(\boldsymbol{\tau}')$ in B. Since they are distinct fnodes, they disagree on at least one variable argument. Without loss of generality, let $f(\boldsymbol{\tau}) = f(X, \cdot)$ and $f(\boldsymbol{\tau}') = f(Y, \cdot)$, where $X \neq Y$.

[1] This result assumes that no functor node contains the same variable twice. This assumption does not involve a loss of modelling power because a functor node with a repeated variable can be rewritten using a new functor symbol (provided the functor node contains at least one variable). For instance, a functor node $Friend(X, X)$ can be replaced by the unary functor symbol $Friend_{self}(X)$.

Pick any two distinct members a, b of the common population associated X, Y. First instantiate $f(X, \cdot)$ as a gnode $f(a, \cdot)$ and $f(Y, \cdot)$ as $f(b, \cdot)$. Then the ground graph $\overline{B}$ contains a directed path

$$f(a, \cdot) \rightarrow \cdots \rightarrow f(b, \cdot).$$

Second, instantiate $f(X, \cdot)$ as $f(b, \cdot)$ and $f(Y, \cdot)$ as $f(a, \cdot)$. Then the ground graph $\overline{B}$ contains a directed path

$$f(b, \cdot) \rightarrow \cdots \rightarrow f(a, \cdot).$$

Therefore the ground graph contains a directed cycle from $f(a, \cdot)$ to $f(b, \cdot)$ and back again, which establishes the claim.

This result shows that cyclic dependencies arise precisely when a node associated with one functor is an ancestor of another node associated with the same functor.[2] This in turn is exactly the graphical condition associated with recursive dependencies, which means that recursive dependencies and cyclic dependencies are closely connected phenomena.

While functor-stratified Bayes nets have the expressive power to represent autocorrelations, there is potential for additional complexity in learning if each functor is treated as a separate random variables. We discuss this issue in the next subsection and propose a normal form constraint for resolving it.

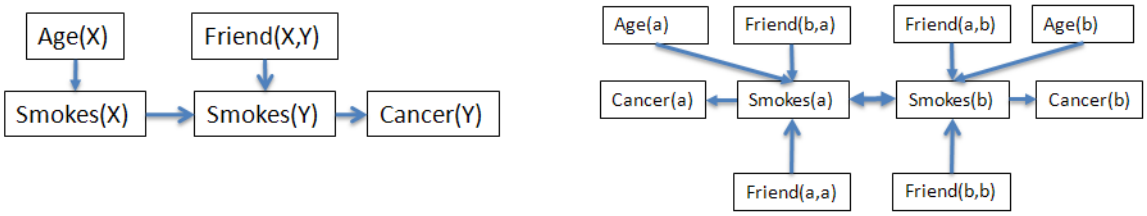
4.5.2 Stratification and the Main Functor Node Format

Redundant edges can be avoided if we restrict the model class to the main functor format, where for each function symbol f, there is a *main functor node* $f(\boldsymbol{\tau})$ such that all other functor nodes $f(\boldsymbol{\tau}')$ associated with the same functor are sources in the graph, that is, they have no parents. The intuition for this restriction is that statistically, two functors with the same function symbol are equivalent, so it suffices to model the distribution of these functors conditional on a set of parents just once. This leads to the following definition.

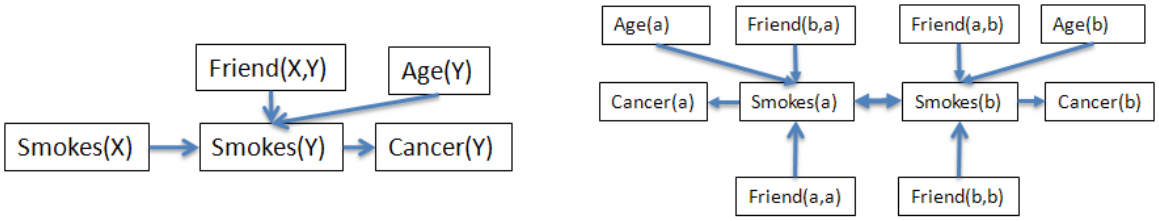
Definition 2 *A Bayes net B is in **main functor node form** if for every functor f of B, there is a distinguished functor node $f(\boldsymbol{\tau})$, called the **main functor node** for f, such that every other functor node $f(\boldsymbol{\tau}')$, where $\boldsymbol{\tau}' \neq \boldsymbol{\tau}$, has no parents in B.*

[2] In some statistical-relational models such as PRMs, the ground graph is constructed somewhat using the known relational context to add fewer edges [26]. In that case strict stratification remains sufficient for acyclicity but may no longer be necessary;

Example. The Bayes net of Figure 4.5(a) does not have any redundant edges, however it is not in main functor form because we have two functor nodes for *Smokes* with nonzero indegree. The Bayes net in Figure 4.5(b) is in main variable format where Smokes(Y) is the main functor for Smokes(X). In terms of ground instances, the two Bayes nets have exactly the same ground graph.



(a) Left: This Bayes net is not in main functor form. Right: The ground graph for two individuals a and b.



(b) Left: This Bayes net is in main functor form. Right: The ground graph for two individuals a and b.

Figure 4.5: An example to illustrate the main functor format. Note that the ground model for both figures is the same.

The next proposition shows that this equivalence holds in general: For any Bayes net B there is an equivalent Bayes net B' in main functor node form. This claim is established constructively by showing how the original B can be transformed into B'. The transformation procedure is a conceptual aid, rather than an algorithm to be used in practice; to build a practical learning algorithm, we simply restrict the Bayes net candidates to be in main functor form (see Section 4.6 below).

It is easy to see that we can make *local* changes to the 1st-order variables such that all child nodes for a given functor are the same. For instance, in the Bayes net of Figure 4.5(a) we can first substitute Y for X to change the edge $Age(X) \rightarrow Smokes(X)$ into the edge $Age(Y) \rightarrow Smokes(Y)$. Then we delete the former edge and add the latter, that is, we

make $Age(Y)$ a parent of $Smokes(Y)$. Figures 4.5(b) and 4.5(a) illustrate that the original and transformed Bayes nets have the same ground graph. However, in general the change of variables may introduce cycles in the Bayes net. The basis for the next proposition is that if the original Bayes net is functor-stratified, the transformed functor node graph is guaranteed not to contain cycles.

Proposition 2 *Let B be a functor-stratified Bayes net. Then there is a Bayes net B' in main functor form such that for every database $\mathcal{D}$, the ground graph $\overline{B}$ is the same as the ground graph $\overline{B'}$.*

Proof of Proposition. Let B be a functor-stratified Bayes net. Consider the first function symbol f at level 0. Enumerate its associated functors as $f(\tau_1), \ldots, f(\tau_k)$, such that for every i, j, if $i < j$, then $f(\tau_i)$ is not a descendant of $f(\tau_j)$ in B. This is possible since B is acyclic. For every edge $g(\sigma) \rightarrow f(\tau_j)$, where $j < k$, change the variables in σ to obtain a term σ_j such that the edge $g(\sigma) \rightarrow f(\tau_j)$ has exactly the same instantiations as the edge $g(\sigma_j) \rightarrow f(\tau_k)$. This is possible because the functors contain neither constants nor repeated variables.

Finally, add all edges of the form $g(\sigma_j) \rightarrow f(\tau_k)$ to B and eliminate all edges into $f(\tau_j)$, for $j < k$. The resulting graph B_0 has the same ground graph as B. It is in main functor format wrt f since $f(\tau_k)$ is the only functor with function symbol f that may have parents. To see that B_0 is acyclic, note that by stratification $f = g$, so all new edges are from functors $f(\tau_j)$ to $f(\tau_k)$. So a cycle in B_0 implies that $f(\tau_k)$ is an ancestor of $f(\tau_j)$ in B, for $j < k$, which is a contradiction.

We now repeat the construction for level 1, 2, etc. The resulting graphs $B_1, B_2, \ldots$ are acyclic because when an edge $g(\sigma_j) \rightarrow f(\tau_k)$ is added, either g is at a lower level than f, or $g = f$, therefore $g(\sigma_j)$ is not an ancestor of $f(\tau_k)$. After completing the construction for the highest stratum, we obtain a graph B' in main functor form whose grounding is the same as that of B, for any database.

4.5.3 Discussion.

While the transformation algorithm produces Bayes nets with the same groundings, at the variable or class level the two models may not be equivalent. For instance, the model of Figure 4.5(a) implies that $Age(X)$ is independent of $Friend(X, Y)$ given $Smokes(X)$. But in the model of Figure 4.5(b), the node $Age(Y)$ is dependent on (d-connected with)

$Friend(X, Y)$ given $Smokes(Y)$. The transformed model represents more of the dependencies in the ground graph. For instance, the gnodes $Age(a)$ and $Friend(b, a)$ are both parents of the gnode $Smokes(a)$, and hence d-connected given $Smokes(a)$.

In general, the transformed main functor Bayes net features more dependencies and nodes with more parents than the original one. If the dependencies do not exist in the data, the independences are not captured in the Bayes net graph, but can be represented in the conditional probability table, or using a more flexible representation. Alternatively, the PBN structure can be annotated so that the moralization contains shorter family formulas (e.g., $Smokes(Y), Age(Y)$). Another option is to use a model that combines clauses with Bayes nets.

For instance, in a Bayes Logic Program [46], we may initially have two Bayesian clauses

$$Smokes(X) \leftarrow Age(X)$$

and

$$Smokes(Y) \leftarrow Smokes(X), Friend(X, Y).$$

In a Parametrized Bayes Net, the two clauses are effectively merged into a single clause

$$Smokes(Y) \leftarrow Age(Y), Smokes(X), Friend(X, Y).$$

Fundamentally, the merging occurs because the graphical format does not distinguish different sets of parents, not because of the main functor node form.

4.6 The Learn-and-Join Structure Algorithm With Recursive Dependencies

The main learn-and-join algorithm was discussed in Section 3.4 and some constraints were introduced for the algorithm. In this section we introduce a constraint related to the main functor form. The algorithm requires the specification of a main functor node for each functor (e.g., $Smokes(Y)$ is the main functor node for the functor *Smokes*). Only main functor nodes are allowed to have edges pointing into them (i.e., indegree greater than 0). The intuition behind this constraint is that it suffices to model the conditional distribution of just one "copy" of the functor. For example, to model the conditional distribution of the *Smokes* attribute, it suffices to have parents only for the functor node $Smokes(Y)$, rather

than allow parents for both the functor node $Smokes(Y)$ and $Smokes(X)$. One way to define main functor nodes is in terms of a lexicographic ordering derived from an ordering of population variables (cf. Section 3.3.2). That is, the main node for function symbol f is the functor $f(\tau)$ where τ is the lexicographically first list of terms for f. We refer to such Functor Bayes nets as **main variable** Functor Bayes nets.

