Multi-Relational Learning with SQL All the Way

by

Zhensong Qian

M.Sc., Shandong University, 2010
B.Sc., Shandong University, 2007

Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Science

Approval

Name:	**Zhensong Qian**
Degree:	**Doctor of Philosophy (Computing Science)**
Title:	***Multi-Relational Learning with SQL All the Way***
Examining Committee:	**Chair:** Dr. James P. Delgrande Professor

Dr. Oliver Schulte
Senior Supervisor
Professor

Dr. Qianping Gu
Supervisor
Professor

Dr. Jiannan Wang
Internal Examiner
Assistant Professor

Dr. Xue (Steve) Liu
External Examiner
Associate Professor
School of Computer Science
McGill University

Date Defended: November-24-2016

Abstract

Which doctors prescribe which drugs to which patients? Who upvotes which answers on what topics on Quora? Who has followed whom on Twitter/Weibo? These relationships are all visible in data, and they all contain a wealth of information that could be extracted to be knowledge/wisdom. Statistical Relational Learning (SRL) is a recent growing field which extends traditional machine learning from single-table to multiple inter-related tables. It aims to provide integrated statistical analysis of heterogeneous and interdependent complex data. In the thesis, I focus on modelling the interactions between different attributes and the link itself for such complex heterogeneous and richly interconnected data. First, I describe the FactorBase system which combines advanced analytics from statistical-relational machine learning (SRL) with database systems. Within FactorBase, all statistical objects are stored as first-class citizens as well as raw data. This new SQL-based framework pushes the multi-relational model discovery into a relational database management system. Secondly, to solve the scalability issue of computing cross-table sufficient statistics, a new Virtual Join algorithm is proposed and implemented in FactorBase. Bayesian networks (BNs) and Dependency Networks (DNs) are two major classes of SRL. Thirdly, I utilize FactorBase to extend the state-of-the-art learning algorithm for BN of generative modelling with link uncertainty. The learned model captures correlations between link types, link features, and attributes of nodes, simultaneously. Finally, a fast hybrid approach is proposed for instance level discriminative learning of DNs with competitive predictive power but substantially better scalability.

Keywords: Statistical Relational Learning(SRL); Multi-Relational Database; FactorBase; Sufficient Statistics; Log-Linear Model; Bayesian networks (BNs); Dependency Networks (DNs); Link Analysis; Generative Modelling; Discriminative Learning.

Dedication

To my wife, parents and sisters for their constant support and unconditional love.

Acknowledgements

Firstly, I would like to express special appreciation and thanks to my senior supervisor Dr. Oliver Schulte, for his continuous support, great patience, encouraging motivation, and immense knowledge during my Ph.D. study at Simon Fraser University. Oliver has been a tremendous mentor for me, both professionally and personally. His advices on research as well as on my career have been priceless. I could not have imagined having a better senior supervisor and mentor.

Besides my senior supervisor, I would also like to thank the rest of my committee members. I am grateful to Dr. Qianping Gu for his constructive suggestions and personal encouragement throughout my entire Ph.D. study. I appreciate Dr. Jiannan Wang for his insightful comments and feedback. It is a great honor to have Dr. Xue (Steve) Liu as my external examiner. I am also thankful to Dr. James P. Delgrande for serving as the chair of my committee.

I consider myself very lucky to have the chance to collaborate with so many wonderful people who helped me to accomplish the results descripted in this thesis. I am very grateful to all my collaborators. I would also like to thank all of my friends and labmates (too many to list here but you know who you are) who supported me in writing, and incented me to strive towards my goal. I greatly appreciate the administrative and technical staff members of the school of Computing Science who have been kind enough to advise and help in their respective roles. My life would not be so enjoyable without your sharing, blessing and encouragement.

Last but not the least, I owe my deepest thanks to my family: my parents, my sisters and my wife. I know I always have my family to count on during a challenging period. My hard-working parents have sacrificed their lives for my sisters and myself and provided unconditional life-time love and care. My sisters for listening to my compliants, for their understanding and for believing in me. A very special heartfelt thanks to my wife Yanqin Liu, for her love, support and encouraging me to follow my dream. Words cannot express my gratitude for all the sacrifices they have made on my behalf. This dissertation is impossible without their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many real world datasets are relational and most real world applications are characterized by the presence of uncertainty and complex relational structures (see an example in Figure 1.1 of the domain of online social media). Using interconnected information as the input provides a richer knowledge to the machine and helps the process of learning. The domains often consist of a variety of object types; the interactions between these objects can be linked into multiple types as well. A university network may consist of several kinds of object like students, professors and courses, as well as the interactions, such as teaching, course registration or research assistant relationship. Similar types of instances are ubiquitous, from online social media to scientific research publication networks, public health care systems, and to electrical power grids, and so on. To tackle the interesting yet critical problem of modelling such heterogeneous data, appropriate care must be taken due to the correlations of features and/or different kinds of links [84]: attributes of linked objects are often correlated, and links are more likely to exist between objects that share similar attributes. In the thesis, I focus on modelling the interactions between different attributes and the link itself for complex heterogeneous and richly interconnected data.

1.1 Multi-Relational Data Representation

The choice of an appropriate representation is an important issue which has a significant impact on the quality of the statistical inferences that can be made. The structure of inter-related/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. With the current surroundings, the general *entity* and *relationship* (ER) diagram from the database community is a powerful and expressive representation of the complex real-world interactions between multiple kinds of objects and links in diverse domains. Additionally techniques provided by the relational database management system(RDBMS) can be leveraged for manipulating data as well as models.

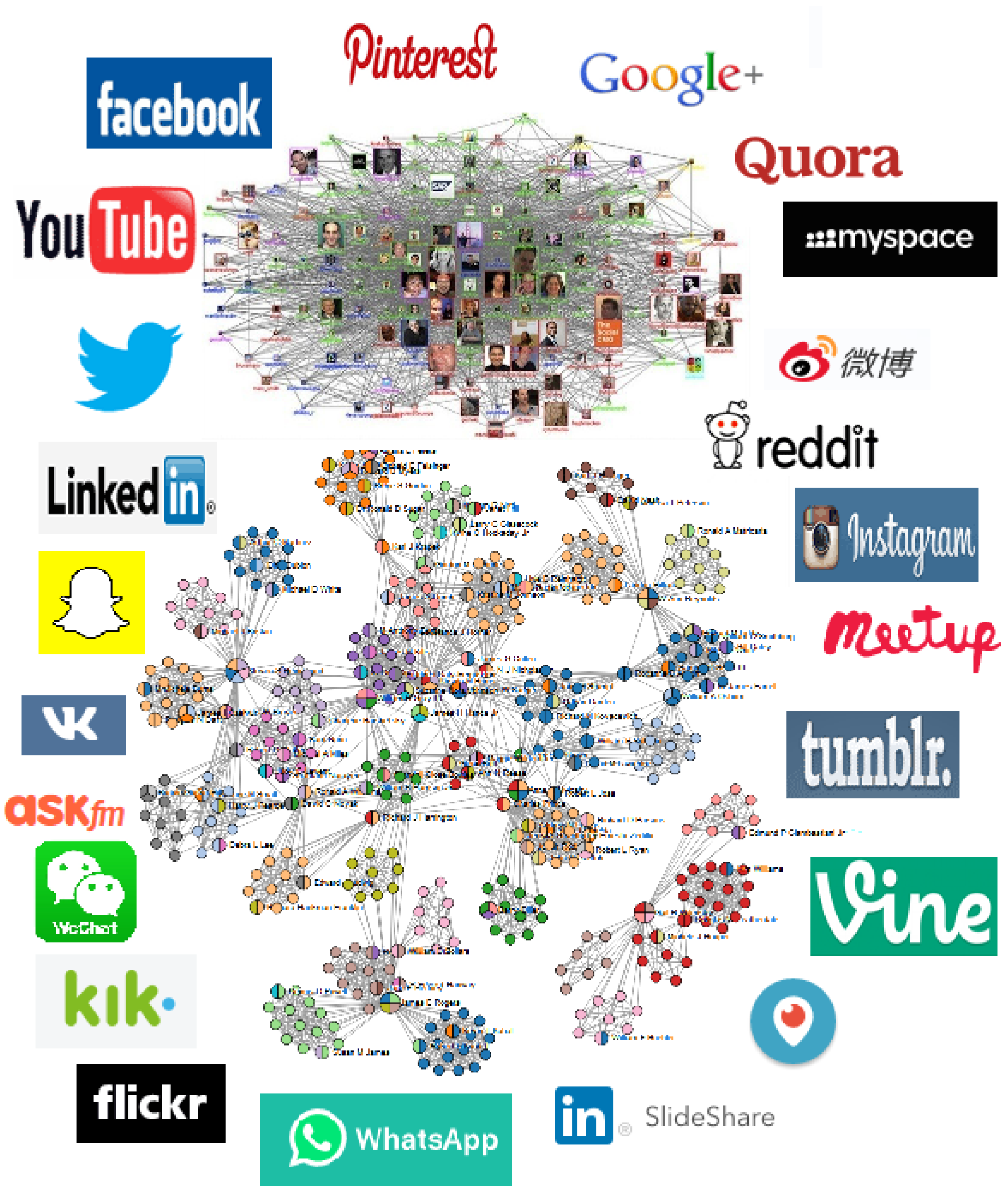


Figure 1.1: Example of interconnected online social media.

In the following chapters, I will assume the format data is represented as an ER diagram. A standard **relational schema** contains a set of tables, each with key fields, descriptive attributes, and possibly foreign key pointers. The tables in the relational schema can be divided into *entity tables* and *relationship tables* [88, Ch.2.2]. An ER diagram shows entity sets as boxes, relationships between entities as diamonds, and attributes as ovals.

Figure 1.2 shows a relational diagram for a database related to a university which contains three objects or entity tables: *Student*, *Course*, and *Professor*, and two relationship tables; *Registration* with foreign key pointers to the *Student* and *Course* tables whose tuples

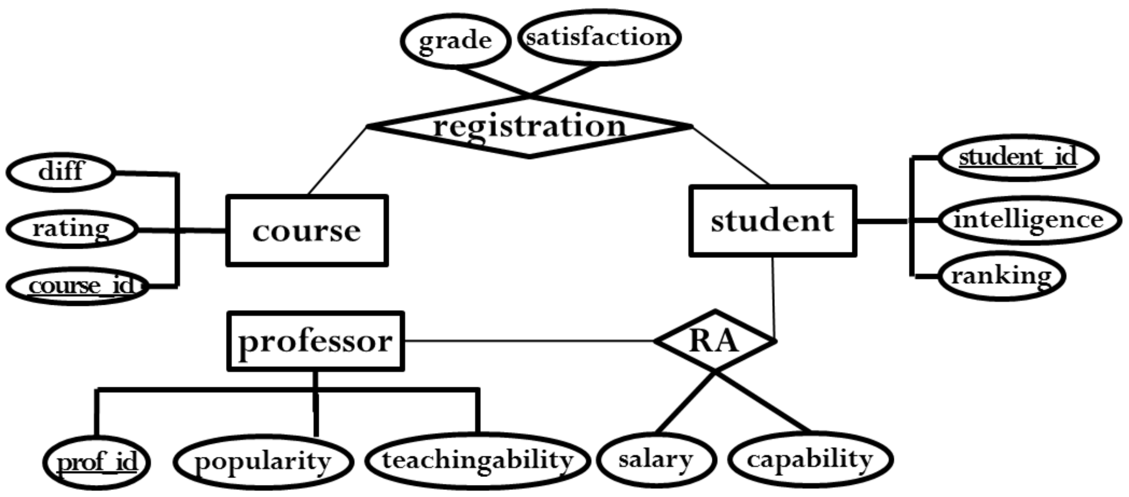


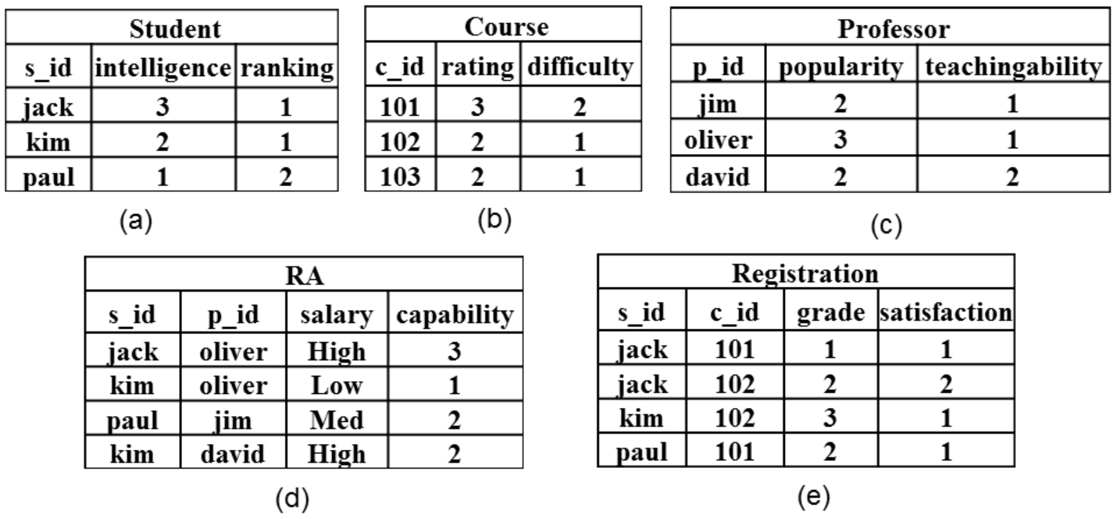
Figure 1.2: A relational ER Design. Registration and RA are many-to-many relationships.

indicate which students have registered in which courses and *RA* with foreign key pointers to the *Student* and *Professor* tables whose tuples indicate the RAship of students for professors. Relationships refer to their related objects using *reference slots*. Each table in the relational database is seen as a class that has some descriptive attributes. A **database instance** specifies the tuples contained in the tables of a given database schema. Figure 1.3 is an instance for the schema in Figure 1.2.

1.2 Log-linear Template Models for Relational Data

As illustrated in Figure 1.4, statistical relational learning (known as SRL) is a recent growing field. It is the intersection of artificial intelligence, machine learning and database systems. SRL extends traditional machine learning from single-table to multiple inter-related tables. It aims to provide integrated statistical analysis of the complex data sources. The extensive survey by Kimmig *et al.* [45] shows that SRL models can be viewed as log-linear models based on "parametrized factor" as follows. Par-factor stands for "parametrized factor". A par-factor represents an interaction among parametrized random variables, or par-RVs for short. In this section, I review the background knowledge from log-linear models to motivate my thesis work. In section 1.2.1, I show the Parametrized Bayesian Network as one example among the SRL models and introduce the structure learning algorithm in section 1.2.2. In the following chapters of my thesis, I consider "Bayesian Network" and "Bayes Net" the same term, and I use them interchangeablely.

The concept of a parametrized random variable (par-RV) marries logical-relational notation to a statistical concept. I employ the following notation for par-RVs [45, 2.2.5]. Constants are expressed in lower-case, e.g. *joe*, and are used to represent entities. A type is associated with each entity, e.g. *joe* is a person. A first-order variable is also typed, e.g.



Student		
s_id	intelligence	ranking
jack	3	1
kim	2	1
paul	1	2

(a)

Course		
c_id	rating	difficulty
101	3	2
102	2	1
103	2	1

(b)

Professor		
p_id	popularity	teachingability
jim	2	1
oliver	3	1
david	2	2

(c)

RA			
s_id	p_id	salary	capability
jack	oliver	High	3
kim	oliver	Low	1
paul	jim	Med	2
kim	david	High	2

(d)

Registration			
s_id	c_id	grade	satisfaction
jack	101	1	1
jack	102	2	2
kim	102	3	1
paul	101	2	1

(e)

Figure 1.3: Database Instance based on Figure 1.2.

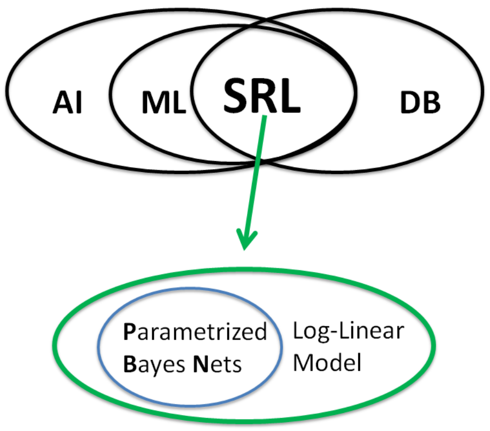


Figure 1.4: SRL is the intersection of artificial intelligence (AI), machine learning (ML) and database systems (DB). Parametrized Bayesian Network is one the SRL models that can be viewed as log-linear models.