Constraint 5 *For any Bayes net associated with a relationship set, if its graph contains an edge* $\rightarrow f(\boldsymbol{\tau})$ *pointing into a node* $f(\boldsymbol{\tau})$*, then* $f(\boldsymbol{\tau})$ *is the main functor node for* f*.*

Implementing the main functor format simply requires adding all edges to the forbidden edge cache that do not point to main functor nodes. Thus the main functor format provides constraints that reduce the complexity of learning. Algorithm 2 provides pseudocode for the case of a single self-relationship R. The presentation for the single-relation case is simpler than for the multi-relational case and highlights the differences with the previous version of the learn-and-join algorithm. Extending the algorithm to the multi-relational case can be done using the lattice search framework.

4.6.1 Example of Algorithm.

We consider a the self-relationship *Friend* defined on the *People* entity set. Figure 4.6 visually illustrates the construction.

1. Applying the single-table Bayes net learner to the *People* table may produce a single-edge graph $Smokes(Y) \rightarrow Cancer(Y)$. (Line 5)

2. Then form the join data table

$$J = Friend \bowtie People \bowtie People$$

   (Line 6), where the join should be read as theta-join. The Bayes net learner is applied to J, with the following constraints.

   (a) From the *People* Bayes net, there must be an edge $Smokes(Y) \rightarrow Cancer(Y)$, since $Cancer(Y)$.

   (b) No edges may point into $Smokes(X)$ or $Cancer(X)$, since these are not the main functor nodes for the functors *Smokes* and *Cancer* (Line 8).

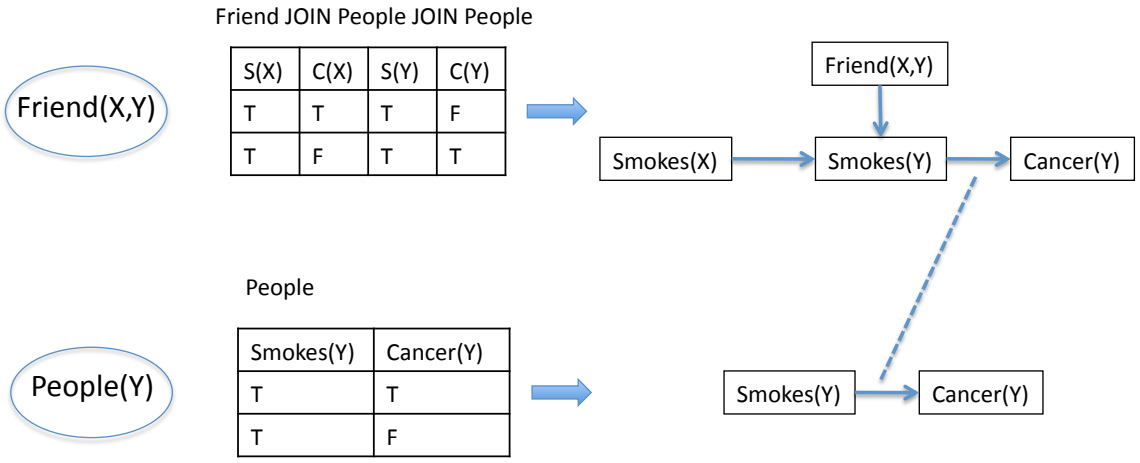


Figure 4.6: The 2-net lattice associated with the DB instance of Figure 4.1. The figure shows the data tables associated with the only entity table *People* and the only relationship table *Friend*. The block arrow indicates that the output of a single-table Bayes net learner on the data table is the Bayes net shown. The dashed line that connects the two edges $Smokes(Y) \rightarrow Cancer(Y)$ indicates that this edge is propagated from the lower-level Bayes net to the higher-level Bayes net.

Algorithm 2: Pseudocode for structure learning (Single Self-Relationship)

Input: Database $\mathcal{D}$ with self-relationship R on entity table E.
Output: PBN graph G for $\mathcal{D}$
Calls: PBN: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.
Notation: PBN(T, Econstraints) denotes the output DAG of PBN. Get-Constraints(G) specifies a new set of edge constraints, namely that all edges in G are required, and edges missing between variables in G are forbidden.

1: Add descriptive attributes of E and R to G. {These are the main functor nodes for the attributes.}
2: Add a duplicate node for each descriptive attribute of E to G. {The duplicates are auxiliary nodes, not main functor nodes.}
3: Add a Boolean indicator B_R for relationship table R to G.
4: Econstraints = $\emptyset$ {Required and Forbidden edges}
5: Econstraints += Get-Constraints(PBN(E , $\emptyset$)).
6: J := join of R, E, E.
7: MainFunctorConstraints := **forbid edges into attribute nodes of E that are not main functor nodes.**
8: Econstraints += MainFunctorConstraints.
9: Econstraints += Get-Constraints(PBN(J, Econstraints).
10: G := Set of all required edges from Econstraints.
11: If there is an edge $u \rightarrow v$ from an auxiliary node to a main functor node, add an edge $B_R \rightarrow v$ to G.
12: Return G.

The Bayes net learner applied to the join table J then may find an edge $Smokes(X) \rightarrow Smokes(Y)$ (Line 9). Since the dependency represented by this edge is valid only for pairs of people that are friends (i.e., conditional on $Friend(X, Y) = T$), the algorithm adds an edge $Friend(X, Y) \rightarrow Smokes(Y)$ (Line 11). In this example, functor node $Cancer(X)$ is disconnected, so the figure does not show it.

Discussion. The learn-and-join algorithm finds a structure that maximizes the pseudo-likelihood described in Section 4.4.2 [93]. Khosravi *et al.* discuss the time complexity of the basic learn-and-join algorithm and show that the edge-inheritance constraint essentially keeps the model search space size constant even as the number of nodes considered grows with larger table joins. For the learn-and-join algorithm, the main computational challenge in scaling to larger table joins is therefore not the increasing number of columns (attributes) in the join, but only the increasing number of rows (tuples). The main functor constraint

contributes further to decreasing the search space. For instance, suppose that we have k duplicate nodes and n nodes in total. Then for each duplicate node, there are $2(n-1)$ possible directed adjacencies. The main functor constraint eliminates a possible direction for adjacencies involving duplicate nodes, hence removes $k(n-1)$ directed adjacencies from consideration.

4.7 Evaluation

All experiments were done on a QUAD CPU Q6700 with a 2.66GHz CPU and 8GB of RAM. Our code and datasets are available on the world-wide web [48]. We made use of the following implementations as in 3.

Single Table Bayes Net Search GES search [11] with the BDeu score as implemented in version 4.3.9-0 of CMU's Tetrad package (structure prior uniform, ESS=10; [106]).

MLN Parameter Learning The default weight training procedure [67] of the Alchemy package [61], Version 30.

MLN Inference The MC-SAT inference algorithm [85] to compute a probability estimate for each possible value of a descriptive attribute for a given object or tuple of objects.

Algorithms. We compared three structure learning algorithms.

MBN The structure is learned using the extended learn-and-join algorithm. The weights of clauses are learned using Alchemy. This method is called MBN for "moralized Bayes Net" by Khosravi *et al.* [53].

LHL Lifted Hypergraph Learning [59] uses relational path finding to induce a more compact representation of data, in the form of a hypergraph over clusters of constants. Clauses represent associations among the clusters.

LSM Learning Structural Motifs [60] uses random walks to identify densely connected objects in data, and groups them and their associated relations into a motif.

We chose LSM and LHL because they are the most recent MLN structure learning methods.

4.7.1 Datasets

University+ Database. To test learning algorithms with autocorrelations, we extend the manually created small dataset of the schema given in Table 2.1 with a self-relationship *Friend* between Students such that the ranking and coffee drinking habits of a student are strongly correlated with the ranking and coffee drinking habits of her friends, respectively. The dataset has about 1000 tuples.

Mondial Database. This dataset contains data from multiple geographical web data sources [69]. We follow the modification of [100], and use a subset of the tables and features. Our dataset includes a self-relationship table *Borders* that relates two countries.

Figure 4.7 shows the learned Bayes nets for the University+ and Mondial datasets. The graphs illustrate how the extended learn-and-join algorithm learns models with complex dependency patterns with recursive dependencies.

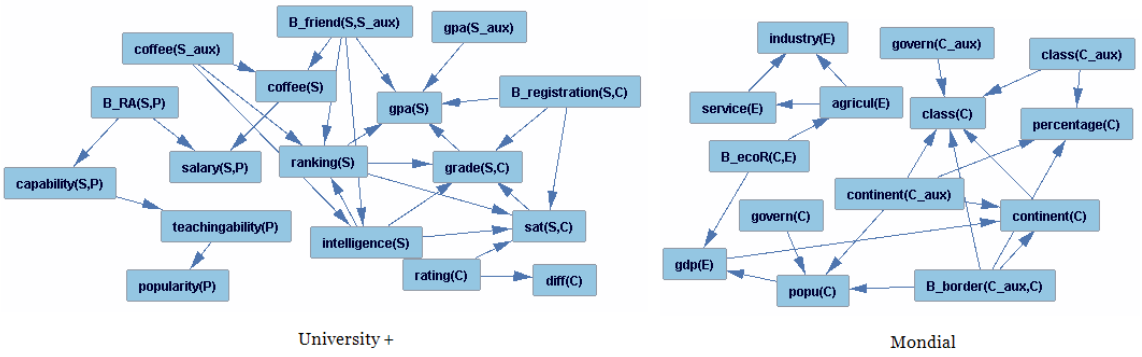


Figure 4.7: The Bayes net structures learned by the learn-and-join algorithm for University+ and Mondial datasets. The aux files are used to learn recursive rules on the main functors.

4.7.2 Lesion Studies-Main Functor Constraints

In this section we study the effects of relaxing the main functor constraint for learning autocorrelations, the other constraints simply reflect the semantics of the relational schema rather than learning principles. To achieve a high resolution, all results are based on 5-fold cross validation. We report a number of quantities for comparing the learned structures as in Chapter 3.

SLtime(s) Structure learning time in seconds

Numrules Number of clauses in the Markov Logic Network excluding rules with weight 0.

AvgLength The average number of atoms per clause.

AvgAbWt The average absolute weight value.

We remove the constraints of specifying a main functor node to study its importance. This means that we introduce a copy of an entity table that is potentially involved in an autocorrelation (e.g., in the University+ database, there are two tables $Student_1$, $Student_2$). This duplication approach is used other relational learning systems (e.g., [115, 114]). The unconstrained method applies the learn-and-join algorithm in the same way to all entity tables, including the copies. We investigated the following main hypotheses about the effect of the main functor constraint.

1. There should be a tendency towards longer clauses associated with the main functor node (see Section 4.5.3).
2. Since Proposition 2 implies that the ground models with and without the constraint are the same, predictive accuracy should be similar.
3. The duplicate edges should lead to duplicate clauses without improvement in predictive performance since the ground models are the same.

Table 4.1 shows the results for University and Table 4.2 shows the results for Mondial dataset. *Constraint* is the learn-and-join algorithm with the main functor constraint, whereas *Duplicate* is the learn-and-join algorithm applied naively to the duplicate tables without the constraint. As expected, the constraint speeds up structure learning, appreciably in the case of the larger Mondial dataset. The number of clauses is significantly less (50-60), while on average clauses are longer. The size of the weights indicates that the main functor constraint focuses the algorithm on the important rules.

Figure 4.8 reports the number of clauses of a given chain length. Since a clause contains conditions on both attributes and links, we use the maximal slot chain length: The chain length of a rule is computed as the maximal length of a sequence of predicates appearing in the rule such that the database tables corresponding to the predicates are related by foreign key pointers [30]. The measurements show that potentially the algorithm can find long chains, although informative long chains are rare.

	Constraint	Duplicate
SLtime(s)	**3.1**	3.2
# Rules	**289**	350
AvgLength	4.26	4.11
AvgAbWt	**2.08**	1.86
ACC	**0.86**	**0.86**
CLL	**-0.89**	**-0.89**

Table 4.1: Comparison to study the effects of removing Main Functor Constraints on University+ dataset.

	Constraint	Duplicate
SLtime(s)	**8.9**	13.1
# Rules	**739**	798
AvgLength	3.98	3.8
AvgAbWt	0.22	**0.23**
ACC	**0.43**	**0.43**
CLL	**-1.39**	**-1.39**

Table 4.2: Comparison to study the effects of removing Main Functor Constraints on Mondial dataset.

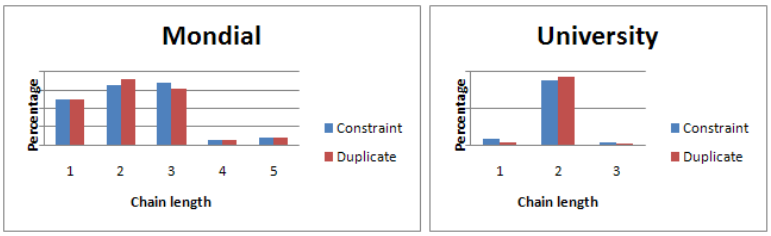


Figure 4.8: The figure illustrates the percentage of rules of a given chain length for Mondial and University+ dataset in the autocorrrelation lesion study.

	MBN	LSM	LHL
Time (seconds)	12	**1**	2941
Accuracy	**0.85**	0.44	0.47
CLL	**-0.8**	-2.21	-4.68

Table 4.3: Results on synthetic data.

	MBN	LSM	LHL
Time (seconds)	50	**2**	15323
Accuracy	**0.50**	0.26	26
CLL	**-1.05**	-1.43	-3.69

Table 4.4: Results on Mondial.

4.7.3 Predictive Accuracy and Data Fit.

Performance Metrics. We use 3 performance metrics: Runtime, Accuracy (ACC), and Conditional log likelihood (CLL). ACC and CLL have been used in previous studies of MLN learning [70, 59]. The CLL of a ground atom in a database given an MLN is its log-probability given the MLN and the information in the database. Accuracy is evaluated using the most likely value for a ground atom. For ACC and CLL the values we report are averages over all attribute predicates. We evaluate the learning methods using 5-fold cross-validation as follows. We formed 5 subdatabases for each by randomly selecting entities from each entity table and restricting the relationship tuples in each subdatabase to those that involve only the selected entities [53]. The models were trained on 4 of the 5 subdatabases, then tested on the remaining fold. We report the average over the 5 runs, one for each fold.

Table 4.3 and 4.4 summarize the results for the University+ and Mondial datasets respectively. *Neither of the Markov Logic methods LHL nor LSM discovered any recursive dependencies.* In contrast, the learn-and-join algorithm discovered the dependencies displayed in Table 4.9 using clausal notation. The dependency

$$\mathit{religion}(X) \leftarrow \mathit{continent}(X), \mathit{Border}(X, Y), \mathit{religion}(Y)$$

is a real-world example of the merging phenomenon discussed in Section 4.5.3. The learn-and-join algorithm analyzes the country table to find the dependency $\mathit{religion}(X) \leftarrow \mathit{continent}(X)$. Intuitively, the continent of a country is associated with its religion. It then joins the Country table with the Borders relationship table to find the recursive dependency $\mathit{religion}(X) \leftarrow \mathit{Border}(X, Y), \mathit{religion}(Y)$. Intuitively, the religion of a country is correlated with the religion of its neighbors. As required by the Bayes net format, the two dependencies are merged to form a single set of parents $\mathit{continent}(X), \mathit{Border}(X, Y), \mathit{religion}(Y)$.

The predictive accuracy using MLN inference was much better in the moralized model (average accuracy improved by 25% or more). This indicates that the discovered recursive dependencies are important for improving predictions.

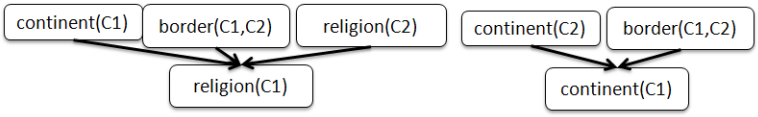


Figure 4.9: Dependencies discovered by the autocorrelation extension of the learn-and-join algorithm.

Both MBN and LSM are fast. The speed of LSM is due to the fact that its rules are mostly just the unit clauses that model marginal probabilities (e.g., $\mathit{intelligence}(S, \mathit{1})$).

Chapter 5

Learning Compact Markov Logic Networks With Decision Trees

In the previous chapters, we introduced the moralization approach: learn a set of directed Horn clauses, then convert them to conjunctions to obtain MLN clauses. The directed clauses are learned using Bayes net methods. The moralization approach takes advantage of the high-quality inference algorithms for MLNs and their ability to handle cyclic dependencies. A weakness of the moralization approach is that it leads to an unnecessarily large number of clauses. The Bayes net method learn dependencies among predicates, not literals, which fail to capture local or context-sensitive independencies. As MLNs have one weight parameter for each clause, this decreases the accuracy of parameter estimates, and slows down inference. In this chapter we show that using decision trees to represent conditional probabilities in the Bayes net is an effective remedy that leads to much more compact MLN structures. The decision trees can be learned using standard propositional decision tree learners. In experiments on benchmark datasets, the decision trees reduce the number of clauses in the moralized MLN by a factor of 5-25, depending on the dataset. The accuracy of predictions is competitive with the unpruned model and in many cases superior. This work has been published in the Journal of Machine Learning [52].

Chapter Organization. We review related work in Section 5.2, then go through an example in Section 5.3. We cover Augmenting Bayes nets with decision trees in Section 5.4 and show how the learn-and-join moralization algorithm can be combined with probability estimation trees in Section 5.5. We explain the experimental design in Section 5.6 and

preform through evaluation on our method in Section 5.7.

5.1 Introduction: Context-Sensitive Moralization

The moralization approach, introduced in Chapter 3 can be seen as a hybrid method that uses directed models for learning and undirected models for inference. This method learns a directed first-order Bayes net model for an input relational database. The Bayes net structure is then converted to an MLN set of clauses using the moralization method, described by Domingos and Richardson [15, 12.5.3]. In graphical terms, moralization connects all co-parents that share a child, then omits edge directions. In logical terms, moralization converts the (probabilistic) Horn clauses defined by a Bayes net to conjunctions of literals. Converting the Bayes net to an undirected model avoids the cyclicity problem.

Because there are many attributes (predicates) in the databases, and most of them have three possible values or more, there are many conditional probability parameters in the Bayes net. A disadvantage of the moralization approach is that it adds a clause for each conditional probability parameter, which produces a relatively large number of clauses. While this rich structure captures most of the relevant correlations in the data, the large number of clauses has several drawbacks. (i) The resulting MLN is harder for a user to understand. (ii) Parameter learning is slower. (iii) Inference is slower. (iv) Since each clause requires the estimate of a separate weight parameter, parameter estimates are less accurate. This chapter presents an extension of the moralization approach that produces significantly smaller MLN structures without sacrificing statistical power.

Decision Trees for Representing Local Independencies. As discussed by Kersting and deRaedt [46, 10.7]), a key factor for efficiently learning a graphical relational model is to search for associations between functions or *predicates*, rather than for associations between function/predicate values or *literals*. For instance, an algorithm may search for an association between the GPA of a student and the difficulty of a course she has taken, rather than an association between the literals (GPA = high) and (difficulty = high). It is well-known that because Bayes net graphs represent associations between random variables, rather than between specific values of these variables, they may fail to capture *local* or *context-sensitive* independencies that hold conditional on specific values of the random

variables [6, 27]. In the relational setting, this means that when Bayes net graphs represent associations between functions/predicates, they may not capture local independencies among literals. Thus while model search in the predicate space has efficiency advantages, it has the disadvantage of missing context-sensitive independencies. A common way to represent context-sensitive independencies is by augmenting the Bayes net with decision trees: instead of keeping a conditional probability table for each node of the Bayes net, learn a decision tree that predicts the probability of a child node value given values for its parents [6, 27, 32, 20]. Such trees are also called probability estimation trees. The main advantages of decision trees for relational models are as follows. (i) Many methods have been developed for learning decision trees that produce probability estimates [87, 21, 116, 56]. (ii) Each tree branch corresponds to a conjunction of literals and is straightforwardly converted to an MLN clause. We refer to this conversion as **context-sensitive moralization**. Figure 5.1 illustrates the system architecture for context-sensitive moralization.

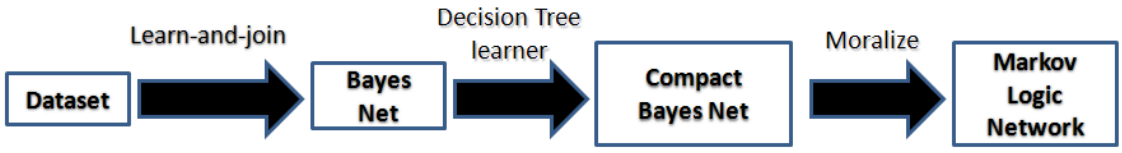


Figure 5.1: System Architecture for context-sensitive moralization: learning a compact Markov Logic Network from an input relational database.