Person denotes some member of the class of persons. A functor maps a tuples of entities to a value. I assume that the range of possible values is finite. An *atom* is an expression of the form $r(\tau_1, \ldots, \tau_a)$ where each τ_i is either a constant or a first-order variable. If all of $\tau_1, \ldots, \tau_a$ are constants, $r(\tau_1, \ldots, \tau_a)$ is a *ground atom* or random variable (RV), otherwise a *first-order atom* or a **par-RV**. A par-RV is instantiated to an RV by grounding, i.e. substituting a constant of the appropriate domain for each first-order variable.

A **par-factor** is a pair $\Phi = (\mathsf{A}, \Psi)$, where A is a set of par-RVs, and Ψ is a function from the values of the par-RVs to the non-negative real numbers.[1] Intuitively, a grounding of a par-factor represents a set of ground random variables that interact with each other locally. SRL models use *parameter tying*, meaning that if two groundings of the same par-factor are assigned the same values, they return the same factor value. A set of parfactors $\mathcal{F}$ defines a joint probability distribution over the ground par-RVs as follows. Let $\mathcal{I}(\Phi_i)$ denote the set of *all* ground par-RVs in par-factor Φ_i. Let $\mathbf{x}$ be a joint assignment of values to all ground random variables. Notice that this assignment determines the values of all ground atoms. An assignment $\mathbf{X} = \mathbf{x}$ is therefore *equivalent to a single database instance.* The probability of a database instance is given by the log-linear equation [45, Eq.7]:

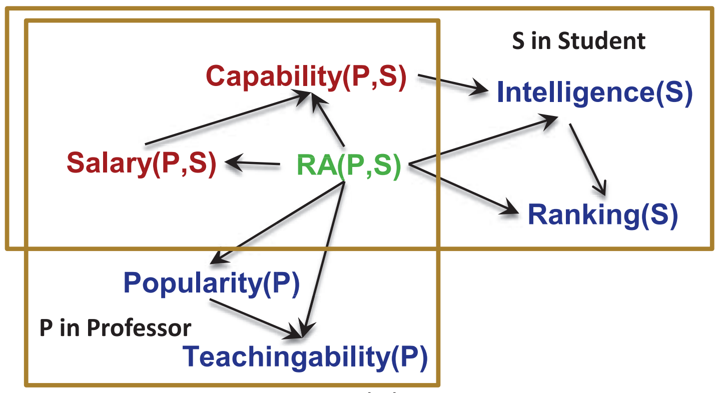
$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\Phi_i \in \mathcal{F}} \prod_{\mathbf{A} \in \mathcal{I}(\Phi_i)} \Psi_i(\mathbf{x_A}) \tag{1.1}$$

where $\mathbf{x_A}$ represents the values of those variables in $\mathbf{A}$ that are necessary to compute Ψ_i. Equation 1.1 can be evaluated, without enumerating the ground par-factors, as follows. (1) For each par-factor, for each possible assignment of values, find the number of ground factors with that assignment of values. (2) Raise the factor value for that assignment to the number of instantiating factors. (3) Multiply the exponentiated factor values together. The number (2) of ground factors with the same assignment of values is known as a **sufficient statistic**.

1.2.1 Examples: Parametrized Bayesian Network

SRL has developed a number of formalisms for describing par-factors [45]. First-order probabilistic graphical models are popular both within SRL and the database community [45, 91]. The model structure is defined by edges connecting par-RVs. For instance, a **parametrized Bayesian network structure** is a directed acyclic graph (DAG) whose nodes are par-RVs. Figure 1.5 shows a Bayesian network for a University domain. The schema for the university domain is given in Figure 1.2. This schema features only one relationship for simplicity; FactorBase learns a model for any number of relationships. While I describe FactorBase abstractly in terms of par-factors, for concreteness I illustrate

[1] A par-factor can also include constraints on possible groundings.



(a)

Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1.00
4	T	high	0.45
5	T	high	0.36
3	T	high	0.18
3	T	low	0.20
2	T	low	0.40
1	T	low	0.40
2	T	med	0.22
3	T	med	0.44
1	T	med	0.33

(b)

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	n/a	F	n/a
5	4	T	high
4	5	T	high
2	3	T	high
1	3	T	low
2	2	T	low
2	1	T	low
2	2	T	med
4	3	T	med
3	1	T	med

(c)

Figure 1.5: (a) Bayesian network for the University domain. I omit the *Registration* relationship for simplicity. The network was learned from the University dataset [67]. (b) Conditional Probability Table $Capability(P, \mathcal{S})\_CPT$, for the node $Capability(P, \mathcal{S})$. Only value combinations that occur in the data are shown. This is an example of a factor table. (c) Contingency Table $Capability(P, \mathcal{S})\_CT$ for the node $Capability(P, \mathcal{S})$ and its parents. Both CP and CT tables are stored in an RDBMS.

it using Bayesian networks. The system takes as input a database instance like that shown in Figure 1.3, and produces as output a graphical model like that shown in Figure 1.5 (a).

A par-factor in a Bayesian network is associated with a **family** of nodes [45, Sec.2.2.1]. A family of nodes comprises a child node and all of its parents. For example, in the BN of Figure 1.5, one of the par-factors is associated with the par-RV set

$$\mathsf{A} = \{\mathit{Capability}(P, \mathcal{S}), \mathit{Salary}(P, \mathcal{S}), \mathit{RA}(P, \mathcal{S})\}.$$

For the database instance of Figure 1.3, there are $3 \times 3 = 9$ possible factors associated with this par-RV, corresponding to the Cartesian product of 3 professors and 3 students. The value of the factor ϕ is a function from an assignment of family node values to a non-negative real number. **In a Bayesian network, the factor value represents the conditional probability of the child node value given its parent node values.** These conditional probabilities are typically stored in a table as shown in Figure 1.5(b). This table represents therefore the function ϕ associated with the family par-factor. Assuming that all par-RVs have finite domains, a factor can always be represented by a **factor table** of the form Figure 1.5(b): there is a column for each par-RV in the factor, each row specifies a joint assignment of values to a par-RV, and the factor column gives the value of the factor for that assignment (cf. [45, Sec.2.2.1]).

The sufficient statistics for the $\mathit{Capability}(P, \mathcal{S})$ family can be represented in a contingency table as shown in Figure 1.5(c). For example, the first row of the contingency table indicates that the conjunction $\mathit{Capability}(P, \mathcal{S}) = n/a, \mathit{Salary}(P, \mathcal{S}) = n/a, \mathit{RA}(P, \mathcal{S}) = \mathrm{F}$ is instantiated 203 times in the University database (publicly available at [67]). This means that for 203 professor-student pairs, the professor did not employ the student as an RA (and therefore the salary and capability of this RA relationship is undefined or n/a).

1.2.2 SRL Structure Learning

Algorithm 1 shows the generic format of a statistical-relational structure learning algorithm (adapted from [45]). The instantiation of procedures in lines 2, 3, 5 and 8 determines the exact behavior of a specific learning algorithm. The structure algorithm carries out a local search in the hypothesis space of graphical relational models. A set of candidates is generated based on the current model (line 3), typically using a search heuristic. For each candidate model, parameter values are estimated that maximize a model selection score function chosen by the user (line 5). A model selection score is computed for each model given the parameter values, and the best-scoring candidate model is selected (line 7).

Algorithm 1: Structure learning algorithm (instantiation of procedures in lines 2, 3, 5 and 8 determines exact behaviour)

```
Input: Hypothesis space $\mathcal{H}$ (describing graphical models), training data $\mathcal{D}$
       (assignments to random variables), scoring function score $(\cdot, \mathcal{D})$
Output: A graph structure $G$ representing par-factors.
1: $G \leftarrow \emptyset$
2: while CONTINUE($G$, $\mathcal{H}$, score $(\cdot, \mathcal{D})$ ) do
3:     $\mathcal{R} \leftarrow$ REFINECANDIDATES($G, \mathcal{H}$)
4:     for each $R \in \mathcal{R}$ do
5:         $R \leftarrow$ LEARNPARAMETERS($R$,score $(\cdot, \mathcal{D})$)
6:     end for
7:     $G \leftarrow \text{argmax}_{G' \in \mathcal{R} \cup \{G\}}$ score($G'$, $\mathcal{D}$)
8: end while
9: return $G$
```

1.3 Related Work

I design the FactorBase system which combines advanced analytics from multi-relational or statistical-relational machine learning (SRL) with database systems. This new SQL-based framework uses the RDBMS to manage the learned model as first-class citizen and the relational data so that one can leverage the RDBMS to support the multi-relational model discovery. In chapter 2, I will discuss the system design details and show how it supports model discovery algorithms that follow Algorithm 1. Figure 2.1 outlines the system components and dependencies among them. The design space for combining machine learning with data management systems offers a number of possibilities, several of which have been explored in previous and ongoing research. I selectively review the work most relevant to my research and the FactorBase system. Figure 1.6 provides a graphical illustration of where FactorBase is located with respect to other works.

1.3.1 Single-Table Machine Learning

Most machine learning systems, such as Weka or R, support learning from a single table or data matrix only. The single-table representation is appropriate when the data points represent a homogeneous class of entities with similar attributes, where the attributes of one entity are independent of those of others [45]. The only way a single-table system can be applied to multi-relational data is after a preprocessing step where multiple interrelated tables are converted to a single data table. When the learning task is classification, such preprocessing is often called propositionalization [45]. This “flattening” of the relational structure typically involves information loss.

RDBMS Learning Leveraging RDBMS capabilities through SQL programming has been explored for a variety of single-table learning tasks. This is the unifying idea of the recent

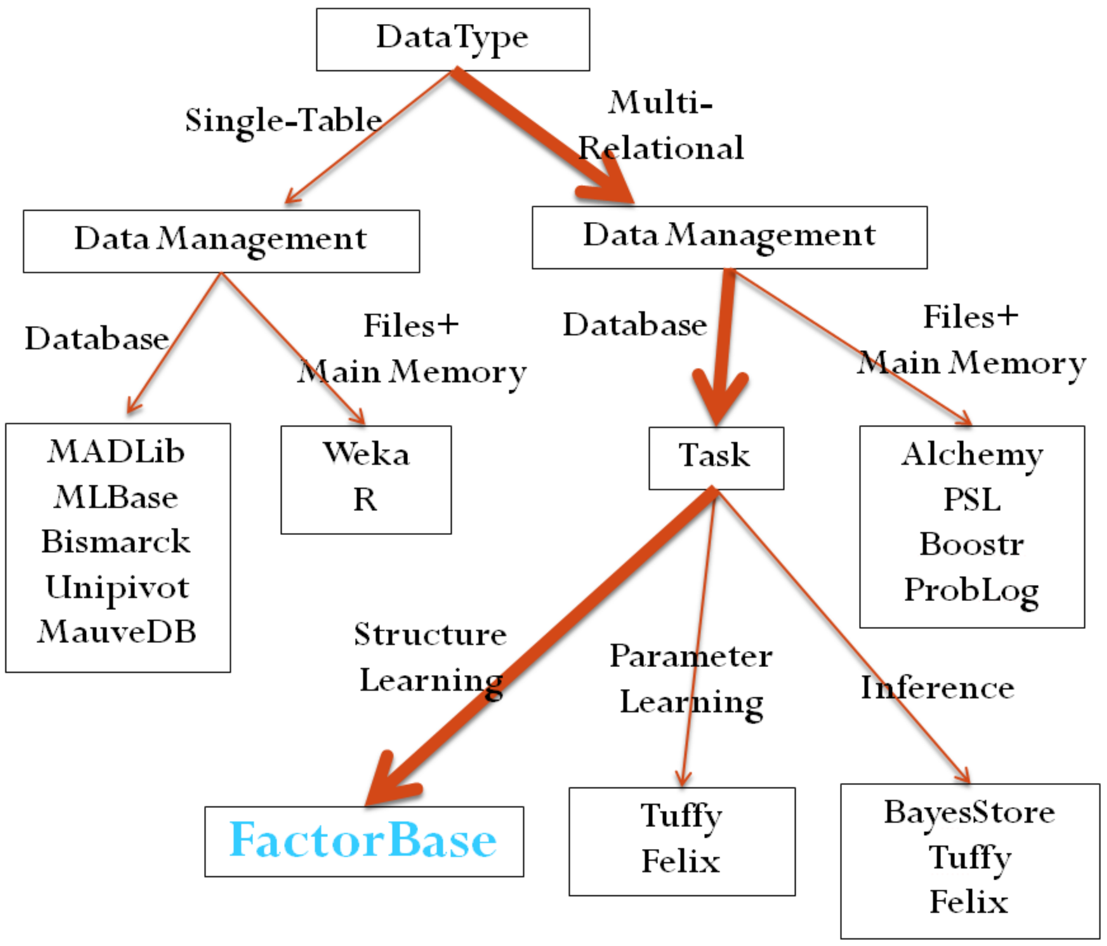


Figure 1.6: A tree structure for related work in the design space of machine learning × data management.

MADLib framework [34]. An advantage of the MADLib approach that is shared by FactorBase is that in-database processing avoids exporting the data from the input database. The Apache Spark [10] framework includes MLBase and SparkSQL that provide support for distributed processing, SQL, and automatic refinement of machine learning algorithms and models [48]. Other RDBMS applications include gathering sufficient statistics [26], and convex optimization [18]. The MauveDB system [14] emphasizes the importance of several RDBMS features for combining statistical analysis with databases. As in FactorBase, this includes storing models and associated parameters as objects in their own right, and using the view mechanism to update statistical objects as the data change. A difference is that MauveDB presents model-based views of the *data* to the user, whereas FactorBase presents views of the *models* to machine learning applications.

RDBMS Inference Wong *et al.* applied SQL operators such as the natural join to perform log-linear inference with a single-table graphical model [93] stored in an RDBMS. Monte Carlo methods have also been implemented with an RDBMS to perform inference with uncertain data [38, 92]. The MCDB system [38] stores parameters in database tables like FactorBase.

Parameter Server (PS) paradigm For scaling up machine learning to large datasets and complex problems, distributed optimization and inference mechanism have been increasingly popular and a variety of approaches have been proposed [5]. Several related systems, including open source projects have been implemented at Google [13], Yahoo [2] and Petuum [35]. The PS framework (globally shares the parameters by scheduling the workload of storing and updating large volume of parameters to distributed servers) was first introduced by Smola et al [83] and it has proliferated in both industrial and academic research communities [51]. There are also some critical issues arised as it moves forward [95]. The PS paradigm could be considered as general-purpose framework, it focuses on improving the "systems" code (notably, communication and synchronization protocols for model state) to increase the efficiency of machine learning algorithms [11]. In contrast, the FactorBase System combines both systems code improvements and algorithmic learning improvements tailor-made for log-linear model of multi-relational data. The PS paradigm can be used to perform inference on the log-linear template model once it is learned, and potentially it could be integrated into Model Manager as introduced in section 2.4.

1.3.2 Multi-Relational Learning

For overviews of multi-relational learning please see [23, 15, 45]. Most implemented systems, such as Aleph and Alchemy, use a logic-based representation of data derived from Prolog facts, that originated in the Inductive Logic Programming community [17].

RDBMS Learning The ClowdFlows system [50] allows a user to specify a MySQL database as a data source, then converts the MySQL data to a single-table representation using propositionalization. Singh and Graepel [82] present an algorithm that analyzes the relational database system catalog to generate a set of nodes and a Bayesian network structure. This approach utilizes SQL constructs as a data description language in a way that is similar to the schema analyzer which will be introduced in section 2.2. Differences include the following. (1) The Bayesian network structure is fixed and based on latent variables, rather than learned for observable variables only, as in the case study. (2) The RDBMS is not used to support learning after random variables have been extracted from the schema.

Qian *et al.* [70] discuss work related to the contingency table problem and introduce contingency table algebra. Their paper focuses on a Virtual Join algorithm for computing sufficient statistics that involve negated relationships. They do not discuss integrating contingency tables with other structured objects for multi-relational learning.

RDBMS Inference Database researchers have developed powerful probabilistic inference algorithms for multi-relational models. The BayesStore system [91] introduced the principle of treating all statistical objects as first-class citizens in a relational database as FactorBase does. The Tuffy system [61] achieves highly reliable and scalable inference for Markov Logic Networks (MLNs) with an RDBMS. It leverages inference capabilities to perform MLN parameter learning. RDBMS support for local search parameter estimation procedures, rather than closed-form maximum-likelihood estimation, has also been explored [18, 61, 60].

1.4 Contributions

Within this section, I highlight the contributions of my thesis evaluated with respect to scalability, quality and other performance criterion (e.g. preprocessing time, usefulness) as follows.

FactorBase In chapter 2, I describe the FactorBase system which combines advanced analytics from multi-relational or statistical-relational machine learning (SRL) with database systems. This new SQL-based framework pushes the multi-relational model discovery into a relational database management system. Empirical evaluation shows significant scalability advantages from utilizing the RDBMS capabilities: Both structure and parameter learning scale well to millions of data records, beyond what previous multi-relational learning systems can achieve. This work has been reported in [68, 69].

Cross-table Sufficient Statistics For most of the learning task, the scalability bottleneck is to compute the cross-table sufficient statistics. In chapter 3, I propose and implemente in FactorBase a new Virtual Join algorithm to tackle this problem. Empirical evaluation demonstrates the scalability of the algorithm and illustrates how access to sufficient statistics for both positive and negative relationships enhances feature selection and rule mining. This provides the evidence of the statistical quality and usefulness of the sufficient statistics computed by the Virtual Join algorithm. This work has been reported in [70, 71].

Generative modelling with Link Uncertainty I incorporate the FactorBase system to tackling challenge applications. In chapter 4, I utilize FactorBase system as a precounting approach on gathering sufficient statistics to extend the state-of-the-art learning algorithm for Bayes net generative modelling of multiple links considering link uncertainty. The extended algorithm supports to capture simultaneous correlations between link types, link features, and attributes of nodes. This type of model supports queries like, “What fraction of the grades are awarded to highly intelligent students?” The empirical study illustrates the usefulness of the proposed model with respect to find additional correlations among different types of links. This work has been reported in [66, 71].

Instance Level Discriminative Learning Bayes nets and Dependency networks (DNs) are two major classes of Statistical Relational Learning (SRL). In chapter 5, I propose a fast scalable hybrid approach for instance level discriminative learning of DNs. This type of model supports queries like, “What is the probability of *Jack* being a highly intelligent student given the grades of his registered courses?” The experiment shows that the predictive power with respect to statistical accuracy of the model learned with the hybrid learning algorithm is competitive with those from state-of-the-art function gradient boosting methods but scales substantially better than the boosting methods. This work has been reported in [80, 81].

1.5 Thesis Organization

The organization of this thesis is as follows. First, in Chapter 1, I review the background from multi-relational data representation of log-linear models and structure learning to motivate my thesis work, followed by related works in the field. And then I describe the design principles and key components of FactorBase system in Chapter 2. The new Virtual Join algorithm for cross-table sufficient statistics computation to tackle the learning bottleneck is introduced in Chapter 3. Following in Chapter 4 and Chapter 5, I conduct the usefulness evaluation of the system with generative and discriminative modelling, respectively. Finally, Chapter 6 concludes this thesis, identifies limitations of the proposed system and methods, and discusses possible directions for future work.

Chapter 2

FactorBase: a new SQL-based framework for multi-relational model discovery

The system described in this chapter combines advanced analytics from multi-relational or *statistical-relational* machine learning (SRL) with database systems. The power of combining machine learning with database systems has been demonstrated in several systems[34, 48, 14]. The novel contribution of FactorBase is supporting machine learning for *multi-relational* data, rather than for traditional learning where the data are represented in a *single* table or data matrix. I discuss new challenges raised by multi-relational model learning compared to single-table learning, and how FactorBase solves them using the resources of SQL (Structured Query Language). The name FactorBase indicates that the system supports learning factors that define a log-linear multi-relational model [45]. Supported new database services include constructing, storing, and transforming complex statistical objects, such as factor-tables, cross-table sufficient statistics, parameter estimates, and model selection scores. My argument is that relational algebra can play the same role for statistical-relational learning that linear algebra does for traditional single-table machine learning: a unified language for both representing and computing with statistical-relational objects.

2.1 Introduction

FactorBase is the first system that leverages relational query processing for learning a multi-relational log-linear graphical model. Whereas the in-database design philosophy has been previously used for multi-relational inference, I am the first to adapt it for multi-relational model structure learning. The experiments show that FactorBase pushes the scalability boundary: Learning scales to databases with over 10^6 records, compared to less than 10^5

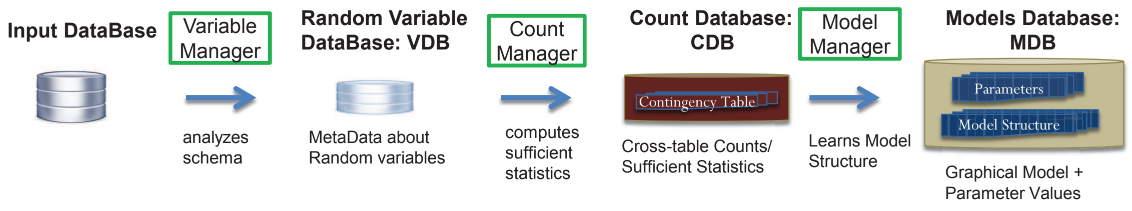


Figure 2.1: System Flow. All statistical objects are stored as first-class citizens in a DBMS. Objects on the left of an arrow are utilized for constructing objects on the right. Statistical objects are constructed and managed by different modules, shown as boxes.

for previous systems. At the same time it is able to discover more complex cross-table correlations than previous SRL systems. I report experiments that focus on two key services for an SRL client: (1) Computing and caching sufficient statistics, (2) computing model predictions on test instances. For the largest benchmark database, the system handles 15M sufficient statistics. SQL facilitates block-prediction for a set of test instances, which leads to a 10 to 100-fold speedup compared to a simple loop over test instances.

Pushing the graphical model inside the database allows us to *use SQL as a high-level scripting language for SRL*, with the following advantages.

1. Extensibility and modularity, which support rapid prototyping. SRL algorithm development can focus on statistical issues and rely on a RDBMS for data access and model management.

2. Increased scalability, in terms of both the size and the complexity of the statistical objects that can be handled.

3. Generality and portability: standardized database operations support "out-of-the-box" learning with a minimal need for user configuration.

Chapter Organization I first provide an overview of the system components and the work flow in Figure 2.1. And for each component, I describe how the component is constructed and managed inside an RDBMS using SQL scripts and the SQL view mechanism. I show how the system manages sufficient statistics and test instance predictions in a block access way. The evaluation section demonstrates the scalability advantages of in-database processing.

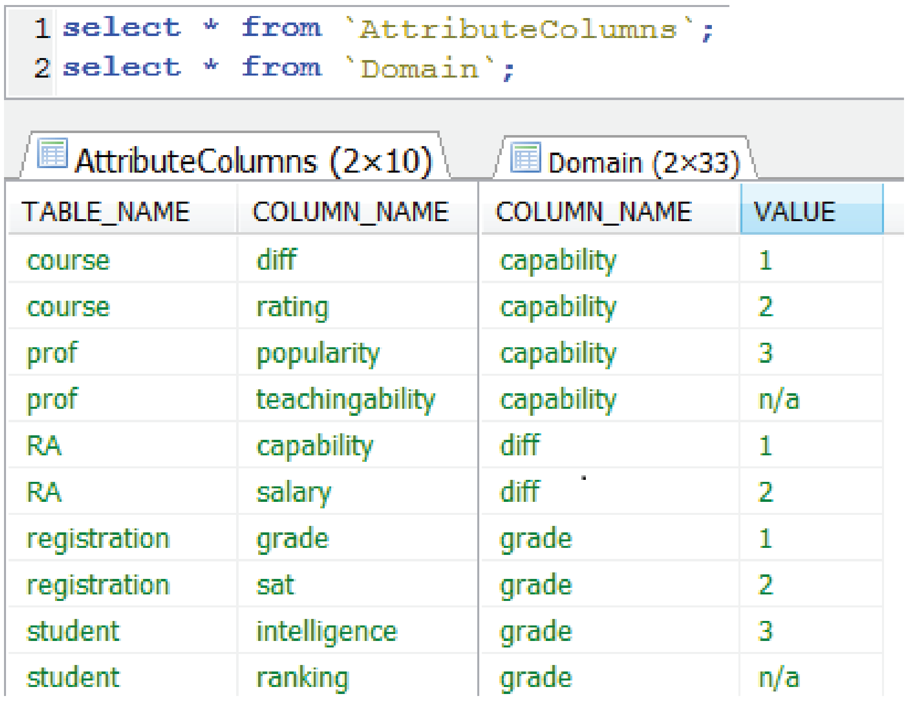
2.2 The Random Variable Database

Statistical-relational learning requires various metadata about the par-RVs in the model. These include the following.

Domain the set of possible values of the par-RV.

Types Pointers to the first-order variables in the par-RV.

Data Link Pointers to the table and/or column in the input database associated with the par-RV.



```
select * from `AttributeColumns`;
select * from `Domain`;
```

AttributeColumns (2×10)

TABLE_NAME	COLUMN_NAME
course	diff
course	rating
prof	popularity
prof	teachingability
RA	capability
RA	salary
registration	grade
registration	sat
student	intelligence
student	ranking

Domain (2×33)

COLUMN_NAME	VALUE
capability	1
capability	2
capability	3
capability	n/a
diff	1
diff	2
grade	1
grade	2
grade	3
grade	n/a

Figure 2.2: The metadata about attributes represented in *VDB* database tables. Left: The table *AttributeColumns* specifies which tables and columns contain the functor values observed in the data. The column name is also the functor ID. Right: The table *Domain* lists the domain for each functor.

The metadata must be machine-readable. Following the in-database design philosophy, I store the metadata in tables so that an SRL algorithm can query it using SQL. The *schema analyzer* uses an SQL script that queries key constraints in the system catalog database and *automatically* converts them into metadata stored in the random variable database VDB. In contrast, existing SRL systems require users to specify information about par-RVs and associated types. Thus FactorBase utilizes the data description resources of SQL to facilitate the "setup task" for relational learning [90]. Here, I illustrate the general principles with the ER diagram of the University domain (Figure 1.2).[1]

The translation of an ER diagram into a set of functors converts each element of the diagram into a functor, except for entity sets and key fields [32]. Table 2.1 illustrates this

[1]A full description is available [79].

Table 2.1: Translation from an ER Diagram to Par-RVs

ER Diagram	Example	par-RV equivalent
Entity Set	Student, Course	$\mathcal{S}, \mathbb{C}$
Relationship Set	RA	RA(P,$\mathcal{S}$)
Entity Attributes	intelligence, ranking	Intelligence($\mathcal{S}$), Ranking($\mathcal{S}$)
Relationship Attributes	capability, salary	Capability($P, \mathcal{S}$), Salary($P, \mathcal{S}$)

translation. In terms of database tables, attribute par-RVs correspond to *columns*. Relationship par-RVs correspond to *tables*, not columns. Including a relationship par-RV in a statistical model allows the model to represent uncertainty about whether or not a relationship exists between two entities [45]. The values of descriptive attributes of relationships are undefined for entities that are not related. I represent this by introducing a new constant n/a in the domain of a relationship attribute [55]; see Figure 2.2 (right). Table 2.2 shows the schema for some of the tables that store metadata for each relationship par-RV, as follows. par-RV and FO-Var are custom types.

Relationship The associated input data table.

Relationship_Attributes Descriptive attributes associated with the relationship and with the entities involved.

Relationship_FOVariables The first-order variables contained in each relationship par-RV.[2]

Table 2.2: Selected Tables In the Variable Database Schema.

Table Name	Column Names
Relationship	RVarID: par-RV, TABLE_NAME: string
Relationship_Attributes	RVarID: par-RV, AVarID: par-RV, FO-ID: FO-Var
Relationship_FOvariables	RVarID: par-RV, FO-ID: FO-Var, TABLE_NAME: string

While I have described constructing the variable database for an ER model, different structured data models can be represented by an appropriate first-order logic vocabulary [45], that is, an appropriate choice of functors. For example, in a star schema, facts can be represented in the form $f(\mathbb{D}_1, \ldots, \mathbb{D}_k)$, where the first-order variable $\mathbb{D}_i$ ranges over the primary key of dimension table i. Attributes of dimension i can be represented by a unary functor $a(\mathbb{D}_i)$. FactorBase can perform structure learning for different data models after the corresponding data format has been translated into the *VDB* format.

[2]The schema assumes that all relationships are binary.

2.3 The Count Manager

The **count database** *CDB* stores a set of *contingency tables*. Contingency tables represent sufficient statistics as follows [57, 70]. Consider a fixed list of par-RVs. A **query** is a set of ($variable = value$) pairs where each value is of a valid type for the variable. The **result set** of a query in a database $\mathcal{D}$ is the set of instantiations of the logical variables such that the query evaluates as true in $\mathcal{D}$. For example, in the database of Figure 1.3 the result set for the query $RA(P, \mathcal{S}) = \mathrm{T}$, $Capability(P, \mathcal{S}) = 3$, $Salary(P, \mathcal{S}) = high$ is the singleton $\{\langle jack, oliver\rangle\}$. The **count** of a query is the cardinality of its result set.

Every set of par-RVs $\mathbf{V} \equiv \{V_1, \ldots, V_n\}$ has an associated **contingency table** (CT) denoted by $CT(\mathbf{V})$. This is a table with a row for each of the possible assignments of values to the variables in $\mathbf{V}$, and a special integer column called *count*. The value of the *count* column in a row corresponding to $V_1 = v_1, \ldots, V_n = v_n$ records the count of the corresponding query. Figure 1.5 (b) shows a contingency table for the par-RVs $RA(P, \mathcal{S})$, $Capability(P, \mathcal{S})$, $Salary(P, \mathcal{S})$. The value of a relationship attribute is undefined for entities that are not related. Following [72], I indicate this by writing $capability(P, \mathcal{S}) = n/a$ for a reserved constant n/a. The assertion $capability(P, \mathcal{S}) = \mathrm{n/a}$ is therefore equivalent to the assertion that $RA(P, \mathcal{S}) = \mathrm{F}$. A **conditional contingency table**, written

$$ct(V_1, \ldots, V_k | V_{k+1} = v_{k+1}, \ldots, V_{k+m} = v_{k+m})$$

is the contingency table whose column headers are $V_1, \ldots, V_k$ and whose rows comprise the subset that match the conditions to the right of the | symbol. I assume that contingency tables omit rows with count 0.

The **contingency table problem** is to compute a contingency table for par-RVs $\mathbf{V}$ and an input database $\mathcal{D}$. This is one of the key issues that most of the learning time is taken up. To improve scalability, computing sufficient statistics needs to be feasible for cross product sizes in the millions or more. I will introduce one Virtual Join method in chapter 3, which is a good solution that computes sufficient statistics without materializing table joins.

2.4 The Model Manager

The Model Manager provides two key services for statistical-relational structure learning:

1. Estimating and storing parameter values (line 5 of Algorithm 1).

2. Computing one or more model selection scores (line 7 of Algorithm 1).

FactorBase uses a *store+score* design for these services, which is illustrated in Figure 2.3. A **model structure table** represents a candidate model. When a candidate model structure is inserted, a view uses the sufficient statistics from a contingency table to compute a table of parameter values. Another view uses the parameter values and sufficient statistics together to compute the score for the candidate model. In this store+score design, the RDBMS provides model selection scores as a service to the SRL client. In many industrial scenarios (e.g. topic modeling: given a large collection of documents, infer the topics contained in each document), one need to deal with very big model containing millions to billions of parameters, the PS paradigm is a very promising approach which could be integrated into the Model Manager [51].

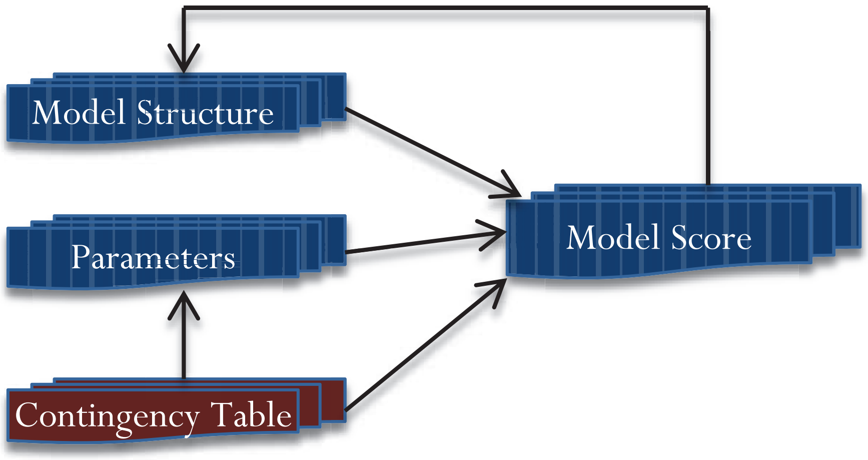
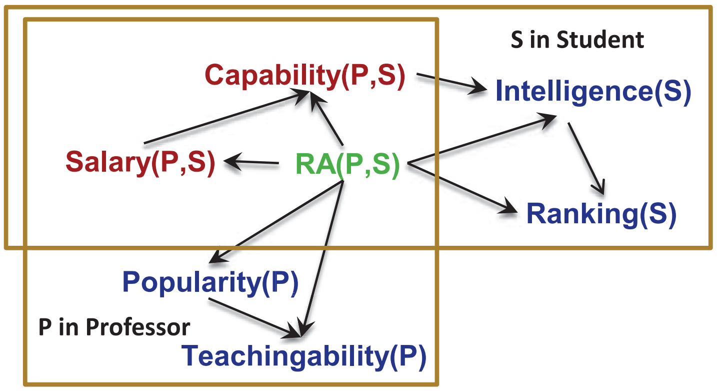


Figure 2.3: Dependencies Among Key Components of the Model Manager.