Learning Decision Trees for Local Independencies. Given a fixed Bayes net structure, the conditional distribution of a child node v given a value assignment to its parent nodes can be learned from local statistics that involve only the family of v, which comprises v together with its parents. Our approach is to apply a standard propositional decision tree learner, which can be chosen by the user, to a data table that represents the local family statistics from the input observed database. Since propositional decision tree learners are applied "as is", our approach leverages the speed of propositional decision tree learning to achieve fast relational learning. The data table is formed by a relational join that involves only links and attributes that appear in the child node or its parents. In logical terms, the table is constructed from the tuples (groundings) that satisfy an assignment of values to the family of v. The resulting decision tree compactly models the conditional frequency, in the input database, of a child node value given an assignment of values to its parents.

Evaluation. We compared our learning algorithms with several state-of-the-art methods using public domain datasets (MovieLens, Mutagenesis, Mondial, Hepatitis). Decision tree pruning is fast and very effective in reducing the number of MLN clauses, by a factor of 5-25 depending on the dataset. The comparison with the unpruned moralized models and with LSM [60] and LHL [59],state-of-the-art MLN structure learning methods, indicates that predictive accuracy with decision trees is competitive and in many cases superior.

5.2 Additional Related Work

Bayes nets and Decision Trees. For nonrelational data, the use of decision trees has been long established to reduce the number of parameters after a Bayes net structure has been learned [6], [27, Sec.1]. Friedman and Goldszmidt use minimum description length as an objective function for Bayes net+decision tree learning [27, 90], which is motivated by the goal of obtaining a compact representation of conditional probabilities. In their work on propositional data, as in ours on relational data, the important feature of decision trees is their capacity for information compression, rather than their discriminatory power for classification.

We use Parametrized Bayes Nets [84] as a relatively straightforward extension of Bayes nets for relational data. While the combination of Parametrized Bayes nets with decision trees appears to be new, several previous statistical-relational formalisms use decision trees for compact representation of conditional probabilities. Getoor, Taskar and Koller used decision trees to augment Statistical-Relational Models [32]. The join-based syntax and semantics of Statistical-Relational Models are different from the logic-based syntax and template grounding semantics of Parametrized Bayes nets and MLNs. Logical Bayesian Networks [20] use decision trees to represent conditional probability parameters. The main difference with our use of decision trees is that the decision tree branches are interpreted as existentially quantified conjunctions of literals as in Tilde [5], which is different from the grounding semantics of MLN formulas.

Relational Dependency Networks. Dependency Networks (DNs) were introduced by Heckerman *et al.* as a graphical model that combines aspects of both Bayes nets and Markov nets [41]. Dependency networks approximate a joint distribution as the product of conditional probabilities of each node given its Markov blanket (which renders the node conditionally independent of all others). In contrast to DNs, the parameters of Bayes nets

are conditional probabilities for a node given its parents, which do *not* render a node independent of all others (except for nodes without children). Another difference between BNs and DNs is that the acyclicity constraint of BNs implies the existence of a topological ordering of the nodes, whereas nodes in a DN are unordered. For further discussion of the relationships between BNs, MNs and DNs see [41].

As a solution to the cyclicity problem, Neville and Jensen proposed upgrading Dependency Networks for relational data [78]. The differences between BNs and DNs in the propositional case carry over to Parametrized Bayes nets and Relational Dependency Networks [93]. As a consequence, propositional Bayes net learning algorithms cannot be applied for learning Relational Dependency Networks. Instead, learning algorithms for Relational Dependency Networks have been based on learning independent conditional probability models for each node. In particular, Neville and Jensen use relational probability trees for learning conditional probabilities in Relational Dependency Networks [78]; these decision trees require the specification of aggregate functions. Natarajan et al. propose the use of functional gradient boosting to learn such trees [74]. Functional gradient boosting has also been used to learn relational regression trees that are converted to MLN clauses [54], similar to our moralization approach. We compare relational regression tree learning with Bayes net+decision tree learning in Section 5.5.2 below, after we have presented the details of our algorithm.

5.3 Examples

Table 5.1 shows the university relational schema that was introduced in Chapter 2. Figure 5.2 shows a small relational database instance for this schema for illustration.

Student(*Name*, *intelligence*, *ranking*)
Course(*Number*, *difficulty*, *rating*)
Professor (*Name*, *teaching_ability*, *popularity*)
Registration (*S.name*, *C.number*, *grade*, *satisfaction*)
RA (*S.name*, *P.name*, *salary*, *capability*)

Table 5.1: The relational schema for a university domain that was introduced in Chapter 2. Key fields are underlined.

In the schema of Table 2.1 the attribute *ranking* of the *Student* table can be represented as a functor node $ranking(S)$ where S is a first-order variable ranging over the population of

Course			
Number	Prof	rating	difficulty
101	Oliver	3	1
102	David	2	2
103	Oliver	3	2

Student		
Name	intelligence	ranking
Jack	3	1
Kim	2	1
Paul	1	2

Professor		
Name	popularity	teaching Ability
Oliver	3	1
Jim	2	1

Registration			
S.name	C.number	grade	satisfaction
Jack	101	A	1
Jack	102	B	2
Kim	102	A	1
Kim	103	A	1
Paul	101	B	1
Paul	102	C	2

RA			
S.Name	P.Name	salary	capability
Jack	Oliver	High	High
Kim	Oliver	Med	Med
Kim	Jim	Low	Med

Figure 5.2: A simple relational database instance for the relational schema of Table 5.1.

students. The range of this functor node comprises 3 values, 1, 2, 3 that represent different ranking levels. The *Registration* relationship can be represented as a Boolean functor node, or predicate, $\mathit{Registered}(S, C)$, whose range is is $\{T, F\}$. The functor node $\mathit{grade}(S, C)$ represents the grade of a student in a course, which is a descriptive attribute associated with a registration link. Table 5.2 shows the database frequencies of various conjunctions.

Table 5.2: To illustrate the computation of the frequency of a conjunction formula in the database example of Figure 5.2.

Formula	#groundings	Frequency $P_{\mathcal{D}}$
$\mathit{ranking}(S) = 1$	2	$2/3$
$\mathit{ranking}(S) = 1, \mathit{intelligence}(S) = 2$	1	$1/3$
$RA(S, P) = T$	3	$3/(3 \cdot 3)$
$RA(S, P) = T, \mathit{popularity}(P) = 3$	2	$2/9$

Figure 5.3 shows a Parametrized Bayes net for the schema of Table 5.1, where each descriptive attribute and each relationship table (link type) is represented as a functor node. The Bayes net was learned from an expanded version of the database in Figure 5.2 using the learn-and-join algorithm. Figure 5.4 illustrates the moralization process.

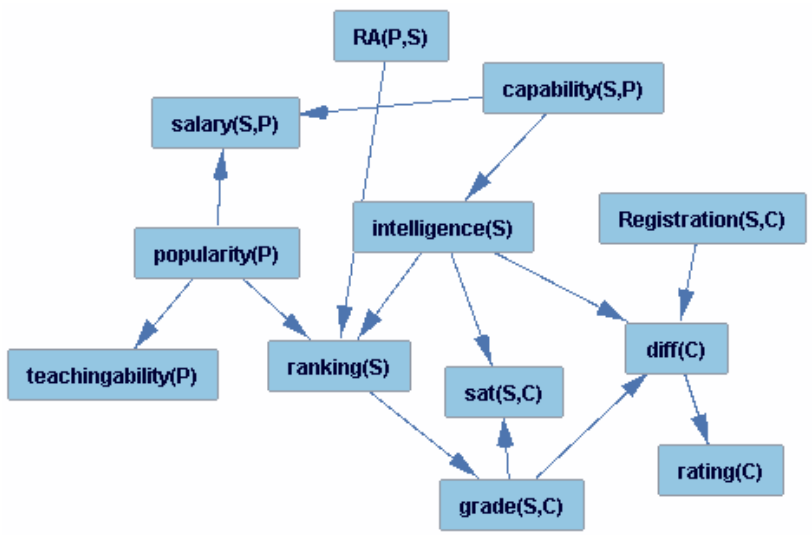


Figure 5.3: A Parametrized Bayes net graph for the relational schema of Table 5.1.

Pop (P)	Int (S)	RA(P,S)	Rank(S) = 1	Rank(S) = 2	Rank(S) = 3
1	1	True	$r_{1,1}$	$r_{2,1}$	$r_{3,1}$
1	1	False	$r_{1,2}$	$r_{2,2}$	$r_{3,2}$
..	..	...	...	...	...
..	..	...	...	...	...
3	3	False	$r_{1,18}$	$r_{2,18}$	$r_{3,18}$

$w_{1,1}$	Pop(P,1), Int(S,1), RA(P,S,True), Rank(S,1)
$w_{2,1}$	Pop(P,1), Int(S,1), RA(P,S,True), Rank(S,2)
$w_{3,1}$	Pop(P,1), Int(S,1), RA(P,S,True), Rank(S,3)
$w_{1,2}$	Pop(P,1), Int(S,1), RA(P,S,False), Rank(S,1)
....	
$w_{3,18}$	Pop(P,3), Int(S,3), RA(P,S,False), Rank(S,3)

Figure 5.4: The figure on the left shows a conditional probability table for the functor node *ranking*(S) in the Parametrized Bayes net of Figure 5.3. We use obvious abbreviations for functors. The range of *popularity*, *intelligence*, *ranking* is $\{1, 2, 3\}$ and the range of $RA = \{True, False\}$. A tabular representation therefore requires a total of $3 \times 3 \times 2 \times 3 = 54$ conditional probability parameters. The figure on the right illustrates the corresponding 54 clauses, one for each row in the conditional probability table.

5.4 Augmenting Bayes Nets with Decision Trees

We first discuss decision trees for representing conditional probabilities in Parametrized Bayes nets, then how to convert the decision tree branches to Markov Logic Network clauses.

5.4.1 Decision Trees for Conditional Probabilities.

Local or **context-sensitive** independencies are a well-known phenomenon that can be exploited to reduce the number of parameters required in a Bayes net. Suppose that a node X has three binary parents U, V, W. It may be the case that $P(x|u, V, W)$ is equal to some constant p_1 regardless of the values taken by V and W. Then the Bayes net requires only 5 parameters rather than 8 as in a tabular representation. A **decision tree** can take advantage of local independencies to compactly represent conditional probabilities [6]. The nodes in a decision tree for a Parametrized random variable *class* are parametrized random variables. An edge that originates in a PRV $f(t_1, \ldots, t_k)$ is labelled with one of the possible values in the range of f. The leaves are labelled with probabilities for the different possible values of the *class* variable. Figure 5.5 shows a tree with $class = ranking(S)$. In decision tree research, it is common to term such trees *probability estimation trees* to distinguish them from classification trees that select a definite class label at the leaves. In this chapter, we follow the usage in Bayes net research and statistical-relational learning and refer to probability estimation trees simply as decision trees.