2.4.1 The *MDB* Schema

The relational schema for the Models Database is shown in Table 2.3. The @par-RVID@ parameter refers to the ID of a par-RV, for instance $\mathit{Capability}(P, \mathcal{S})$. The model manager stores a set of factor tables (cf. Section 1.2.1). In a graphical model, each factor is defined by the local topology of the model template graph. For concreteness, I illustrate how factor tables can be represented for Bayesian networks. The graph structure can be stored straightforwardly in a database table *BayesNet* whose columns are *child* and *parent*. The table entries are the IDs of par-RVs. An entry such as $(\mathit{Capability}(P, \mathcal{S}), \mathit{Salary}(P, \mathcal{S}))$ means that $\mathit{Capability}(P, \mathcal{S})$ is a child of $\mathit{Salary}(P, \mathcal{S})$. For each node, the *MDB* manages a conditional probability table. This is a factor table that represents the factor associated with the node's family (see Figure 2.4(b)).

In a Bayesian network, model selection scores are decomposable. This means that there is a local score associated with each family, such that the total score for the BN model is



(a)

Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1.00
4	T	high	0.45
5	T	high	0.36
3	T	high	0.18
3	T	low	0.20
2	T	low	0.40
1	T	low	0.40
2	T	med	0.22
3	T	med	0.44
1	T	med	0.33

(b)

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	n/a	F	n/a
5	4	T	high
4	5	T	high
2	3	T	high
1	3	T	low
2	2	T	low
2	1	T	low
2	2	T	med
4	3	T	med
3	1	T	med

(c)

Figure 2.4: (a) Bayesian network for the University domain. I omit the *Registered* relationship for simplicity. The network was learned from the University dataset [67]. (b) Conditional Probability table $Capability(P, \mathcal{S})\_CPT$, for the node $Capability(P, \mathcal{S})$. Only value combinations that occur in the data are shown. This is an example of a factor table. (c) Contingency Table $Capability(P, \mathcal{S})\_CT$ for the node $Capability(P, \mathcal{S})$ and its parents. Both CP and CT tables are stored in an RDBMS.

the sum of the local scores. For each family, the local score is stored in the *Scores* table indexed by the family's child node.

2.4.2 Parameter Manager

Deriving predictions from a model requires estimating values for its parameters. Maximizing the data likelihood is the basic parameter estimation method for Bayesian networks. The maximum likelihood estimates equal the observed frequency of a child value given its parent values.

Table 2.3: The main tables in the Models Database *MDB*. For a Bayesian network, the *MDB* stores its structure, parameter estimates, and model selection scores.

BayesNet(child:par-RV,parent:par-RV)
@par-RVID@_CPT(@par-RVID@:par-RV,parent$_1$:par-RV,. . . , parent$_k$:par-RV,cp:real)
Scores(child:par-RV,loglikelihood:real,#par:int,aic:real)

SQL Implementation of Conditional Probability Tables With Natural Join. Given the sufficient statistics in a contingency table, a conditional probability table containing the maximum likelihood estimates can be computed by aggregation using SQL. As shown in Figure 2.4(b)[3], the conditional probability table $Capability(P, \mathcal{S})\_CPT$, for the node $Capability(P, \mathcal{S})$ is computed as below.

```
SELECT count/temp.parent_count as CP,
capability(P,S), RA(P,S), salary(P,S)
FROM capability(P,S)_CT
NATURAL JOIN
(SELECT sum(Count) as parent_count,
RA(P,S), salary(P,S)
FROM capability(P,S)_CT
GROUP BY  RA(P,S), salary(P,S) ) as temp
```

2.4.3 Model Score Computation

A typical model selection approach is to maximize the likelihood of the data, balanced by a penalty term. For instance, the Akaike Information Criterion (AIC) is defined as follows

$$AIC(G, \mathcal{D}) \equiv ln(P_{\widehat{G}}(\mathcal{D})) - par(G)$$

where $\widehat{G}$ is the BN G with its parameters instantiated to be the maximum likelihood estimates given the database $\mathcal{D}$, and $par(G)$ is the number of free parameters in the structure G. The number of free parameters for a node is the product of (the possible values for the parent nodes) $\times$ (the number of the possible values for the child node -1). Given the likelihood and the number of parameters, the AIC column is computed as $AIC = loglikelihood - par$. Model selection scores other than AIC can be computed in a similar way given the model likelihood and number of parameters.

Parameter Number Computation To determine the number of parameters of the child node @parVar-ID@, the number of possible child and parent values can be found from the *VDB.Domain* table in the Random Variable Database.

Likelihood Computation As explained in Section 1.2, the log-likelihood can be computed by multiplying the instantiation counts of a factor by its value. Assuming that instantiation counts are represented in a contingency table and factor values in a factor table, this multiplication can be elegantly performed using the Natural Join operator. For instance, the log-likelihood score associated with the $Capability(P, \mathcal{S})$ family is given by the SQL query below.

[3]Same as Figure 1.5.

```
SELECT Capability(P,S),  SUM(CPT.cp * CT.count) AS loglikelihood
FROM MDB.Capability(P,S)_CPT as CPT
NATURAL JOIN CDB.Capability(P,S)_CT as CT
```

The aggregate computation in this short query illustrates how well SQL constructs support complex computations with structured objects. This completes the description of how the modules of FactorBase are implemented using SQL. I next show how these modules support a key learning task: computing the predictions of an SRL model on a test instance.

2.5 Test Set Predictions

Computing probabilities over the label of a test instance is important for several tasks.

1. Classifying the test instance, which is one of the main applications of a machine learning system for end users.
2. Comparing the class labels predicted against true class labels is a key step in several approaches to model scoring [45].
3. Evaluating the accuracy of a machine learning algorithm by the train-and-test paradigm, where the system is provided a training set for learning and then one test its predictions on unseen test cases.

I first discuss how to compute a prediction for a single test case, then how to compute an overall prediction score for a set of test cases. Recall that log-linear equation 1.1 is as follows [45, Eq.7]:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\Phi_i \in \mathcal{F}} \prod_{\mathbf{A} \in \mathcal{I}(\Phi_i)} \Psi_i(\mathbf{x}_{\mathbf{A}})$$

where $\mathbf{x}_{\mathbf{A}}$ represents the values of those variables in $\mathbf{A}$ that are necessary to compute Ψ_i.

Class probabilities can be derived from above Equation as follows. Let Y denote a ground par-RV to be classified, which I refer to as the **target variable**. For example, a ground atom may be *Intelligence*(*jack*). In this example, I refer to jack as the **target entity**. Write $\mathbf{X}_{-Y}$ for a database instance that specifies the values of all ground par-RVs, except for the target, which are used to predict the target node. Let $[\mathbf{X}_{-Y}, y]$ denote the completed database instance where the target node is assigned value y. The log-linear model uses the likelihood $P([\mathbf{X}_{-Y}, y])$ as the joint score of the label and the predictive features. The conditional probability is proportional to this score:

$$P(y|\mathbf{X}_{-\mathbf{Y}}) \propto P([\mathbf{X}_{-Y}, y]) \tag{2.1}$$

where the joint distribution on the right-hand side is defined by Equation 1.1, and the scores of the possible class labels need to be normalized to define conditional probabilities.

Table 2.4: SQL queries for computing target contingency tables supporting test set prediction. <Attribute-List> and <Key-Equality-List> are as in Figure 3.2.

Access	SELECT	WHERE	GROUP BY
Single	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List> AND S.s_id = jack	<Attribute-List>
Block	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List>	<Attribute-List>, **S.sid**

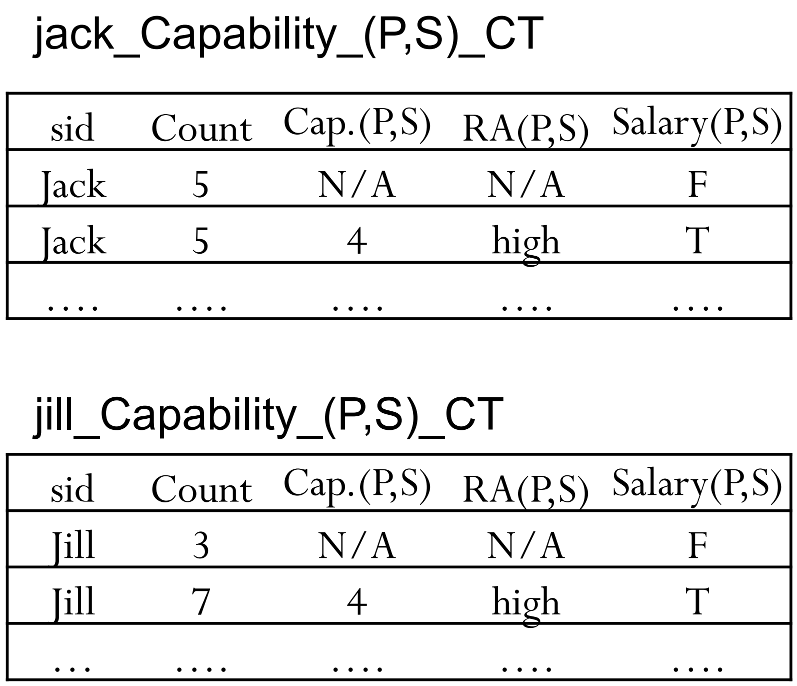
SQL Implementation of the Log-Linear Classification Score The obvious approach to computing the log-linear score would be to use the likelihood computation of Section 2.4.3 for the entire database. This is inefficient because only instance counts that involve the target entity change the classification probability. For example, if jack is the target entity, then the grades of jill do not matter. This means that I need only consider query instantiations that match the appropriate logical variable with the target entity (e.g., $\mathcal{S} = jack$). I show how the log-likelihood computation of Section 2.4.3 can be adapted to compute a log-linear classification score for a set of target entities. This illustrates how the modularity of FactorBase supports reusing its components for different tasks.

For a given set of random variables, target entity instantiation counts can be represented in a contingency table that I call the **target contingency table**. Figure 2.5 shows the format of a contingency table for target entities Jack resp. Jill.

Assuming that for each node with ID @parRVID@, a target contingency table named *CDB.target_@parRVID@_CT* has been built in the Count Database *CDB*, the log-likelihood SQL is as in Section 2.4.3. For instance, the contribution of the $Capability(P, \mathcal{S})$ family is computed by the SQL query shown, but with the contingency table jack_Capability(P,S)_CT in place of Capability(P,S)_CT. The new problem is finding the target contingency table. SQL allows us to solve this easily by restricting counts to the target entity in the WHERE clause. To illustrate, suppose I want to modify the contingency table query of Figure 3.2 to compute the contingency table for $\mathcal{S} = jack$. I add the student id to the SELECT clause, and the join condition $S.s\_id = jack$ to the WHERE clause; see Table 2.4.[4] The FROM clause is the same as in Figure 3.2. The metaquery of Figure 3.2 is easily changed to produce these SELECT and WHERE clauses. The details of metequery will be discussed in section 3.3.1.

Next consider a setting where a model is to be scored against an entire test set. This occurs for instance in the standard cross-validation computation for model scoring. For concreteness, suppose the problem is to predict the intelligence of a set of students $Intelligence(jack)$, $Intelligence(jill)$, $Intelligence(student_3), \ldots, Intelligence(student_m)$. An obvious approach is to loop through the set of test instances, repeating the likelihood query

[4]Omit apostrophes for readability.


jack_Capability_(P,S)_CT

sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
Jack	5	N/A	N/A	F
Jack	5	4	high	T
....	....	....	....	....

jill_Capability_(P,S)_CT

sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
Jill	3	N/A	N/A	F
Jill	7	4	high	T
...	....	....	....	....

Figure 2.5: Target contingency tables for target = Jack and for target = Jill.

above for each single instance. Instead, SQL supports *block access* where I process the test instances as a block. Intuitively, instead of building a contingency table for each test instance, I build a single contingency table that stacks together the individual contingency tables (Figure 2.5). Blocked access can be implemented in a beautifully simple manner in SQL: I simply add the primary key id field for the target entity to the GROUP BY list; see Table 2.4.

2.6 Evaluation

The experimental study describes how FactorBase can be used to implement a challenging machine learning application: Constructing a Bayesian network model for a relational database. Bayesian networks are a good illustration of typical challenges and how RDBMS capabilities can address them. I will explain the learning details in chapter 4. In the experiments I followed a *pre-counting* approach where the count manager constructs a **joint contingency table** for *all* par-RVs in the random variable database.

Code was written in MySQL Script and Java, JRE 1.7.0. and executed with 8GB of RAM and a single Intel Core 2 QUAD Processor Q6700 with a clock speed of 2.66GHz (no hyper-threading). The operating system was Linux Centos 2.6.32. The MySQL Server version 5.5.34 was run with 8GB of RAM and a single core processor of 2.2GHz. All code and datasets are available on-line [67].

2.6.1 Datasets

I used six benchmark real-world databases. For detailed descriptions and the sources of the databases, please see [67] and the references therein. Table 2.5 summarizes basic information about the benchmark datasets. IMDb is the largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. It combines the MovieLens database[5] with data from the Internet Movie Database (IMDb)[6] following [63]. Table 2.5 provides information about the number of par-RVs generated for each database. More complex schemas generate more random variables.

Table 2.5: Datasets characteristics.
#Tuples = total number of tuples over all tables in the dataset.

Dataset	**#Relationship Tables/ Total**	**# par-RV**	**#Tuples**
Movielens	1 / 3	7	1,010,051
Mutagenesis	2 / 4	11	14,540
UW-CSE	2 / 4	14	712
Mondial	2 / 4	18	870
Hepatitis	3 / 7	19	12,927
IMDb	3 / 7	17	1,354,134

2.6.2 Results

Table 2.6 reports the number of sufficient statistics for constructing the joint contingency table. This number depends mainly on the number of par-RVs. The number of sufficient statistics can be quite large, over 15M for the largest dataset IMDb. Even with such large numbers, constructing contingency tables using the SQL metaqueries is feasible, taking just over 2 hours for the very large IMDb set. The number of Bayesian network parameters is much smaller than the number of sufficient statistics. The difference between the number of parameters and the number of sufficient statistics measures how compactly the BN summarizes the statistical information in the data.

Table 2.6 shows that Bayesian networks provide very compact summaries of the data statistics. For instance for the Hepatitis dataset, the ratio is $12,374,892/569 > 20,000$. The IMDb database is an outlier, with a complex correlation pattern that leads to a dense Bayesian network structure.

Table 2.7 shows that the graph structure of a Bayesian network contains a small number of edges relative to the number of parameters. The parameter manager provides fast

[5]www.grouplens.org, 1M version
[6]www.imdb.com, July 2013

Table 2.6: Count Manager: Sufficient Statistics and Parameters

Dataset	**# Database Tuples**	**# Sufficient Statistics (SS)**	**SS Computing Time (s)**	**#BN Parameters**
Movielens	1,010,051	252	2.7	292
Mutagenesis	14,540	1,631	1.67	721
UW-CSE	712	2,828	3.84	241
Mondial	870	1,746,870	1,112.84	339
Hepatitis	12,927	12,374,892	3,536.76	569
IMDb	1,354,134	15,538,430	7,467.85	60,059

maximum likelihood estimates for a given structure. This is because computing a local contingency table for a BN family is fast given the joint contingency table.

Table 2.7: Model Manager Evaluation

Dataset	**# Edges in Bayes Net**	**# Bayes Net Parameters**	**Parameter Learning Time (s)**
Movielens	72	292	0.57
Mutagenesis	124	721	0.98
UW-CSE	112	241	1.14
Mondial	141	339	60.55
Hepatitis	207	569	429.15
IMDb	195	60,059	505.61

Figure 2.6 compares computing predictions on a test set using an instance-by-instance loop, with a separate SQL query for each instance, v.s. a single SQL query for all test instances as a block (Table 2.4).

Table 2.8: # of Test Instances

Dataset	Movielens	Mutagenesis	UW-CSE	Mondial	Hepatitis	IMDb
#instance	4,742	3,119	576	505	2,376	46,275

Table 2.8 specifies the number of test instances for each dataset. I split each benchmark database into 80% training data, 20% test data. The test instances are the ground atoms of all descriptive attributes of entities. The blocked access method is 10-100 faster depending on the dataset. The single access method did not scale to the large IMDb dataset (timeout after 12 hours).

Table 2.9 reports result for the complete learning of a Bayesian network, structure and parameters. It benchmarks FactorBase against functional gradient boosting, a state-of-the-art multi-relational learning approach [58]. MLN_Boost learns a Markov Logic Network, and RDN_Boost a Relational Dependency Network. I used the Boostr implementation

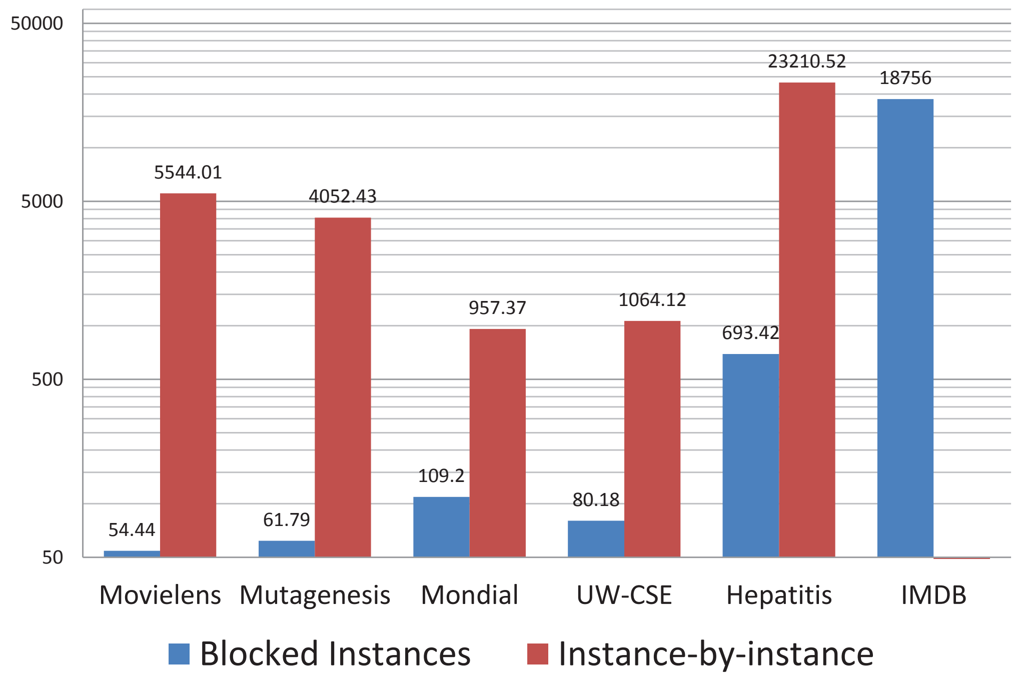


Figure 2.6: Times (s) for Computing Predictions on Test Instances. The right red column shows the time for looping over single instances using the Single Access Query of Table 2.4. The left blue column shows the time for the Blocked Access Query of Table 2.4.

Table 2.9: Learning Time Comparison (sec) with other statistical-relational learning systems. NT = non-termination

Dataset	RDN_Boost	MLN_Boost	FB-Total	FB-Count
MovieLens	5,562	N/T	1.12	0.39
Mutagenesis	118	49	1	0.15
UW-CSE	15	19	1	0.27
Mondial	27	42	102	61.82
Hepatitis	251	230	286	186.15
IMDb	N/T	N/T	524.25	439.29

[44]. To make the results easier to compare across databases and systems, I divide the total running time by the number of par-RVs for the database (Table 2.5).

Table 2.9 shows that structure learning with FactorBase is fast: even the large complex database IMDb requires only around 8 minutes/par-RV. Compared to the boosting methods, FactorBase shows excellent scalability: neither boosting method terminates on the IMDb database, and while RDN_Boost terminates on the MovieLens database, it is almost 5,000 times slower than FactorBase. Much of the speed of my implementation is due to quick access to sufficient statistics. As the last column of Table 2.9 shows, on the larger datasets FactorBase spends about 80% of computation time on gathering sufficient statistics via the count manager. This suggests that a large speedup for the boosting algorithms could be achieved if they used the FactorBase in-database design.

I do not report accuracy results because predictive accuracy is not the focus of this chapter. On the standard conditional log-likelihood metric, as defined by Equation 2.1, the model learned by FactorBase performs better than the boosting methods on all databases. This is consistent with the results of previous studies [76]. FactorBase leverages RDBMS capabilities for scalable management of statistical analysis objects. It efficiently constructs and stores large numbers of sufficient statistics and parameter estimates. The RDBMS support for statistical-relational learning translates into orders of magnitude improvements in speed and scalability.

2.7 Conclusion

Compared to traditional learning with a single data table, learning for multi-relational data requires new system capabilities. In this chapter I described FactorBase, a system that leverages the existing capabilities of an SQL-based RDBMS to support statistical-relational learning. Representational tasks include specifying metadata about structured first-order random variables, and storing the structure of a learned model. Computational tasks include storing and constructing sufficient statistics, and computing parameter estimates and model selection scores. I showed that SQL scripts can be used to implement these capabilities, with multiple advantages. These advantages include: 1) Fast program development through high-level SQL constructs for complex table and count operations. 2) Managing large and complex statistical objects that are too big to fit in main memory. For instance, some of the benchmark databases require storing and querying millions of sufficient statistics. Empirical evaluation on six benchmark databases showed significant scalability advantages from utilizing the RDBMS capabilities: Both structure and parameter learning scaled well to millions of data records, beyond what previous multi-relational learning systems can achieve.

Chapter 3

Computing Multi-Relational Sufficient Statistics for Large Databases

Relational databases contain information about attributes of entities, and which relationships do and do not hold among entities. To make this information accessible for knowledge discovery requires computing *sufficient statistics*. Whereas sufficient statistics with positive relationships only can be efficiently computed by SQL joins of existing database tables, a table join approach is not feasible for negative relationships. Negative relationships concern the nonexistence of a relationship. This is because people would have to enumerate all tuples of entities that are *not* related (consider the number of user pairs who are *not* friends on Facebook). The cost of the enumeration approach is close to materializing the Cartesian cross product of entity sets, which grows exponentially with the number of entity sets involved. It may therefore seem that sufficient statistics with negative relationships can be computed only for small databases.

Within this chapter, I describes a new dynamic programming algorithm for computing cross-table sufficient statistics that may contain any number of *positive and negative* relationships. I show that on the contrary, assuming that sufficient statistics with positive relationships are available, extending them to negative relationships can be achieved in a highly scalable manner, which does not depend on the size of the database.

3.1 Introduction

3.1.1 Virtual Join Approach

My approach to this problem introduces a new Virtual Join operation. A Virtual Join algorithm computes sufficient statistics *without* materializing a cross product [94]. Sufficient statistics can be represented in contingency tables [57]. The Virtual Join operation is a dynamic programming algorithm that successively builds up a large contingency table from smaller ones, *without a need to access the original data tables.* I refer to it as the Möbius Join since it is based on the Möbius extension theorem [78].

I introduce algebraic operations on contingency tables that generalize standard relational algebra operators. I establish a contingency table algebraic identity that reduces the computation of sufficient statistics with $k+1$ negative relationships to the computation of sufficient statistics with only k negative relationships. The Möbius Join applies the identity to construct contingency tables that involve $1, 2, \ldots, \ell$ relationships (positive and negative), until I obtain a joint contingency table for all tables in the database. A theoretical upper bound for the number of contingency table operations required by the algorithm is $O(r \log r)$, where r is the number of sufficient statistics involving negative relationships. In other words, the number of table operations is nearly linear in the size of the required output.

3.1.2 Evaluation

I evaluate the *scalability* of my Möbius Join algorithm by computing contingency tables for seven real-world databases. The observed computation times exhibit the near-linear growth predicted by the theoretical analysis. They range from two seconds on the simpler database schemas to just over two hours for the most complex schema with over 1 million tuples from the IMDb database.

Given that computing sufficient statistics for negative relationships is *feasible*, the remainder of the experiments evaluate their *usefulness*. These sufficient statistics allow statistical analysis to utilize the absence or presence of a relationship as a feature. The benchmark datasets provide evidence that the positive and negative relationship features enhance different types of statistical analysis, as follows.

1. Feature selection: When provided with sufficient statistics for negative and positive relationships, a standard feature selection method selects relationship features for classification.

2. Association Rule Mining: A standard association rule learning method includes many association rules with relationship conditions in its top 20 list.

Contributions My main contributions are as follows.

1. A dynamic program to compute a joint contingency table for sufficient statistics that combine several tables, and that may involve any number of *positive and negative* relationships.

2. An extension of relational algebra for contingency tables that supports the dynamic program conceptually and computationally.

Chapter Organization The main part of the chapter describes the dynamic programming algorithm for computing a joint contingency table for all random variables. I define the contingency table algebra. A complexity analysis establishes feasible upper bounds on the number of contingency table operations required by the Möbius Join algorithm. I also investigate the scalability of the algorithm empirically. The final set of experiments examines how the cached sufficient statistics support the analysis of cross-table dependencies for different learning and data mining tasks.

3.2 Background and Notation

As described in section 2.2, I assume a standard **relational schema** containing a set of tables, each with key fields, descriptive attributes, and possibly foreign key pointers. A **database instance** specifies the tuples contained in the tables of a given database schema, see Figure 1.2 and 1.3. I also adopt the function-based notation from section 2.2.

The functor formalism is rich enough to represent the constraints of an entity-relationship schema via the following translation: Entity sets correspond to populations, descriptive attributes to functions, relationship tables to relationships, and foreign key constraints to type constraints on the arguments of relationship predicates. Table 2.1 illustrates this translation, distinguishing attributes of entities (*1Attributes*) and attributes of relationships (*2Attributes*).

3.2.1 Related Work

Sufficient Statistics for Single Data Tables Several data structures have been proposed for storing sufficient statistics defined on a *single* data table. One of the best-known are ADtrees [57]. An ADtree provides a memory-efficient data structure for *storing* and retrieving sufficient statistics once they have been computed. In this chapter, I focus on the problem of *computing* the sufficient statistics, especially for the case where the relevant rows have not been materialized. Thus ADtrees and contingency tables are complementary representations for different purposes: contingency tables support a computationally efficient block access to sufficient statistics, whereas ADtrees provide a memory efficient compression of the sufficient statistics. An interesting direction for future work is to build an ADtree for the contingency table once it has been computed.

Relational Sufficient Statistics Schulte *et al.* review previous methods for computing statistics with negative relationships [78]. They show that the fast Möbius transform can be used in the case of multiple negative relationships. Their evaluation considered only Bayes net parameter learning with only one relationship. I examined computing joint sufficient statistics over the entire database. Other novel aspects are the *ct*-table operations and using the relationship chain lattice to facilitate dynamic programming.

3.2.2 Review of Contingency Table

Here I review the *contingency table* representation of sufficient statistics which was introduced in section 2.3. Consider a fixed list of random variables. A **query** is a set of (*variable* = *value*) pairs where each value is of a valid type for the random variable. The **result set** of a query in a database $\mathcal{D}$ is the set of instantiations of the first-order variables such that the query evaluates as true in $\mathcal{D}$. For example, in the database of Figure 1.3 the result set for the query ($\mathit{intelligence}(\mathcal{S}) = \mathit{2}$, $\mathit{rank}(\mathcal{S}) = \mathit{1}$, $\mathit{popularity}(P) = \mathit{3}$, $\mathit{teachingability}(P) = \mathit{1}$, $\mathit{RA}(P, \mathcal{S}) = T$) is the singleton $\{\langle \mathit{kim}, \mathit{oliver}\rangle\}$. The **count** of a query is the cardinality of its result set.

For every set of variables $\mathbf{V} = \{V_1, \ldots, V_n\}$ there is a **contingency table** $ct(\mathbf{V})$. This is a table with a row for each of the possible assignments of values to the variables in $\mathbf{V}$, and a special integer column called *count*. The value of the *count* column in a row corresponding to $V_1 = v_1, \ldots, V_n = v_n$ records the count of the corresponding query. A **conditional contingency table**, written

$$ct(V_1, \ldots, V_k | V_{k+1} = v_{k+1}, \ldots, V_{k+m} = v_{k+m})$$

is the contingency table whose column headers are $V_1, \ldots, V_k$ and whose rows comprise the subset that match the conditions to the right of the | symbol. I assume that contingency tables omit rows with count 0.

3.2.3 Relational Contingency Table

Many relational learning algorithms take an iterative deepening approach: explore correlations along a single relationship, then along relationship chains of length 2, 3, etc. Chains of relationships form a natural lattice structure, where iterative deepening corresponds to moving from the bottom to the top. The Möbius Join algorithm computes contingency tables by reusing the results for smaller relationships for larger relationship chains.

A relationship variable 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 relationship variable $R_{i+1}(\boldsymbol{\tau}_{i+1})$ shares at least one first-order 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 Figure 1.2, a chain is formed by the two relationship variables

$$\mathit{Registration}(\mathcal{S}, \mathbb{C}), \mathit{RA}(P, \mathcal{S}).$$

If relationship variable $Teaches(P, \mathbb{C})$ is added, I may have a three-element chain

$$Registration(\mathcal{S}, \mathbb{C}), RA(P, \mathcal{S}), Teaches(P, \mathbb{C}).$$

The subset ordering defines a lattice on relationship sets/chains. Figure 3.1 illustrates the lattice for the relationship variables in the university schema. For reasons that I explain below, entity tables are also included in the lattice and linked to relationships that involve the entity in question.

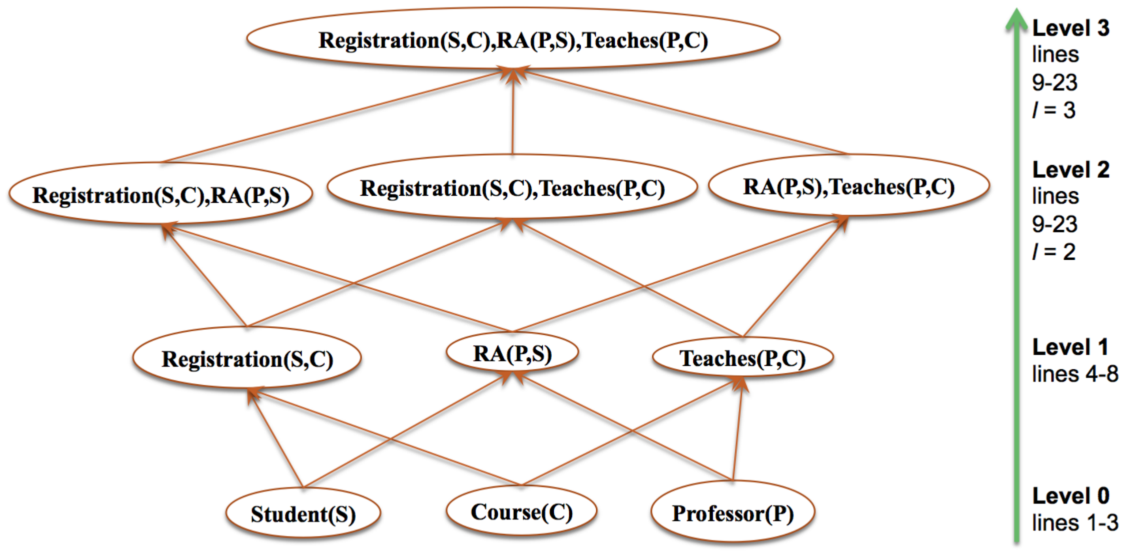


Figure 3.1: A lattice of relationship sets for the university schema of Figure 1.2. The Möbius Join constructs contingency table tables for each relationship chain for each level ℓ of the lattice. I reference the lines of the pseudo-code in Algorithm 3.

With each relationship chain $\mathbf{R}$ (Rchain for short) is associated a *ct*-table $ct_{\mathbf{R}}$. The variables in the *ct*-table $ct_{\mathbf{R}}$ comprise the relationship variables in $\mathbf{R}$, and the unary/binary descriptive attributes associated with each of the relationships. To define these, I introduce the following notation (cf. Table 2.1).

- $\mathit{1Attributes}(\mathbb{A})$ denotes the attribute variables of a first-order variable $\mathbb{A}$ collectively (1 for unary).
- $\mathit{1Attributes}(\mathbf{R})$ denotes the set of entity attribute variables for the first-order variables that are involved in the relationships in $\mathbf{R}$.
- $\mathit{2Attributes}(\mathbf{R})$ denotes the set of relationship attribute variables for the relationships in $\mathbf{R}$ (2 for binary).
- $\mathit{Atts}(\mathbf{R}) \equiv \mathit{1Attributes}(\mathbf{R}) \cup \mathit{2Attributes}(\mathbf{R})$ is the set of all attribute variables in the relationship chain $\mathbf{R}$.