5.4.2 Context-sensitive Moralization: Converting Decision Trees to MLN Clauses

A Parametrized Bayes net structure with decision trees can be converted to an MLN by adding a clause for each complete branch in the tree that is the conjunction of the literals along the branch. Figure 5.5 illustrates a decision tree representation of the conditional probabilities for the child node $ranking(S)$ and the clauses corresponding to the complete branches. Comparing Figure 5.5 with Figure 5.4 illustrates how the decision tree representation of the conditional probability parameters produces fewer clauses than standard moralization shown. In terms of MLN clauses, pruning decision tree nodes corresponds to merging clauses. A decision tree model may have branches of different sizes, so the clauses that are extracted for one child node may vary in the number of predicates.

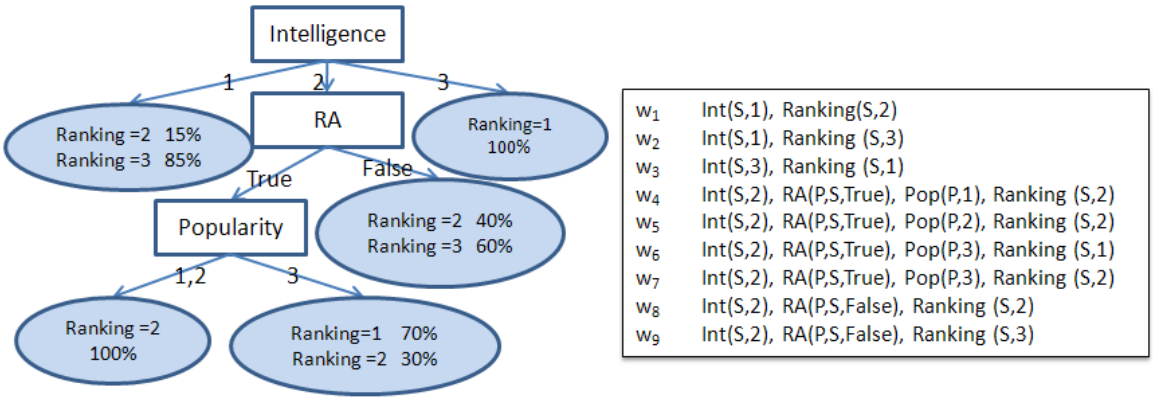


Figure 5.5: A decision tree that specifies conditional probabilities for the *ranking*(S) node in Figure 5.3 and the corresponding MLN clauses generated from the decision tree . The number of clauses has been reduced from 54 to 18

5.5 Learning Decision Trees for a Bayes Net Structure

We discuss how the decision tree representation can be combined with the learn-and-join algorithm. We use it to find an initial Bayes net structure from a relational database. The Bayes net structure is then augmented with decision trees. The approach of this chapter works with any structure learning algorithm for Parametrized Bayes nets, not just the learn-and-join algorithm.

5.5.1 Learning Decision Trees for Conditional Probabilities.

For a fixed Parametrized Bayes net structure, we learn a set of decision trees that represent the conditional probabilities of each node given an assignment of values to its parents. Our system design is modular and can use any propositional decision tree learner that estimates class probabilities at the leaves. As the learn-and-join algorithm applies a Bayes net learner to join tables, see 3, we apply the decision tree learner to the same type of join tables as follows, for each node v in the Bayes net.

1. Form a **family join table** that combines all functor nodes in its family. This table is constructed from the set of all groundings that satisfy all literals that can be formed from the functor nodes in the family, that is, all possible assignments of values to nodes in the family.

2. Omit the ids of entities from the family join table, and apply the decision tree learner to the remaining columns. The result is a tree that represents, for each possible assignment of values to the parents of node v, a corresponding conditional probability in its leaf.

In a generic data table T, the conditional probability $P_T(\mathit{child} = \mathit{value}|\mathit{parents} = \mathbf{pa})$ is the number of rows with $\mathit{child} = \mathit{value}$ and $\mathit{parents} = \mathbf{pa}$, divided by the number of rows with $\mathit{parents} = \mathbf{pa}$. The family join table is constructed such that the conditional probability in the table is the number of groundings in the database that satisfy $\mathit{child} = \mathit{value}$ and $\mathit{parents} = \mathbf{pa}$, divided by the number of groundings that satisfy $\mathit{parents} = \mathbf{pa}$. Therefore a decision tree learner applied to the family data table learns a model of the conditional probabilities $P_{\mathcal{D}}(\mathit{child} = \mathit{value}|\mathit{parents} = \mathbf{pa})$ defined by the database distribution $P_{\mathcal{D}}$.

After augmenting the Bayes net structure with decision trees, we convert the decision tree branches to Markov Logic Network clauses to obtain an MLN structure. Algorithm 3 summarizes the MLN structure learning algorithm in pseudo code.

Algorithm 3: Pseudocode for compact MLN structure learning using the learn-and-join structure learning algorithm with decision trees.

Input: Database instance $\mathcal{D}$
Output: MLN for $\mathcal{D}$
Calls: $\mathit{LearnAndJoin}(\mathcal{D})$: Outputs a DAG G for an input database $\mathcal{D}$.
Calls: $\mathit{Join}(V)$: Takes in a set of nodes from $\mathcal{D}$ and outputs the data join table for the nodes in V.
Calls: $\mathit{DecisionTree}(T, \mathit{child})$: A Decision Tree learner that outputs conditional class probabilities for *child* given data table T.

```
G = LearnAndJoin(D)
for all nodes v in G do
    v_family = v + Parents(v)
    T = Join(v_family)
    Tree_v = DecisionTree(T, v)
    for all leaf node entries of Tree_v do
        Add to MLN M the conjunction that corresponds to the decision tree branch of the leaf node.
    end for
end for
Return MLN M
```

Example. The family join table for the node *ranking*(S) is the join of the tables RA, *Student*, *Professor*, followed by projecting (selecting) the attributes *ranking*,*intelligence*, and *popularity*. Figure 5.6 illustrates the join data table for the example database of Figure 5.2. This data table provides the satisfying groundings for all literals that involve the four functor nodes in the family of *ranking*(S), conditional on the existence of an RA relationship, that is conditional on $RA(S, P) = T$.

S.name(S)	P.name(P)	ranking(S)	intelligence(S)	popularity(P)
Jack	Oliver	3	1	1
Kim	Oliver	2	1	1
Kim	Jim	2	1	2

Figure 5.6: The join data table for learning a decision tree that represents the conditional probabilities of *ranking*(S) given its parents *intelligence*(S), *popularity*(P), $RA(S, P)$, where $RA(S, P) = T$. The tuples shown are the ones from the RA table, joined with the applicable information from the *Professor* and *Student* tables. The last three columns are given as input data to a propositional decision tree learner, with *ranking*(S) designated as the class label.

A decision tree learner applied to such a data table might produce the decision tree shown in Figure 5.7(left). Since the popularity of a random professor is independent of that of a random student, the association between the functor nodes *popularity*(P) and *ranking*(S) depends on the existence of an RA link between them (i.e., the popularity of a professor predicts the ranking of a student only if the student is an RA for the professor). To indicate this dependence, we add the functor node $RA(S, P)$ as a parent of *popularity*(P) in the decision tree for *ranking*(S). Like most statistical-relational systems [30, 9], and like the learn-and-join algorithm, we consider associations between attributes of two entities only conditional of the existence of a link between the entities. Therefore there is no branch corresponding to $RA(S, P) = F$ in the decision tree of Figure 5.7(right). We leave as a project for future work learning associations conditional on the *absence* of a link between two entities (e.g., an association between the ranking of a student and the popularity of a professor given that the student is *not* an RA for the professor).

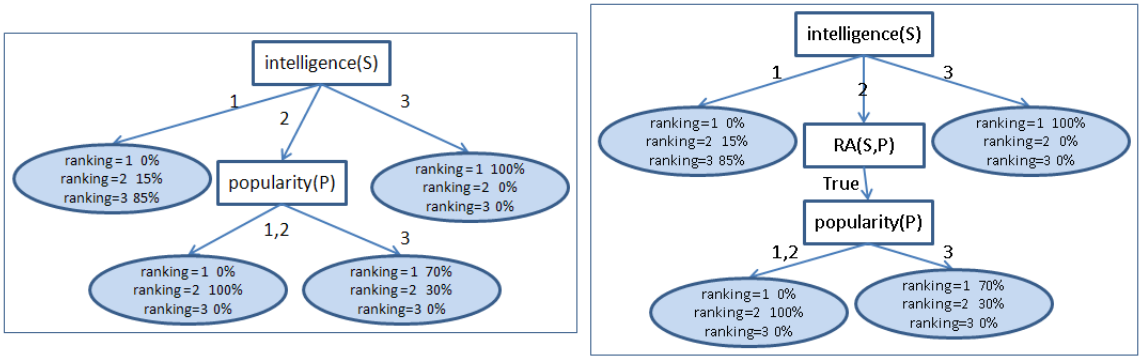


Figure 5.7: Left: A decision tree learner applied to an input table like that shown in Figure 5.6 may produce this decision tree. Right: The split on the node $popularity(P)$ depends implicitly on the existence of an RA link between a professor and a student. The output of the decision tree learner is augmented with a functor node $RA(S, P)$ to indicate this dependence.

5.5.2 Discussion.

Two bodies of related work are relevant: how to learn probability estimation trees for a single table, and how to upgrade a propositional decision tree learner for relational data. Most work on upgrading decision tree learning for relational data has been on learning classifiers rather than probability estimation trees.

Learning Probability Estimation Trees. In a seminal paper, Provost and Domingos observed that algorithms that build decision tree classifiers may not lead to good class probability estimates, mainly because trees for classification may be too small [87]. A number of improvements for probability estimation have been suggested, including the use of local probability models at the leaves of a tree [87, 21, 116, 56]. Our focus in this chapter is on whether the decision tree *representation* is sufficient in principle to produce more compact Markov Logic Networks; we leave exploring different tree learners for future work.

Upgrading Propositional Tree Learners. The tree learning approach in this chapter can be viewed as a form of *lifted learning*, in analogy to lifted probabilistic inference [84]. Lifted inference uses as much as possible frequency information defined at the class level in terms of first-order variables, rather than facts about specific individuals. Likewise, our

approach uses frequency information defined in terms of first-order variables, namely the number of satisfying groundings of a first-order formula, which is provided by the family join table. Applying a propositional learner to the family join table can be justified theoretically using the random instantiation pseudo-likelihood measure [93]. Some ILP systems for discriminative learning, such as FOIL and Linus [16], are also based on the number of groundings of various clauses, which is similar to the join tables constructed by our algorithm.