In this notation, the variables in the *ct*-table $ct_{\mathbf{R}}$ are denoted as $\mathbf{R} \cup Atts(\mathbf{R})$. The goal of the Möbius Join algorithm is to compute a contingency table for each chain $\mathbf{R}$. In the example of Figure 3.1, the algorithm computes 10 contingency tables. The *ct*-table for the top element of the lattice is the **joint *ct*-table** for the entire database.

3.3 Computing Contingency Tables For Positive Relationships

If a conjunctive query involves only positive relationships, then it can be computed using SQL's count aggregate function applied to a table join. This is relatively easy for a *fixed* set of par-RVs; the challenge is a general construction that works for different sets of par-RVs.

For a fixed set, a contingency table can be computed by an SQL count(*) query of the form

```
CREATE VIEW CT-table AS
SELECT COUNT(*) AS count, <VARIABLE-LIST>
FROM <TABLE-LIST>
GROUP BY <VARIABLE-LIST>
WHERE <Join-Conditions>
```

To illustrate, I also show one concrete SQL query for computing the positive relationship part of the *ct*-table for the $RA(P, \mathcal{S})$ chain.

```
CREATE VIEW ct_T AS
SELECT COUNT(*) as count,
s.ranking, s.intelligence, p.popularity, p.teachingability, RA.capability, RA.salary
FROM professor p, student s, RA
WHERE RA.p_id = professor.p_id and RA.s_id = student.s_id
GROUP BY s.ranking, s.intelligence, p.popularity, p.teachingability,
RA.capability, RA.salary
```

3.3.1 Metaquery

For the general construction, I describe how the contingency table problem can be solved using SQL with *metaqueries*. The FactorBase uses SQL itself to construct the count-conjunction query. I refer to this construction as an SQL **metaquery**. Thus an SQL metaquery maps schema information to the components of another SQL query.

I represent a count(*) query in four kinds of tables: the Select, From, Where and Group By tables. Each of these tables lists the entries in the corresponding count(*) query part. The Select table lists the entries in the Select clause of the target query, the From table lists the entries in the From clause, and similar for Where and Group By tables. Given the four metaquery tables, the corresponding SQL count(*) query can be easily constructed and executed in an application to construct the contingency table.

Given a list of par-RVs as input, the metaquery tables are constructed as follows from the metadata in the database *VDB* as described in section 2.2.

FROM LIST Find the tables referenced by the par-RV's. A par-RV references the entity tables associated with its first-order variables (see VDB.Relationship_FOvariables). Relational par-RV's also reference the associated relationship table (see VDB.Relationship).

WHERE LIST Add join conditions on the matching primary keys of the referenced tables in the WHERE clause. The primary key columns are recorded in VDB.

SELECT LIST For each attribute par-RV, find the corresponding column name in the original database (see VDB.AttributeColumns). Rename the column with the ID of the par-RV. Add a *count* column.

GROUP BY LIST The entries of the Group By table are the same as in the Select table without the *count* column.

Metaqueries	Entries

CREATE VIEW Select_List AS
SELECT RVarID, CONCAT(‘COUNT(*)',' as "count"') AS Entries
FROM VDB.Relationship
UNION DISTINCT
SELECT RVarID, AVarID AS Entries
FROM VDB.Relationship_Attributes;
	COUNT(*) as “count"

	\`Popularity(P)\`
	\`Teachingability(P)\`
	\`Intelligence(S)\`
	\`Ranking(S)\`

CREATE VIEW From_List AS
SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries
FROM VDB.Relationship_FOvariables
UNION DISTINCT
SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries
FROM VDB.Relationship;
	@database@.prof AS P

	@database@.student AS S
	@database@.RA AS \`RA\`

CREATE VIEW Where_List AS
SELECT
RVarID, CONCAT(RVarID,'.',COLUMN_NAME,' = ',
FO-ID,'.', REFERENCED_COLUMN_NAME) AS Entries
FROM VDB.Relationship_FOvariables;
	\`RA\`.p_id = P.p_id

	\`RA\`.s_id =S.s_id

Figure 3.2: Example of metaquery results based on university database and the par-RV metadata (Table 2.2).

Figure 3.2 shows an example of a metaquery for the university database. This metaquery defines a view that in turn defines a contingency table for the random variable list associated with the relationship table *RA*. This list includes the entity attributes of professors and of students, as well as the relationship attributes of the *RA* relationship. The Bayesian network of Figure 1.5 was learned from this contingency table. The contingency table defined by the metaquery of Figure 3.2 contains only rows where the value of RA is true. Even more efficient than SQL count queries is the Tuple ID propagation method which could be used for computing query counts with positive relationships only [94].

In the next section I assume that contingency tables for positive relationships only have been computed already, and consider how such tables can be extended to full contingency tables with both positive and negative relationships (e.g, RA is true and RA is false).

3.4 Computing Contingency Tables For Negative Relationships

I describe a Virtual Join algorithm that computes the required sufficient statistics without materializing a cross product of entity sets.

1. I introduce an extension of relational algebra that I term **contingency table algebra**. The purpose of this extension is to show that query counts using $k + 1$ negative relationships can be computed from two query counts that each involve at most k relationships.

2. A dynamic programming algorithm applies the algebraic identify repeatedly to build up a complete contingency table from partial tables.

3.4.1 Contingency Table Algebra

I introduce relational algebra style operations defined on contingency tables.

Unary Operators

Selection $\sigma_{\phi} ct$ selects a subset of the rows in the ct-table that satisfy condition ϕ. This is the standard relational algebra operation except that the selection condition ϕ may not involve the *count* column.

Projection $\pi_{V_1,\ldots,V_k} ct$ selects a subset of the columns in the ct-table, excluding the count column. The counts in the projected subtable are the sum of counts of rows that satisfy the query in the subtable. The ct-table projection $\pi_{V_1,\ldots,V_k} ct$ can be defined by the following SQL code template:

```
SELECT SUM(count) AS count, $V_1, \ldots, V_k$
FROM $ct$
GROUP BY $V_1, \ldots, V_k$
```

Conditioning $\chi_{\phi} ct$ returns a conditional contingency table. Ordering the columns as $(V_1, \ldots, V_k, \ldots, V_{k+j})$, suppose that the selection condition is a conjunction of values of the form

$$\phi = (V_{k+1} = v_{k+1}, \ldots, V_{k+j} = v_{k+j}).$$

Conditioning can be defined in terms of selection and projection by the equation:

$$\chi_{\phi} ct = \pi_{V_1,\ldots,V_k}(\sigma_{\phi} ct)$$

Binary Operators

I use $\mathbf{V}$, $\mathbf{U}$ in SQL templates to denote a list of column names in arbitrary order. The notation $ct_1.\mathbf{V} = ct_2.\mathbf{V}$ indicates an equijoin condition: the contingency tables ct_1 and ct_2 have the same column set $\mathbf{V}$ and matching columns from the different tables have the same values.

Cross Product The **cross-product** of $ct_1(\mathbf{U}), ct_2(\mathbf{V})$ is the Cartesian product of the rows, where the product counts are the products of count. The cross-product can be defined by the following SQL template:

> SELECT
> $(ct_1.count * ct_2.count)$ AS *count*, $\mathbf{U}$, $\mathbf{V}$
> FROM ct_1, ct_2

Addition The **count addition** $ct_1(\mathbf{V}) + ct_2(\mathbf{V})$ adds the counts of matching rows, as in the following SQL template.

> SELECT $ct_1.count$+$ct_2.count$ AS *count*, $\mathbf{V}$
> FROM ct_1, ct_2
> WHERE $ct_1.\mathbf{V} = ct_2.\mathbf{V}$

If a row appears in one *ct*-table but not the other, I include the row with the count of the table that contains the row.

Subtraction The **count difference** $ct_1(\mathbf{V}) - ct_2(\mathbf{V})$ equals $ct_1(\mathbf{V}) + (-ct_2(\mathbf{V}))$ where $-ct_2(\mathbf{V})$ is the same as $ct_2(\mathbf{V})$ but with negative counts. Table subtraction is defined only if (i) without the *count* column, the rows in ct_1 are a superset of those in ct_2, and (ii) for each row that appears in both tables, the count in ct_1 is at least as great as the count in ct_2.

Implementation

The selection operator can be implemented using SQL as with standard relational algebra. Projection with *ct*-tables requires use of the GROUP BY construct as shown in section 3.4.1 [Unary Operators].

For addition/subtraction, assuming that a sort-merge join is used [88], a standard analysis shows that the cost of a sort-merge join is $size(table1) + size(table2) +$ the cost of sorting both tables.

The cross product is easily implemented in SQL as shown in section 3.4.1 [Binary Operators]. The cross product size is quadratic in the size of the input tables.

3.4.2 Lattice Computation of Contingency Tables

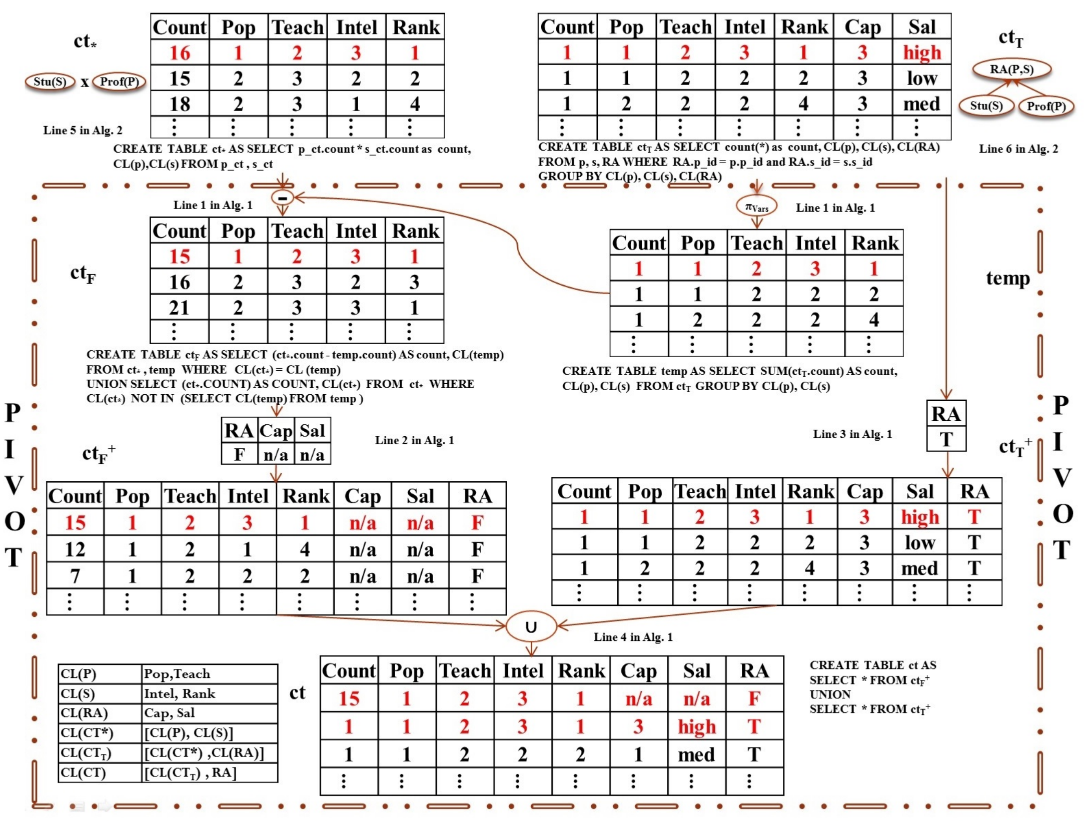
This section describes a method for computing the contingency tables level-wise in the relationship chain lattice. I start with a contingency table algebra equivalence that allows us to compute counts for rows with negative relationships from rows with positive relations. Following [57], I use a "don't care" value $*$ to indicate that a query does not specify the value of a node. For instance, the query $R_1 = \mathrm{T}, R_2 = *$ is equivalent to the query $R_1 = \mathrm{T}$.

Algorithm 2: The Pivot function returns a conditional contingency table for a set of attribute variables and all possible values of the relationship R_{pivot}, including $R_{pivot} = \mathrm{F}$. The set of conditional relationships $\mathbf{R} = (R_{pivot}, \ldots, R_\ell)$ may be empty in which case the Pivot computes an unconditional ct-table.

```
Input: Two conditional contingency tables
        $ct_{\mathrm{T}} := ct(Vars, 2Attributes(R_{pivot}) | R_{pivot} = \mathrm{T}, \mathbf{R} = \mathrm{T})$ and
        $ct_{*} := ct(Vars | R_{pivot} = *, \mathbf{R} = \mathrm{T})$ .
Precondition: The set $Vars$ does not contain the relationship variable $R_{pivot}$ nor any
  of its descriptive attributes $2Attributes(R_{pivot})$.;
Output: The conditional contingency table
        $ct(Vars, 2Attributes(R_{pivot}), R_{pivot} | \mathbf{R} = \mathrm{T})$ .
1: $ct_{\mathrm{F}} := ct_{*} - \pi_{Vars} ct_{\mathrm{T}}$.
   {Implements the algebra Equation 3.1 in proposition 1.}
2: $ct_{\mathrm{F}}^{+}$ := extend $ct_{\mathrm{F}}$ with columns $R_{pivot}$ everywhere false and $2Attributes(R_{pivot})$
   everywhere $n/a$.
3: $ct_{\mathrm{T}}^{+}$ := extend $ct_{\mathrm{T}}$ with columns $R_{pivot}$ everywhere true.
4: return $ct_{\mathrm{F}}^{+} \cup ct_{\mathrm{T}}^{+}$
```

Proposition 1. *Let R be a relationship variable and let $\mathbf{R}$ be a set of relationship variables. Let Vars be a set of variables that does not contain R nor any of the 2Attributes of R. Let $X_1, \ldots, X_l$ be the first-order variables that appear in R but not in Vars, where l is possibly zero. Then I have*

$$\begin{aligned} & ct(Vars \cup 1Attributes(R) | \mathbf{R} = \mathrm{T}, R = F) = \\ & ct(Vars | \mathbf{R} = \mathrm{T}, R = *) \times ct(X_1) \times \cdots \times ct(X_l) \\ & - ct(Vars \cup 1Attributes(R) | \mathbf{R} = \mathrm{T}, R = T). \end{aligned} \tag{3.1}$$



ct*

Count	Pop	Teach	Intel	Rank
16	1	2	3	1
15	2	3	2	2
18	2	3	1	4
⋮	⋮	⋮	⋮	⋮

```
CREATE TABLE ct* AS SELECT p_ct.count * s_ct.count as count,
CL(p),CL(s) FROM p_ct , s_ct
```

ct_T

Count	Pop	Teach	Intel	Rank	Cap	Sal
1	1	2	3	1	3	high
1	1	2	2	2	3	low
1	2	2	2	4	3	med
⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
CREATE TABLE ctT AS SELECT count(*) as count, CL(p), CL(s), CL(RA)
FROM p, s, RA WHERE RA.p_id = p.p_id and RA.s_id = s.s_id
GROUP BY CL(p), CL(s), CL(RA)
```

ct_F

Count	Pop	Teach	Intel	Rank
15	1	2	3	1
16	2	3	2	3
21	2	3	3	1
⋮	⋮	⋮	⋮	⋮

```
CREATE TABLE ctF AS SELECT (ct*.count - temp.count) AS count, CL(temp)
FROM ct* , temp WHERE CL(ct*) = CL (temp)
UNION SELECT (ct*.COUNT) AS COUNT, CL(ct*) FROM ct* WHERE
CL(ct*) NOT IN (SELECT CL(temp) FROM temp )
```

temp

Count	Pop	Teach	Intel	Rank
1	1	2	3	1
1	1	2	2	2
1	2	2	2	4
⋮	⋮	⋮	⋮	⋮

```
CREATE TABLE temp AS SELECT SUM(ctT.count) AS count,
CL(p), CL(s) FROM ctT GROUP BY CL(p), CL(s)
```

RA	Cap	Sal
F	n/a	n/a

RA
T

ct_F^+

Count	Pop	Teach	Intel	Rank	Cap	Sal	RA
15	1	2	3	1	n/a	n/a	F
12	1	2	1	4	n/a	n/a	F
7	1	2	2	2	n/a	n/a	F
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

ct_T^+

Count	Pop	Teach	Intel	Rank	Cap	Sal	RA
1	1	2	3	1	3	high	T
1	1	2	2	2	3	low	T
1	2	2	2	4	3	med	T
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

CL(P)	Pop,Teach
CL(S)	Intel, Rank
CL(RA)	Cap, Sal
CL(CT*)	[CL(P), CL(S)]
CL(CT$_T$)	[CL(CT*) ,CL(RA)]
CL(CT)	[CL(CT$_T$) , RA]

ct

Count	Pop	Teach	Intel	Rank	Cap	Sal	RA
15	1	2	3	1	n/a	n/a	F
1	1	2	3	1	3	high	T
1	1	2	2	2	1	med	T
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
CREATE TABLE ct AS
SELECT * FROM ctF+
UNION
SELECT * FROM ctT+
```

Figure 3.3: Top: Equation (3.1) is used to compute the conditional contingency table $ct_{\mathrm{F}} = ct(\mathit{1Attributes}(R)|R = F)$. (Set $\mathit{Vars} = \emptyset$, $R = RA(P, \mathcal{S})$, $\mathbf{R} = \emptyset$). Bottom: The Pivot operation computes the contingency table $ct_{RA(P,\mathcal{S})}$ for the relationship $RA(P, \mathcal{S}) := R_{pivot}$. The ct-table operations are implemented using dynamic SQL queries as shown. Lists of column names are abbreviated as shown and also as follows. $CL(ct_*) = CL(temp) = CL(ct_F)$, $CL(ct) = CL(ct^+_{\mathrm{F}}) = CL(ct^+_{\mathrm{T}})$. I reference the corresponding lines of Algorithms 2 and 3.

If $l = 0$, the equation holds without the cross-product term.

Proof. The equation

$$\begin{aligned} & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = *) = \\ & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = T) + \\ & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = F) \end{aligned} \tag{3.2}$$

holds because the set $\mathit{Vars} \cup \mathit{1Attributes}(R)$ contains all first-order variables in R.[1] Equation (3.2) implies

$$\begin{aligned} & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = \mathrm{F}) = \\ & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = *)- \\ & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = \mathrm{T}). \end{aligned} \tag{3.3}$$

To compute the *ct*-table conditional on the relationship R being unspecified, I use the equation

$$\begin{aligned} & ct(\mathit{Vars} \cup \mathit{1Attributes}(R)|\mathbf{R} = \mathrm{T}, R = *) = \\ & ct(\mathit{Vars}|\mathbf{R} = \mathrm{T}, R = *) \times ct(X_1) \times \cdots \times ct(X_l) \end{aligned} \tag{3.4}$$

which holds because if the set *Vars* does not contain a first-order variable of R, then the counts of the associated $\mathit{1Attributes}(R)$ are independent of the counts for *Vars*. If $l = 0$, there is no new first-order variable, and Equation (3.4) holds without the cross-product term. Together Equations (3.3) and (3.4) establish the proposition. □

Figure 3.3 illustrates Equation (3.1). The construction of the ct_{F} table in Algorithm 2 provides pseudo-code for applying Equation (3.1) to compute a complete *ct*-table, given a partial table where a specified relationship variable R is true, and another partial table that does not contain the relationship variable. I refer to R as the **pivot** variable. For extra generality, Algorithm 2 applies Equation (3.1) with a condition that lists a set of relationship variables fixed to be true. Figure 3.3 illustrates the Pivot computation for the case of only one relationship. Algorithm 3 shows how the Pivot operation can be applied repeatedly to find all contingency tables in the relationship lattice.

Initialization. Compute *ct*-tables for entity tables. Compute *ct*-tables for each single relationship variable R , conditional on $R = \mathrm{T}$. If $R = *$, then no link is specified between the first-order variables involved in the relation R. Therefore the individual counts for each first-order variable are independent of each other and the joint counts can be obtained by the cross product operation. Apply the Pivot function to construct the complete *ct*-table for relationship variable R.

Lattice Computation. The goal is to compute *ct*-tables for all relationship chains of length > 1. For each relationship chain, order the relationship variables in the chain arbitrarily. Make each relationship variable in order the Pivot variable R_i. For the current Pivot variable R_i, find the conditional *ct*-table where R_i is unspecified, and the subsequent relations R_j with $j > i$ are true. This *ct*-table can be computed from a *ct*-table for a shorter chain that has been constructed already. The conditional *ct*-table has been constructed al-

[1] I assume here that for each first-order variable, there is at least one *1Attribute*, i.e., descriptive attribute.

Algorithm 3: Möbius Join algorithm for Computing the Contingency Table for Input Database.

```
Input: A relational database $\mathcal{D}$; a set of variables
Output: A contingency table that lists the count in the database $D$ for each
        possible assignment of values to each variable.
1:  for each first-order variables $X$ do
2:      compute $ct(\mathit{1Attributes}(X))$ using SQL queries.
3:  end for
4:  for each relationship variable $R$ do
5:      $ct_* := ct(X) \times ct(Y)$ where $X$,$Y$ are the first-order variables in $R$.
6:      $ct_{\mathrm{T}} := ct(\mathit{1Attributes}(R)|R = \mathrm{T})$ using SQL joins.
7:      Call $Pivot(ct_{\mathrm{T}}, ct_*)$ to compute $ct(\mathit{1Attributes}(R), \mathit{2Attributes}(R), R)$.
8:  end for
9:  for Rchain length $\ell = 2$ to $m$ do
10:     for each Rchains $\mathbf{R} = R_1, \ldots, R_\ell$ do
11:         $Current\_ct := ct(\mathit{1Attributes}(R_1, \ldots, R_\ell), \mathit{2Attributes}(R_1, \ldots, R_\ell)|R_1 =$
            $\mathrm{T}, \ldots, R_\ell = \mathrm{T})$ using SQL joins.
12:         for $i = 1$ to $\ell$ do
13:             if $i$ equals 1 then
14:                 $ct_* := ct(\mathit{1Attributes}(R_2, \ldots, R_\ell), \mathit{2Attributes}(R_2, \ldots, R_\ell)|R_1 = *, R_2 =$
                    $\mathrm{T}, \ldots, R_\ell = \mathrm{T}) \times ct(X)$ where $X$ is the first-order variable in $R_1$, if any, that
                    does not appear in $R_2, \ldots, R_\ell$ {$ct_*$ can be computed from a $ct$-table for a
                    Rchain of length $\ell - 1$.}
15:             else
16:                 $\mathit{1Attributes}_{\bar{i}} := \mathit{1Attributes}(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_\ell)$.
17:                 $\mathit{2Attributes}_{\bar{i}} := \mathit{2Attributes}(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_\ell)$.
18:                 $ct_* := ct(\mathit{1Attributes}_{\bar{i}}, \mathit{2Attributes}_{\bar{i}}, R_1, \ldots, R_{i-1})|R_i = *, R_{i+1} = \mathrm{T}, \ldots, R_\ell =$
                    $\mathrm{T}) \times ct(Y)$ where $Y$ is the first-order variable in $R_i$, if any, that does not
                    appear in $\mathbf{R}$.
19:             end if
20:             $Current\_ct := Pivot(Current\_ct, ct_*)$.
21:         end for{Loop Invariant: After iteration $i$, the table $Current\_ct$ equals
            $ct(\mathit{1Attributes}(R_1, \ldots, R_\ell), \mathit{2Attributes}(R_1, \ldots, R_\ell), R_1, \ldots, R_i|R_{i+1} =$
            $\mathrm{T}, \ldots, R_\ell = \mathrm{T})$}
22:     end for{Loop Invariant: The $ct$-tables for all Rchains of length $\ell$ have been
        computed.}
23: end for
24: return the $ct$-table for the Rchain involves all the relationship variables.
```

ready, where R_i is true, and the subsequent relations are true (see loop invariant). Apply the Pivot function to construct the complete *ct*-table, for any Pivot variable R_i, conditional on the subsequent relations being true.

lines	Operation	Resulting ct-table
11	Reg(S,C) = T, RA(P,S) = T	Current_ct
13-14	i = 1, Reg(S,C) = *, RA(P,S) = T	ct_*
20	PIVOT	Current_ct
16-18	i = 2, RA(P,S) = *	ct_*
20	PIVOT	Final ct-table for Reg(S,C),RA(P,S)

Figure 3.4: Illustrates the relationship chain loop of Algorithm 3 (lines 11-21) for the chain $\mathbf{R} = Reg(\mathcal{S}, \mathbb{C}), RA(P, \mathcal{S})$. This loop is executed for each relationship chain at each level.

3.4.3 Complexity Analysis

The key point about the Möbius Join (MJ) algorithm is that it avoids materializing the cross product of entity tuples. *The algorithm accesses only* ***existing*** *tuples, never constructs nonexisting tuples.* The number of *ct*-table operation is therefore independent of the number of data records in the original database. The Virtual Join algorithm scales well with the number of rows, but not with the number of columns and relationships in the database. This limitation stems from the fact that the contingency table size grows exponentially with the number of random variables in the table.

I bound the total number of *ct*-algebra operations performed by the Möbius Join algorithm in terms of the size of its output: the number of sufficient statistics that involve negative relationships.

Proposition 2. *The number of ct-table operations performed by the Möbius Join algorithm is bounded as*

$$\#ct\_ops = O(r \cdot \log_2 r)$$

where r is the number of sufficient statistics that involve negative relationships.

To analyze the computational cost, I examine the total number of *ct*-algebra operations performed by the Möbius Join algorithm. I provide upper bounds in terms of two parameters: the number of relationship nodes m, and the number of rows r in the *ct*-table that involve negative relationships. For these parameters I establish that

$$\#ct\_ops = O(r \cdot \log_2 r) = O(m \cdot 2^m).$$

This shows the efficiency of the algorithm for the following reasons. (i) Since the time cost of any algorithm must be at least as great as the time for writing the output, which is as least as great as r, the Möbius Join algorithm adds at most a logarithmic factor to this lower bound. (ii) The second upper bound means that the number of *ct*-algebra operations is fixed-parameter tractable with respect to m.[2] In practice the number m is on the order of the number of tables in the database, which is very small compared to the number of tuples in the tables.

Derivation of Upper Bounds. For a given relationship chain of length ℓ, the Möbius Join algorithm goes through the chain linearly (Algorithm 3 inner for loop line 12). At each iteration, it computes a ct_* table with a single cross product, then performs a single Pivot operation. Each Pivot operation requires three *ct*-algebra operations. Thus overall, the number of *ct*-algebra operations for a relationship chain of length ℓ is $6 \cdot \ell = O(\ell)$. For a fixed length ℓ, there are at most $\binom{m}{\ell}$ relationship chains. Using the known identity[3]

$$\sum_{\ell=1}^{m} \binom{m}{\ell} \cdot \ell = m \cdot 2^{m-1} \tag{3.5}$$

I obtain the $O(m \cdot 2^{m-1}) = O(m \cdot 2^m)$ upper bound.

Here I give a quick proof of the identity (3.5).

Proof. With the Binomial theorem, for any non-negative integers l, m, the binomial formula of two variables a, b is

$$f(a,b) = \sum_{\ell=0}^{m} \binom{m}{\ell} \cdot a^{m-\ell} \cdot b^{\ell} = (a+b)^m.$$

Suppose I substitute a with one, for any b, I have

$$f(1,b) = \sum_{\ell=0}^{m} \binom{m}{\ell} \cdot b^{\ell} = (1+b)^m.$$

And then I could compute the partial derivative with respect to b

$$f_b'(1,b) = \sum_{\ell=1}^{m} \binom{m}{\ell} \cdot \ell \cdot b^{\ell-1} = m \cdot (1+b)^{m-1}.$$

By substituting b with one, we end the proof

$$f_b'(1,1) = \sum_{\ell=1}^{m} \binom{m}{\ell} \cdot \ell \cdot 1^{\ell-1} = \sum_{\ell=1}^{m} \binom{m}{\ell} \cdot \ell = m \cdot (2)^{m-1}.$$

□

[2]For arbitrary m, the problem of computing a *ct* table in a relational structure is #P-complete [16, Prop.12.4].

[3]math.wikia.com/wiki/Binomial_coefficient, Equation 6a

For the upper bound in terms of *ct*-table rows r, I note that the output *ct*-table can be decomposed into 2^m subtables, one for each assignment of values to the m relationship nodes. Each of these subtables contains the same number of rows d , one for each possible assignment of values to the attribute nodes. Thus the total number of rows is given by $r = d \cdot 2^m$. Therefore I have $m \cdot 2^m = \log_2(r/d) \cdot r/d \leq \log_2(r) \cdot r$. Thus the total number of *ct*-algebra operations is $O(r \cdot \log_2(r))$.

From this analysis I see that both upper bounds are overestimates. (1) Because relationship chains must be linked by foreign key constraints, the number of valid relationship chains of length ℓ is usually much smaller than the number of all possible subsets $\binom{m}{\ell}$. (2) The constant factor d grows exponentially with the number of attribute nodes, so $\log_2(r) \cdot r$ is a loose upper bound on $\log_2(r/d) \cdot r/d$. I conclude that the number of *ct*-algebra operations is not the critical factor for scalability, but rather the cost of carrying out a single *ct*-algebra operation. In the benchmark datasets, the number of sufficient statistics was feasible, as I report below. In Section 3.7 below I discuss options in case the number of sufficient statistics grows too large.

Table 3.1: Notations for time complexity analysis

number of relationship tables	m
number of entity tables	k
size of entity table rows (domain)	n
size of relationship table rows	j
number of attribute columns (predicates)	c
number of values per attributes	v
number of sufficient statistics for true relationships only	$s = v^c * m$
number of sufficient statistics for positive and negative relationships	$ss = v^c * m * 2^m$
time complexity of Virtual Join algorithm with respect to s	$O(s * (2^m - 1))$
number of rows for all cross table join	$cp = n^k * j^m$
the compression ratio of cp over ss	$cr = \frac{cp}{ss}$

Time Complexity Analysis in terms of input raw database. The Virtual Join algorithm scales well with the number of rows, but not with the number of columns and relationships in the database. This limitation stems from the fact that the contingency table size grows exponentially with the number of random variables in the table.

To analysis the time complexity in terms of the input raw database with multiple tables, I introduce the notations for some important parameters (cf. Table 3.1). Let s be the number of sufficient statistics where all relationship variables are true. For instance, if there are c attributes, the number of values per attributes is v, then $s = v^c * m$. The number of sufficient statistics for the whole input database with positive and negative relationships can be represented as

$$ss = v^c * m * 2^m.$$

Then I have

$$r = v^c * m * (2^m - 1) = s * (2^m - 1),$$

since there are $2^m - 1$ combinations of relationship values with at least one negative relationship. So the time complexity of Virtual Join algorithm with respect to s is

$$O(s * (2^m - 1)).$$

The experiments in section 3.5.2 will demonstrate the compression ratio cr is an important parameter as well. Give the notations in Table 3.1, I establish that

$$cr = \frac{cp}{ss} = \frac{n^k * j^m}{v^c * 2^m}.$$

These numbers align into two independent computational spaces. Generally the higher the compression ratio, the higher the time savings. However, if the compression ratio is unusually low, so materializing the cross-product was faster. For further depth analysis of how cr will affect the performance of the Virtual Join algorithm, I leave it to future work.

3.5 Evaluation of Contingency Table Computation: Scalability

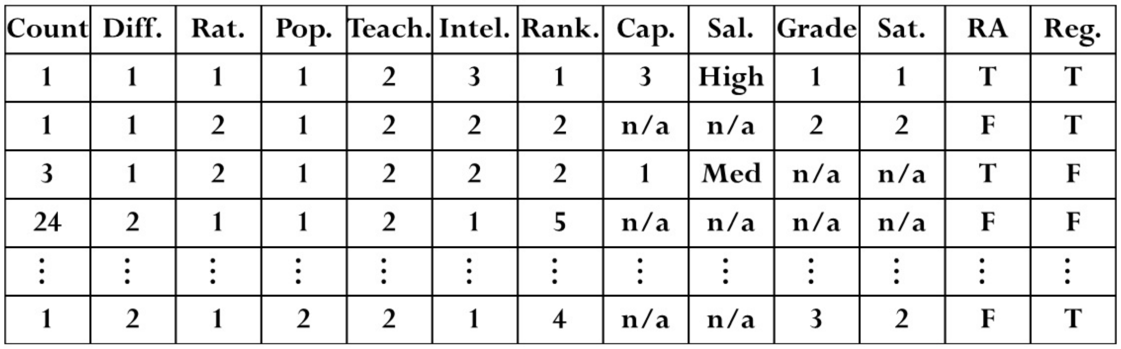
3.5.1 Datasets

I used seven benchmark real-world databases. For detailed descriptions and the sources of the databases, please see reference [76]. Table 3.2 summarizes basic information about the benchmark datasets. A self-relationship relates two entities of the same type (e.g. *Borders* relates two countries in Mondial). Random variables for each database were defined as described in Section 3.2 (see also [76]). IMDb is the largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. It combines the MovieLens database[4] with data from the Internet Movie Database (IMDb)[5] following [63].