Propositionalization approaches use aggregate functions to "flatten" relational data into a single table. Inductive Logic Programming (ILP) systems learn clauses that classify an example as positive by logical entailment [16, 5]. Typically this involves the use of existential quantification as an aggregation mechanism. Relational probability trees employ a range of aggregate functions as features for predicting class probabilities [78]. While Markov Logic networks can be extended with aggregate functions, the basic log-linear prediction model of MLNs is different from approaches that use aggregate features for classification.

Gradient Boosting for Relational Regression Trees. A recent method for learning Markov Logic Networks, developed independently of our work on decision trees, is functional gradient boosting of regression trees [54]. The basic similarity is that paths in the regression tree are converted to MLN formulas in a similar manner to our moralization method. The main differences to our Bayes net approach are as follows.

Model Class. (i) The leaves of a relational regression tree contain general weights for the MLN clauses. Hence the tree cannot be interpreted as specifying a conditional probability. (ii) Boosting produces an ensemble of trees for each node (target predicate), rather than a single tree as in our system.

Learning. (i) Two-class boosting is used, so the learning method can be applied to Boolean predicates (e.g., relationship literals), but not immediately to multi-valued attributes, which are the focus of our evaluation.[1] (ii) Gradient boosting performs simultaneously MLN structure and parameter learning, because the regression trees specify both the Markov blanket of a node and the weights of clauses. In contrast, we apply decision tree learning only to obtain a more compact structure, and we apply it only to the parents of a target node, not to its entire Markov blanket (which in a Bayes net includes children and co-parents). This means that decision trees for different nodes are learned independently

[1] It is possible to change the data representation so that all attributes are binary, see [58, 53, 74].

of each other. By contrast, in the gradient boosting approach, trees learned for one node v produce clauses that are applied to learning trees for other nodes (namely, those in the Markov blanket of v).

Since functional gradient boosting is a powerful and very general framework for regression, a promising topic for future work is to apply gradient boosting for augmenting a Bayes net with decision trees, by learning a set of decision trees for a target node v conditional on its parents. While the learned set of decision trees may not be as easy to interpret as a single one [74, Sec.3.5], the Markov Logic Network that results from converting the trees is a set of clauses in either case.

5.6 Experimental Design

We first discuss the datasets used, then the systems compared, finally the comparison metrics.

Datasets. We used 4 benchmark real-world databases. For more details please see the references in [53] and on-line sources such as [48].

MovieLens Database. This is a standard dataset from the UC Irvine machine learning repository.

Mutagenesis Database. This dataset is widely used in ILP research. It contains information on Atoms, Molecules, and Bonds between them. We use the discretization of [53].

Hepatitis Database. This data is a modified version of the PKDD02 Discovery Challenge database. The database contains information on the laboratory examinations of hepatitis B and C infected patients.

Mondial Database. This dataset contains data from multiple geographical web data sources. We followed the modification of [100], and used a subset of the tables and features for fast inference.

Table 5.3 lists the resulting full database sizes in terms of total number of tuples and number of ground atoms, which is the input format for Alchemy. We also show the average values for the functor nodes in the database.

Comparison Systems. All simulations were done on a QUAD CPU Q6700 with a 2.66GHz CPU and 8GB of RAM. Our code and datasets are available on the world-wide web [48].

Dataset	#tuples	#Ground atoms	#Values (Average)
Movielens	82623	170143	2.7
Mutagenesis	15218	35973	3.9
Hepatitis	12447	71597	4.4
Mondial	814	3366	4.3

Table 5.3: Size of full datasets in total number of table tuples and ground atoms. Each descriptive attribute is represented as a separate function, so the number of ground atoms is larger than that of tuples. The average number of values of the descriptive attributes in each database is shown. Binary descriptive attributes (e.g., gender) are rare.

We made use of the following existing implementations.

Single Table Bayes Net Search GES search [11] with the BDeu score as implemented in version 4.3.9-0 of CMU's Tetrad package (structure prior uniform, ESS=10; [106]).

Single Table Decision Tree Learning The J48 program of the Weka package [35], which implements the C4.5 decision tree algorithm. We used the probability estimation setting, which turns off pruning and applies the Laplace correction, as recommended by Provost and Domingos [87].

MLN Parameter Learning The default weight training procedure [67] of the Alchemy package [61], Version 30.

MLN Inference The MC-SAT inference algorithm [85] to compute a probability estimate for each possible value of a descriptive attribute for a given object or tuple of objects.

We use the Alchemy parameter learning and the state-of-the art MC-SAT inference method for compatibility with previous studies [70, 59, 60]. We also carried out simulations with an exact evaluation of the MLN classification formula given by Domingos and Richardson [15]. The relative improvement of context-sensitive vs. standard moralization is similar, but the absolute accuracies are lower, which is consistent with the findings of previous studies of MLN inference.

Algorithms. We compared four MLN structure learning algorithms.

MBN The structure is learned using the learn-and-join algorithm (Section 3.4). The weights of clauses are learned using Alchemy.

MBN + DT The structure is first learned using the learn-and-join algorithm and then augmented with decision trees using Algorithm 1. As illustrated in Figure 5.5, this algorithm produces clauses with positive relationship literals only. The weights of clauses are learned using Alchemy.

LHL Lifted Hypergraph Learning [59] uses relational path finding to induce a more compact representation of data, in the form of a hypergraph over clusters of constants. Clauses represent associations among the clusters.

LSM Learning Structural Motifs [60] uses random walks to identify densely connected objects in data, and groups them and their associated relations into a motif.

The first two methods compare variants of the moralization method, whereas the last two are reference methods. We chose LSM and LHL because they are the most recent MLN structure learning methods that are based on the Alchemy system.

Performance Metrics. We use 4 performance metrics: Number of Clauses or Parameters, learning time, Accuracy (ACC), and Conditional log likelihood (CLL). ACC and CLL have been used in previous studies of MLN learning [59]. The CLL of a ground atom in a database given an MLN is its log-probability given the MLN and the information in the database. Accuracy is evaluated using the most likely value for a ground atom. For ACC and CLL the values we report are averages over all predicates that represent descriptive attributes. We do not use Area under Curve (AUC), as it mainly applies to binary values, and most of the attributes in our dataset are nonbinary. We evaluate the learning methods using two different schemes.

5-fold cross-validation. We formed 5 subdatabases for each using standard subgraph subsampling [24, 53], which selects entities from each entity table uniformly at random and restricts the relationship tuples in each subdatabase to those that involve only the selected entities. The models were trained on 4 of the 5 subdatabases, then tested on the remaining fold. We report the average over the 5 runs, one for each fold.

Learning Curve. To study the learning behavior at different sample sizes, we performed a set of experiments that train the model on N% of the data, where N ranges from 10 to 100 in step sizes of 10. Results for each sample size are averages over 10 runs.

	MBN + DT	MBN	LSM	LHL
MovieLens	39	327	10	NT
Mondial	102	2470	20	25
Mutagen	50	880	13	NT
Hepatitis	120	793	23	27

Table 5.4: 5-fold cross-validation estimate of the number of parameters in learned model.

5.7 Evaluation Results

As our aim is to learn more compact structures, we first examine the number of parameters or clauses learned. Our results indicate that the decision tree representation leads to substantially more compact models. Learning time measurements show a significant speed-up in weight learning with the smaller models. Predictive performance is competitive with standard moralization based on conditional probability tables, in many cases even superior.

5.7.1 Number of Parameters/Clauses

Table 5.4 shows the average number of clauses produced by the MLN models using 5-fold cross-validation (so the average is over 5 different measurements). *Adding decision trees to the learn-and-join algorithm leads to much more compact models*, with improvement ratios in the range of 5-25. The LSM algorithm and the LHL algorithm learn a very small number of clauses most of which are unit or short clauses. Figure 5.8 shows the number of parameters learned respectively by LSM, LHL, and MBN + DT using a Learning Curve scheme. MBN + DT exploits increasing data to learn a more complex model. LSM and LHL learn almost the same very small number of clauses independent of the data size. Inspection of the learned clauses by LSM and LHL shows that the rules are mostly just the unit clauses that model marginal probabilities (e.g., $\mathit{intelligence}(S, I)$) [15]. This indicates underfitting the data, as our measurements of ACC and CLL confirm (Tables 5.6, 5.7 below).

5.7.2 Learning times

Table 5.5 shows average times for learning using 5-fold cross-validation. LHL fails to terminate on two of the datasets and is very slow on the other two datasets. The learn-and-join structure learning method scales well, even with decision tree learning added. The computational bottleneck for the two moralization methods is the weight optimization that uses

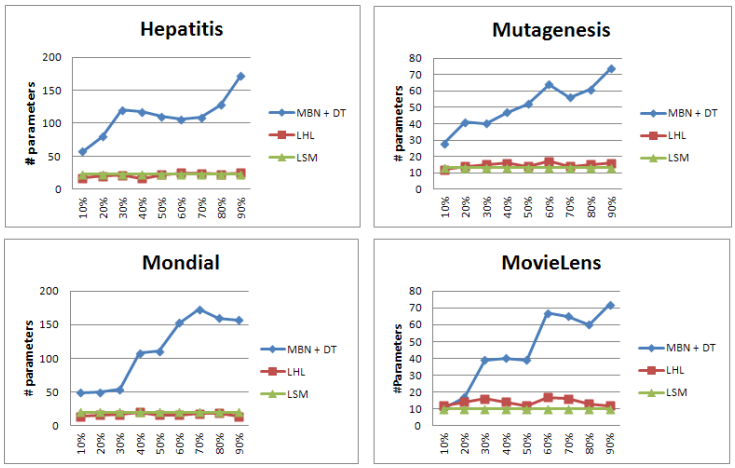


Figure 5.8: The number of parameters learned using a Learning Curve scheme. MBN + DT learns more clauses when more data is available. LSM and LHL produce a very small number of clauses that is essentially independent of the data size. The experiments train the model on N% of the data, where N ranges from 10 to 100 in step sizes of 10.

the relatively slow Alchemy routines. *Since decision trees reduce the number of model parameters, they speed up weight optimization by a factor of about 10.* So a small increase in the complexity of structure learning achieves a very significant decrease in the complexity of parameter learning.

To further examine the scalability of the moralization algorithms, we use a Learning Curve design. Figure 5.9 indicates that the weight learning time for MBN increases exponentially with the size of the dataset, but adding decision trees leads to much better scaling. The LSM method (not shown) is very fast for all dataset sizes, because it produces

	MBN + DT	MBN	LSM	LHL
MovieLens	22 + 345	15 + 3401	34.03	NT
Mondial	9 + 18	4+ 1168	29.0	11524
Mutagen	18 + 274	12 + 4425	26.47	NT
Hepatitis	21 + 813	15 + 6219	10.94	72452

Table 5.5: 5-fold cross-validation estimate for Average learning times in seconds. Learning times for the moralization methods are given as (structure learning time + weight learning time).

a small set of short rules that is essentially independent of the dataset size. This is mostly due to under-fitting as indicated by the number of parameters in LSM models, and by the predictive accuracy results, which we report next.

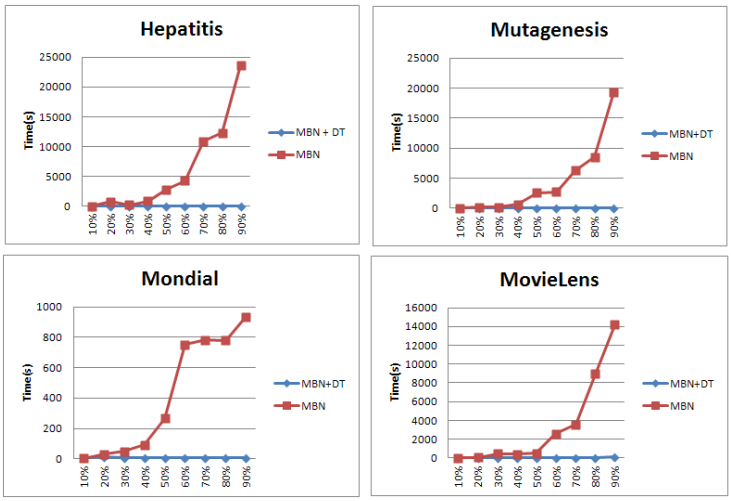


Figure 5.9: Weight learning time curve is shown in this figure.While structure learning is fast using the learn-and-join algorithm, the weight learning time for standard moralization method MBN increases exponentially with the size of the database. Context-sensitive moralization with decision trees scales much better (MBN+DT). The experiments train the model on N% of the data, where N ranges from 10 to 100 in step sizes of 10.

5.7.3 Predictive Performance

We first discuss accuracy, then conditional log-likelihood.

Accuracy. The average accuracy of the two learn-and-join methods is quite similar, and *about 10-15% higher than that of LSM and LHL.* The accuracy numbers are fairly low overall because many of the descriptive attributes have many possible values (e.g. 9 for Lumo in Mutagenesis); see Table 3.1. The LSM accuracy variance is low, which together with poor average accuracy is consistent with the hypothesis that LSM underfits the data.

The moralization method performs generative learning over all attributes, a significantly more difficult task than discriminative learning. While it is usual in Markov Logic Network evaluation to report an average over all predicates in a database, we observed that there is

	MBN + DT	MBN	LSM	LHL
MovieLens	0.55 ± 0.04	**0.56 ± 0.04**	0.39 ± 0.04	NT
Mondial	**0.41 ± 0.054**	**0.41 ± 0.055**	0.26 ± 0.018	0.25± 0.03
Mutagen	**0.58 ± 0.064**	0.54 ± 0.074	0.45 ± 0.043	NT
Hepatitis	**0.51 ± 0.04**	**0.51 ± 0.01**	0.3 ±0.01	0.37 ± 0.052

Table 5.6: 5-fold cross-validation estimate for the accuracy of predicting the true values of descriptive attributes, averaged over all descriptive attribute instances. Observed standard deviations are shown.

considerable variance among the predictive accuracies for different predicates. For example in the Mutagenesis dataset, the accuracy of the learned MBN model for predicting positive mutagenicity is 87% on 10-fold cross-validation, which is in the 86%−88% range of accuracies reported for discriminative methods [103, 88, 99].

Conditional Log-likelihood. This measure is especially sensitive to the quality of the parameter estimates. Without decision trees, the MBN method performs clearly worse on 3 of the 4 datasets, both in terms of average CLL and variance. The CLL performance of LSM is acceptable on average. The parameter estimates are biased towards uniform values, which leads to predictions whose magnitudes are not extreme. Because the average accuracy is low, this means that when mistaken predictions are made, they are not made with great confidence.

	MBN + DT	MBN	LSM	LHL
MovieLens	-0.8 ± 0.25	-0.79 ± 0.12	**-0.65 ± 0.10**	NT
Mondial	**-1.36 ± 0.12**	−1.76 ± 0.37	-1.43 ± 0.027	-1.98 ± 0.035
Mutagen	**-0.97 ± 0.0134**	-1.31 ± 0.197	-1.01± 0.065	NT
Hepatitis	**-1.16 ±0.04**	-1.74 ± 0.08	-1.36 ±0.03	-2.13 ± 0.011

Table 5.7: 5-fold cross-validation estimate for the conditional log-likelihood assigned to the true values of descriptive attributes, averaged over all descriptive attribute instances. Observed standard deviations are shown.

Chapter 6

Summary and Conclusion

Markov Logic Networks form one of the most prominent SRL model classes which have achieved impressive performance on a variety of SRL tasks. An MLN is a set of weighted first-order formulas that compactly defines a Markov Network comprising ground instances of logical predicates. In this dissertation we addressed the problem of learning the structure of Markov Logic networks in relational schemas that feature a significant number of descriptive attributes, compared to the number of relationships. The main motivation for writing this dissertation was that previous MLN learning algorithms do not scale well with such datasets.

In Chapter 3, we considered the task of building a non recursive statistical-relational model for databases with many descriptive attributes. We combined Bayes net learning, one of the most successful machine learning techniques, with Markov Logic Networks, one of the most successful statistical-relational formalisms. Our approach is a hybrid method that uses directed models for learning and undirected models for inference.The moralization approach, combines the learning efficiency, scalability, and interpretability of directed relational models with the inference power and theoretical foundations of undirected relational models. The main algorithmic contribution of this Chapter is an efficient new structure learning algorithm for a parametrized Bayes net that models the joint frequency distribution over attributes in the database, given the links between entities. Moralizing the BN leads to a Markov Logic Network structure. Our evaluation on six benchmark databases with descriptive attributes shows that compared to current Markov Logic Network structure learning methods, the approach using moralization improves the scalability and run-time performance by orders of magnitude. With default parameter estimation algorithms from the alchemy package, the

moralized Markov Logic Network structures make substantially more accurate predictions on standard metrics.

In Chapter 4, we presented a new method for applying Bayes net learning for *recursive dependencies* based on a recent pseudo-likelihood score and a new normal form theorem. The pseudo-likelihood score quantifies the fit of a recursive dependency model to relational data, and allows us to efficiently apply model search algorithms. A new normal form eliminates potential redundancies that arise when predicates are duplicated to capture recursive relationships. For stratified directed models, we proved that the normal form involves no loss of expressive power. In evaluations, our structure learning method was very efficient and found recursive dependencies that were missed by Markov Logic Network structure learning methods.

In Chapter 5 we proposed methods for learning compact Markov Logic Networks via Bayes nets and decision trees. Augmenting Bayes net learning with decision tree learning leads to a compact set of Horn clauses that represent generative statistical patterns in a relational database. The main algorithmic contribution of this Chapter is an efficient structure learning algorithm for MLNs that uses Bayes nets and decision trees to compactly model the joint frequency distribution over attributes in the database. In our simulations on four benchmark relational databases, decision trees significantly reduced the number of Bayes net parameters, by factors ranging from 5-25. The pattern of average predictive performance and its variance is consistent with the hypothesis that the decision tree method strikes an attractive balance: It avoids the overfitting tendencies of the basic Bayes net moralization algorithm, and it avoids the underfitting tendencies of the Markov Logic learners (LSM and LHL). After converting the Bayes net Horn clauses to Markov Logic Networks, MLN inference can be used to evaluate the predictive accuracy of the resulting models. In our empirical evaluation, the predictive performance of the pruned models is competitive with or superior to the unpruned models.

The algorithms and scripts developed for this dissertation have all been documented and are available online [48].

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc International Conference on Very Large Databases*, pages 478–499, Santiage, Chile, 1994. Morgan Kaufmann, Los Altos, CA.

[2] Krzysztof R. Apt and Marc Bezem. Acyclic programs. *New Generation Comput.*, 9(3/4):335–364, 1991.

[3] Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

[4] Marenglen Biba, Stefano Ferilli, and Floriana Esposito. Structure learning of Markov logic networks through iterated local search. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI*, pages 361–365, 2008.

[5] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[6] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in bayesian networks. In *UAI*, pages 115–123, 1996.

[7] Ivan Bratko. *Prolog (3rd ed.): programming for artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[8] George Casella and Edward I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[9] Hailiang Chen, Hongyan Liu, Jiawei Han, and Xiaoxin Yin. Exploring optimization of semantic relationship graph for multi-relational Bayesian classification. *Decision Support Systems*, 48:1:112–121, July 2009.

[10] J. Cheng and R. Greiner. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137:43–90, 2002.

[11] D. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2003.

[12] Jennifer Neville David Jensen. Linkage and autocorrelation cause feature selection bias in relational learning (2002). In *ICML*, 2002.

[13] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Mach. Learn.*, 26(2-3):99–146, 1997.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[15] Pedro Domingos and Matthew Richardson. Markov logic: A unifying framework for statistical relational learning. In *Introduction to Statistical Relational Learning* [33].

[16] Saso Dzeroski. Inductive logic programming in a nutshell. In *Introduction to Statistical Relational Learning* [33].

[17] F. Y. Edgeworth. On the probable errors of frequency-constants (contd.). *Journal of the Royal Statistical Society*, 71(3):499–512, 1908.

[18] Thomas Eiter and Georg Gottlob. Propositional circumscription and extended closed world reasoning are π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.

[19] Daan Fierens. On the relationship between logical bayesian networks and probabilistic logic programming based on the distribution semantics. In *ILP*, pages 17–24, 2009.

[20] Daan Fierens, Hendrik Blockeel, Maurice Bruynooghe, and Jan Ramon. Logical bayesian networks and their relation to other probabilistic logical models. In *ILP*, pages 121–135, 2005.

[21] Daan Fierens, Jan Ramon, Hendrik Blockeel, and Maurice Bruynooghe. A comparison of pruning criteria for probability trees. *Machine Learning*, 78(1-2):251–285, 2010.

[22] Daan Fierens, Jan Ramon, Maurice Bruynooghe, and Hendrik Blockeel. Learning directed probabilistic logical models: Ordering-search versus structure-search. In *ECML*, pages 567–574, 2007.

[23] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160, New York, NY, USA, 2000. ACM.

[24] O. Frank. Estimation of graph totals. *Scandinavian Journal of Statistics*, 4:2:81–89, 1977.

[25] Richard Frank, Flavia Moser, and Martin Ester. A method for multi-relational classification using single and multi-feature aggregation functions. In *PKDD*, 430-437, 2007.

[26] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *In IJCAI*, pages 1300–1309. Springer-Verlag, 1999.

[27] Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In *NATO ASI on Learning in graphical models*, pages 421–459, 1998.

[28] Dan Geiger and David Heckerman. Knowledge representation and inference in similarity networks and bayesian multinets. *Artificial Intelligence*, 82(1-2):45–74, 1996.

[29] Lise Getoor. *Learning Statistical Models From Relational Data*. PhD thesis, Department of Computer Science, Stanford University, 2001.

[30] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Benjamin Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning* [33], chapter 5, pages 129–173.

[31] Lise Getoor and John Grant. Prl: A probabilistic relational language. *Machine Learning*, 62(1-2):7–31, 2006.

[32] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. *ACM SIGMOD Record*, 30(2):461–472, 2001.

[33] Lise Getoor and Ben Tasker. *Introduction to statistical relational learning*. MIT Press, 2007.

[34] Lise Getoor Getoor, Nir Friedman, and Benjamin Taskar. Learning probabilistic models of relational structure. In *ICML*, pages 170–177. Morgan Kaufmann, 2001.

[35] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[36] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[37] D. Heckerman. A tutorial on learning with Bayesian networks. In *NATO ASI on Learning in graphical models*, pages 301–354, 1998.

[38] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. In R. Lopez de Mantaras and D. Poole, editors, *Uncertainty in Artificial Intelligence*, volume 10, pages 293–301, San Mateo, CA, 1994. Morgan Kaufmann.

[39] D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In Getoor and Taskar [33].

[40] David Heckerman and John S. Breese. Causal independence for probability assessment and inference using Bayesian networks. Technical report, IEEE Trans. on Systems, Man and Cybernetics, 1995.

[41] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, Carl Kadie, and Pack Kaelbling. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[42] Tuyen N. Huynh and Raymond J. Mooney. Discriminative structure and parameter learning for markov logic networks. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, pages 416–423. ACM, 2008.

[43] David Jensen and Jennifer Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *ICML*, 2002.

[44] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–598. 2004.

[45] Michael Jordan. Graphical models. *Statistical Science (Special Issue on Bayesian Statistics)*, 19:140–155, 2004.

[46] Kristian Kersting and Luc de Raedt. Bayesian logic programming: Theory and tool. In *Introduction to Statistical Relational Learning* [33], chapter 10, pages 291–318.

[47] Kristian Kersting, Brian Milch, Luke S. Zettlemoyer, Michael Haimes, and Leslie Pack Kaelbling. Reasoning about large populations with lifted probabilistic inference. NIPS Workshop, 2008.

[48] H. Khosravi, T. Man, J. Hu, E. Gao, and O. Schulte. Learn and join algorithm code. URL = `http://www.cs.sfu.ca/~oschulte/jbn/`.

[49] Hassan Khosravi. Fast parameter learning for markov logic networks using bayes nets. In *To appear in proceedings of ILP*, 2012.

[50] Hassan Khosravi, Oliver Schulte, and Bahareh Bina. Virtual joins with nonexistent links. 19th Conference on Inductive Logic Programming (ILP), 2009. URL = `http://www.cs.kuleuven.be/~dtai/ilp-mlg-srl/papers/ILP09-39.pdf`.

[51] Hassan Khosravi, Oliver Schulte, Jianfeng Hu, and Tianxing Gao. Learning compact markov logic networks with decision trees. In *ILP*, volume 7207 of *Springer LNAI*, pages 21–26, 2011.

[52] Hassan Khosravi, Oliver Schulte, Jianfeng Hu, and Tianxing Gao. Learning compact markov logic networks with decision trees. *Machine Learning*, page Forthcoming., 2012.

[53] Hassan Khosravi, Oliver Schulte, Tong Man, Xiaoyuan Xu, and Bahareh Bina. Structure learning for Markov logic networks with many descriptive attributes. In *AAAI*, pages 487–493, 2010.

[54] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude W. Shavlik. Learning markov logic networks via functional gradient boosting. In *ICDM*, pages 320–329, 2011.

[55] Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

[56] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, pages 202–207, 1996.

[57] Stanley Kok and Pedro Domingos. Learning the structure of Markov logic networks. In Luc De Raedt and Stefan Wrobel, editors, *ICML*, pages 441–448. ACM, 2005.

[58] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *ICML*, pages 433–440. ACM, 2007.

[59] Stanley Kok and Pedro Domingos. Learning markov logic network structure via hypergraph lifting. In *ICML*, pages 64–71, 2009.

[60] Stanley Kok and Pedro Domingos. Learning Markov logic networks using structural motifs. In *ICML*, pages 551–558, 2010.

[61] Stanley Kok, M. Summer, Matthew Richardson, Parag Singla, H. Poon, D. Lowd, J. Wang, and Pedro Domingos. The Alchemy system for statistical relational AI. Technical report, University of Washington., 2009. Version 30.

[62] Daphne Koller and Avi Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI*, pages 1316–1323, 1997.

[63] Wim Van Laer and Luc de Raedt. How to upgrade propositional learners to first-order logic: A case study. In *Relational Data Mining*. Springer Verlag, 2001.

[64] Steffen L. Lauritzen. *Graphical Models (Oxford Statistical Science Series)*. Oxford University Press, USA, July 1996.

[65] Vladimir Lifschitz. Foundations of logic programming. Principles of Knowledge Representation, CSLI Publications, 1996.

[66] Huma Lodhi and Stephen Muggleton. Is mutagenesis still challenging? In *Inductive Logic Programming*, pages 35,40, 2005.

[67] Daniel Lowd and Pedro Domingos. Efficient weight learning for Markov logic networks. In *PKDD*, pages 200–211, 2007.

[68] D. Margaritis and S. Thrun. Bayes. net. induction via local neighbor. In *NIPS*, pages 505–511, 2000.

[69] Wolfgang May. Information extraction and integration: The mondial case study. Technical report, Universität Freiburg, Institut für Informatik, 1999.

[70] Lilyana Mihalkova and Raymond J. Mooney. Bottom-up learning of Markov logic network structure. In *ICML*, pages 625–632. ACM, 2007.

[71] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[72] Stephen Muggleton. Inductive logic programming. *New Gen. Comput.*, 8(4):295–318, 1991.

[73] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of Uncertainty in AI*, pages 467–475, 1999.

[74] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude W. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1):25–56, 2012.

[75] Sriraam Natarajan, Prasad Tadepalli, Thomas G. Dietterich, and Alan Fern. Learning first-order probabilistic models with combining rules. *Annals of Mathematics and Artifical Intelligence*, 54(1-3):223–256, 2008.

[76] R. E. Neapolitan. *Learning Bayesian Networks*. Pearson Education, 2004.

[77] J. Neville, D. Jensen, B. Gallagher, and R. Fairgrieve. Simple estimators for relational bayesian classifiers. In *Proceedings of the 3rd IEEE international conference on data mining*, pages 609–612. Citeseer, 2003.

[78] Jennifer Neville and David Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, 2007.

[79] Jennifer Neville and David Jensen. Relational dependency networks. In *An Introduction to Statistical Relational Learning* [33], chapter 8.

[80] Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD*, pages 625–630, New York, NY, USA, 2003. ACM Press.

[81] Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theor. Comput. Sci.*, 171(1-2):147–177, 1997.

[82] S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *LNAI*. SV, February 1997.

[83] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[84] David Poole. First-order probabilistic inference. In *IJCAI*, pages 985–991, 2003.

[85] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, 2006.

[86] Alexandrin Popescul and Lyle Ungar. Feature generation and selection in multi-relational learning. In *An Introduction to Statistical Relational Learning* [33], chapter 8.

[87] Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[88] J. Quinlan. Boosting first-order learning. In *Algorithmic Learning Theory*, pages 143–155. Springer, 1996.

[89] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. 10.1007/BF00116251.

[90] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Inf. Comput.*, 80(3):227–248, 1989.

[91] Jan Ramon, Tom Croonenborghs, Daan Fierens, Hendrik Blockeel, and Maurice Bruynooghe. Generalized ordering-search for learning directed probabilistic logical models. *Machine Learning*, 70(2-3):169–188, 2008.

[92] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[93] Oliver Schulte. A tractable pseudo-likelihood function for Bayes nets applied to relational data. In *SIAM SDM*, pages 462–473, 2011.

[94] Oliver Schulte and Hassan Khosravi. Learning graphical models for relational data via lattice search. *Machine Learning*, 88:3:331–368, 2012.

[95] Oliver Schulte, Hassan Khosravi, and Bahareh Bina. Bayes nets for combining logical and probabilistic structure. In *Proceedings STRUCK Workshop on Learning Structural Knowledge From Observations*. IJCAI-09, 2009.

[96] Oliver Schulte, Hassan Khosravi, Ted Kirkpatrick, Tianxiang Gao, and Yuke Zhu. Modelling relational statistics with bayes nets. In *To appear in proceedings of ILP*, 2012.

[97] Oliver Schulte, Hassan Khosravi, and Tong Man. Learning directed relational models with recursive dependencies. In *ILP*, volume 7207 of *Springer LNAI*, pages 39–44, 2011.

[98] Oliver Schulte, Hassan Khosravi, and Tong Man. Learning directed relational models with recursive dependencies. *Machine Learning*, page Forthcoming., 2012.

[99] Michèle Sebag and Céline Rouveirol. Tractable induction and classification in first order logic via stochastic matching. In *IJCAI*, pages 888–893, 1997.

[100] Rong She, Ke Wang, and Yabo Xu. Pushing feature selection ahead of join. In *SIAM SDM*, 2005.

[101] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2000.

[102] Sampath Srinivas. A generalization of the noisy-or model. In *Proceedings of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI-93)*, San Francisco, CA, 1993. Morgan Kaufmann.

[103] A. Srinivasan, SH Muggleton, MJE Sternberg, and RD King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[104] B. Taskar, M. Wong, P. Abbeel, and D. Koller. Link prediction in relational data, 2004.

[105] Benjamin Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *UAI*, pages 485–492, 2002.

[106] The Tetrad Group. The Tetrad project, 2008. http://www.phil.cmu.edu/projects/tetrad/.

[107] J. D. Ullman. *Principles of database systems*. 2. Computer Science Press, 1982.

[108] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[109] Moshe Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276. ACM Press, 1995.

[110] Yitong Wang and Masaru Kitsuregawa. Link based clustering of web search results. In *Lecture Notes in Computer Science*, pages 225–236. Springer, 2001.

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

[112] M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7:35–53, 1992.

[113] W. Winkler. The state of record linkage and current research problems, 1999.

[114] Xiaoxin Yin and Jiawei Han. Exploring the power of heuristics and links in multi-relational data mining. In *ISMIS'08: Proceedings of the 17th international conference on Foundations of intelligent systems*, pages 17–27, Berlin, Heidelberg, 2008. Springer-Verlag.

[115] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *Constraint-Based Mining and Inductive Databases*, pages 172–195, 2004.

[116] Harry Zhang and Jiang Su. Conditional independence trees. In *ECML*, pages 513–524. Springer, 2004.