3.5.2 Joint Contingency Tables With Negative Relationships

In this subsection I compare two different approaches for constructing the *joint contingency tables* for all variables together, for each database: **the Möbius Join algorithm (MJ)** vs. **materializing the cross product (CP)** of the entity tables for each first-order variable (primary keys). Cross-checking the MJ contingency tables with the cross-product contingency tables confirmed the correctness of the implementation. Figure 3.5 shows the

[4]www.grouplens.org, 1M version
[5]www.imdb.com, July 2013



Count	Diff.	Rat.	Pop.	Teach.	Intel.	Rank.	Cap.	Sal.	Grade	Sat.	RA	Reg.
1	1	1	1	2	3	1	3	High	1	1	T	T
1	1	2	1	2	2	2	n/a	n/a	2	2	F	T
3	1	2	1	2	2	2	1	Med	n/a	n/a	T	F
24	2	1	1	2	1	5	n/a	n/a	n/a	n/a	F	F
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	2	1	2	2	1	4	n/a	n/a	3	2	F	T

Figure 3.5: Excerpt from the joint contingency table for the university database of Figure 1.3.

Table 3.2: Datasets characteristics.
#Tuples = total number of tuples over all tables in the dataset.

Dataset	**#Relationship Tables/ Total**	**#Self Relationships**	**#Tuples**	**#Attributes**
Movielens	1 / 3	0	1,010,051	7
Mutagenesis	2 / 4	0	14,540	11
Financial	3 / 7	0	225,932	15
Hepatitis	3 / 7	0	12,927	19
IMDb	3 / 7	0	1,354,134	17
Mondial	2 / 4	**1**	870	18
UW-CSE	2 / 4	**2**	712	14

joint contingency table for the university database. The value of a relationship attribute is undefined for entities that are not related.

Table 3.3 compares the time and space costs of the MJ vs. the CP approach. The cross product was materialized using an SQL query. The ratio of the cross product size to the number of statistics in the *ct*-table measures how much compression the *ct*-table provides compared to enumerating the cross product. It shows that cross product materialization requires an infeasible amount of space resources. The *ct*-table provides a substantial compression of the statistical information in the database, by a factor of over 4,500 for the largest database IMDb.

Computation Time. The numbers shown are the complete computation time for all statistics. For faster processing, both methods used a B+ tree index built on each column in the original dataset. The MJ method also utilized B+ indexes on the *ct*-tables. I include the cost of building these indexes in the reported time. The Möbius Join algorithm returned a contingency table with negative relationships in feasible time. On the biggest dataset IMDb with 1.3 million tuples, it took just over 2 hours.

Table 3.3: Constructing the contingency table for each dataset.
M = million. N.T. = non-termination. Compress Ratio = CP-#tuples/#Statistics.
Computation times are given in seconds.

Dataset	**MJ-time**(s)	**CP-time**(s)	**CP-#tuples**	**#Statistics**	**Compress Ratio**
Movielens	2.70	703.99	23M	252	93,053.32
Mutagenesis	1.67	1096.00	1M	1,631	555.00
Financial	1421.87	N.T.	149,046,585M	3,013,011	49,467,653.90
Hepatitis	3536.76	N.T.	17,846M	12,374,892	1,442.19
IMDb	7467.85	N.T.	5,030,412,758M	15,538,430	323,740,092.05
Mondial	1112.84	132.13	5M	1,746,870	2.67
UW-CSE	3.84	350.30	10M	2,828	3,607.32

The cross product construction did not always terminate, crashing after around 4, 5, and 10 hours on Financial, IMDb and Hepatitis respectively. When it did terminate, it took orders of magnitude longer than the MJ method except for the Mondial dataset. Generally the higher the compression ratio, the higher the time savings. On Mondial the compression ratio is unusually low, so materializing the cross-product was faster.

3.5.3 Contingency Tables with Negative Relationships vs. Positive Relationships Only

In this section I compare **the time and space costs** of computing both positive and negative relationships, vs. positive relationships only. I use the following terminology. **Link Analysis On** refers to using a contingency table with sufficient statistics for both positive and negative relationships. An example is table *ct* in Figure 3.3. **Link Analysis Off** refers to using a contingency table with sufficient statistics for positive relationships only. An example is table ct^{+}_{T} in Figure 3.3. Table 3.4 shows the number of sufficient statistics required for link analysis on vs. off. The difference between the link analysis on statistics and the link analysis off statistics is the number of Extra Statistics.

The Extra Time column shows how much time the MJ algorithm requires to compute the Extra Statistics *after* the contingency tables for positive relationships are constructed using SQL joins.

As Figure 3.6 illustrates, the Extra Time stands in a nearly linear relationship to the number of Extra Statistics, which confirms the analysis of Section 3.4.3. Figure 3.7 shows that most of the MJ run time is spent on the Pivot component (Algorithm 2) rather than the main loop (Algorithm 3). In terms of *ct*-table operations, most time is spent on subtraction/union rather than cross product.

Table 3.4: Number of Sufficient Statistics for Link Analysis On and Off.
Extra Time refers to the total MJ time (Table 3.3 Col.2) minus the time for computing the positive statistics only.

Dataset	Link On	Link Off	#extra statistics	extra time (s)
MovieLens	252	210	42	0.27
Mutagenesis	1,631	565	1,066	0.99
Financial	3,013,011	8,733	3,004,278	1416.21
Hepatitis	12,374,892	2,487	12,372,405	3535.51
IMDb	15,538,430	1,098,132	14,440,298	4538.62
Mondial	1,746,870	0	1,746,870	1112.31
UW-CSE	2,828	2	2,826	3.41

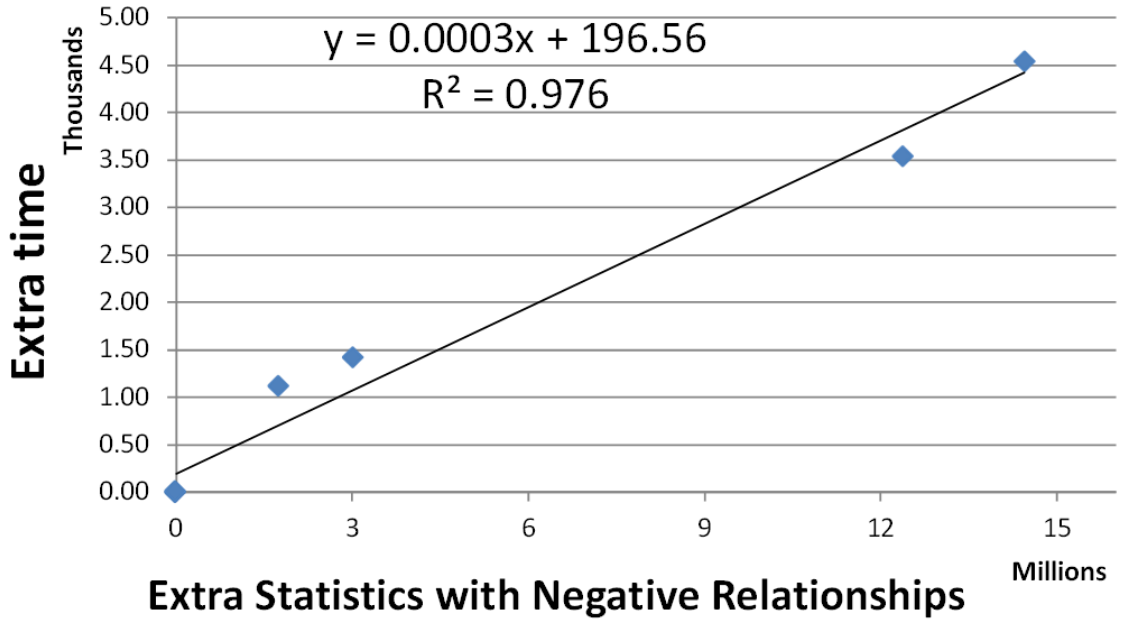
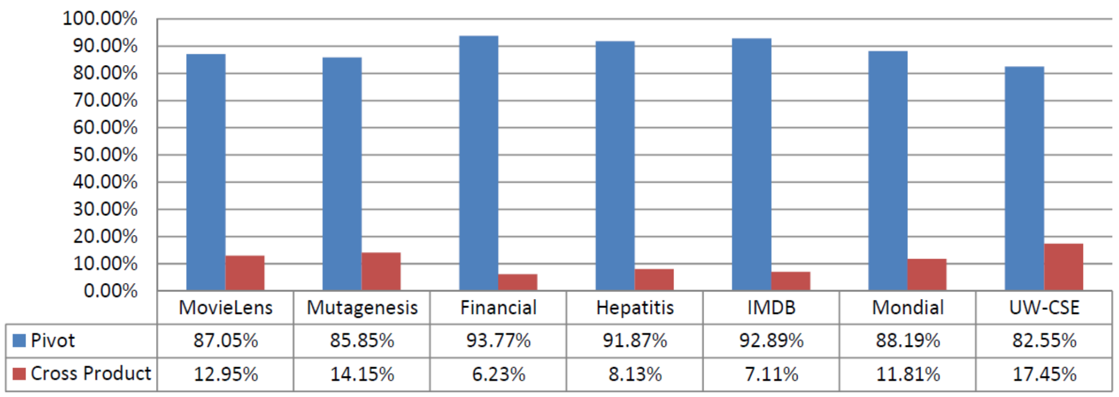


Figure 3.6: Möbius Join Extra Time (s)

3.6 Evaluation of Statistical Applications: Usefulness

I evaluate the usefulness on two different types of cross-table statistical analysis: feature selection and association rule mining.



	MovieLens	Mutagenesis	Financial	Hepatitis	IMDB	Mondial	UW-CSE
🟦 Pivot	87.05%	85.85%	93.77%	91.87%	92.89%	88.19%	82.55%
🟥 Cross Product	12.95%	14.15%	6.23%	8.13%	7.11%	11.81%	17.45%

Figure 3.7: Breakdown of MJ Total Running Time

3.6.1 Feature Selection

For each database, I selected a target for classification, then used Weka's CFS[6] feature subset selection method (Version 3.6.7) to select features for classification [28], given a contingency table. The idea is that if the existence of relationships is relevant to classification, then there should be a difference between the set selected with link analysis on and that selected with link analysis off. I measure how different two feature sets are by 1-Jaccard's coefficient:

$$\mathit{Distinctness}(A, B) = \mathit{1} - \frac{A \cap B}{A \cup B}.$$

Table 3.5: Selected Features for Target variables for Link Analysis Off vs. Link Analysis On. Rvars denotes the number of relationship features selected.

Dataset	Target variable	# Selected Attributes		Distinctness
		Link Analysis Off	Link Analysis On / Rvars	
MovieLens	Horror(M)	2	2 / 0	0.0
Mutagenesis	inda(M)	3	3 / 0	0.0
Financial	balance(T)	3	2 / 1	1.0
Hepatitis	sex(D)	1	2 / 1	0.5
IMDb	avg_revenue(D)	5	2 / 1	1.0
Mondial	percentage(C)	Empty CT	4 / 0	1.0
UW-CSE	courseLevel(C)	1	4 / 2	1.0

Distinctness measures how different the selected feature subset is with link analysis on and off, on a scale from 0 to 1. Here 1 = maximum dissimilarity. Table 3.5 compares the

[6]The Correlation Feature Selection (CFS) measure evaluates subsets of features on the basis of the following hypothesis: "Good feature subsets contain features highly correlated with the classification, yet uncorrelated to each other"[29].

feature sets selected. In almost all datasets, sufficient statistics about negative relationships generate new relevant features for classification. In 4/7 datasets, the feature sets are disjoint (coefficient = 1). For the Mutagenesis and MovieLens data sets, no new features are selected.

3.6.2 Association Rules

A widely studied task is finding interesting association rules in a database. I considered association rules of the form *body* → *head*, where *body* and *head* are conjunctive queries. An example of a cross-table association rule for Financial is

$$\mathit{statement\_freq.}(\mathit{Acc}) = \mathit{monthly} \rightarrow \mathit{HasLoan}(\mathit{Acc}, \mathit{Loan}) = \mathrm{T}.$$

I searched for interesting rules using both the link analysis off and the link analysis on contingency tables for each database. The idea is that if a relationship variable is relevant for other features, it should appear in an association rule. With link analysis off, all relationship variables always have the value T, so they do not appear in any association rule. I used Weka's Apriori[7] implementation to search for association rules in both modes. The interestingness metric was Lift[8]. Parameters were set to their default values. Table 3.6 shows the number of rules that utilize relationship variables with link analysis on, out of the top 20 rules. In all cases, a majority of rules utilize relationship variables, in Mutagenesis and IMDb all of them do.

Table 3.6: Number of top 20 Association Rules that utilize relationship variables.

Dataset	MovieLens	Mutagenesis	Financial	Hepatitis	IMDb	Mondial	UW-CSE
# rules	14/20	20/20	12/20	15/20	20/20	16/20	12/20

3.7 Conclusion

Utilizing the information in a relational database for statistical modelling and pattern mining requires fast access to multi-relational sufficient statistics, that combine information across database tables. I presented an efficient dynamic program that computes sufficient statistics for any combination of positive *and* negative relationships, starting with a set of statistics for positive relationships only. My dynamic program performs a Virtual Join operation, that counts the number of statistics in a table join without actually constructing the join. I showed that the run time of the algorithm is $O(r \log r)$, where r is the number of sufficient statistics to be computed. The computed statistics are stored in contingency tables. I introduced contingency table algebra, an extension of relational algebra, to elegantly

[7]Apriori heuristic: if any length k pattern is not frequent in the database, its length (k + 1) super-pattern can never be frequent [1]

[8]Let X be an item-set, and $X \rightarrow Y$ be an association rule. The **Lift** value of this rule is defined as the ratio of the observed support to that expected if X and Y were independent.

describe and efficiently implement the dynamic program. Empirical evaluation on seven benchmark databases demonstrated the scalability of the algorithm; I compute sufficient statistics with positive and negative relationships in databases with over 1 million data records. The experiments illustrated how access to sufficient statistics for both positive and negative relationships enhances feature selection and rule mining.

While I have focused on statistical analysis, one potential application is to complement the traditional ETL (extraction/transformation/loading) tasks [31]. The ETL + single table machine learning training based model can take advantage of the extra sufficient statistics by incorporating both positive and negative relationships. This could reduce the information loss caused by propositionalization process as well. The computationally efficient block access to sufficient statistics is beneficial to learning tasks (e.g., classification) as long as the classifier accepts the weights of the instances as input. For instance, the contingency tables can be loaded directly into decision tree learner of Weka.

Chapter 4

Learning Bayes Nets for Relational Data With Link Uncertainty

Classic AI research established a fundamental distinction between two types of probabilities associated with a relational structure [30, 4]. *Class-level probabilities* also called type 1 probabilities are assigned to the rates, statistics, or frequencies of events in a database. These concern classes of entities (e.g., students, courses, users) rather than the specific entities. *Instance-level probabilities* also called type 2 probabilities are assigned to specific, non-repeatable events or the properties of specific entities. Syntactically, class-level probabilities are assigned to formulas that contain 1st-order variables (e.g., $P(\mathit{Flies}(X)|\mathit{Bird}(X) = \mathit{90}\%$, or "birds fly" with probability 90%), whereas instance-level probabilities are assigned to formulas that contain constants only (e.g., $P(\mathit{Flies}(\mathit{tweety}) = \mathit{90}\%)$. Within in this chapter I focus on the class-level probabilities and will discuss the instance-level probabilities in chapter 5.

4.1 Background

Standard machine learning techniques are applied to data stored in a single table, that is, in nonrelational, propositional, or "flat" format [56]. Relational data introduces new machine learning problem, building a model that can answer generic statistical queries about classes of individuals in the database [24]. The field of SRL aims to extend machine learning algorithms to relational data [22, 12]. Building a *generative statistical model* for the variables in an application domain [22] is one of the major tasks. The generative statistical model class studied in this chapter are extensions of Bayes nets (BNs) for relational structures.

Examples of such class-level queries include: 1), In a social network, a class-level query may be "what is the percentage of friendship pairs where both are women?" 2), In a University database,"What fraction of the grades are awarded to highly intelligent students?" 3), A movie database example would be "what is the percentage of male users who have given a high rating to an action movie?" As the examples illustrate, class-level probabilities concern the proportion, or rate of generic events and conditions, rather than the attributes and links of individual entities. There are several applications of class-level models:

Statistical first-order Patterns. AI research into combining first-order logic and probability investigated in depth the representation of statistical patterns in relational structures [30, 4]. Often such patterns can be expressed as *generic statements* about the average member of a class, like "intelligent students tend to take difficult courses".

Knowledge Discovery Dependencies provide valuable insights in themselves. For instance, a web search manager may wish to know whether if a user searches for a video in Youtube for a product, they are also likely to search for it on the web.

Relevance Determination Once dependencies have been established, they can be used as a relevance filter for focusing further network analysis only on statistically significant associations. For example, the classification and clustering methods of Sun and Han [84] for heterogeneous networks assume that a set of "metapaths" have been found that connect link types that are associated with each other.

Query Optimization The Bayes net model can also be used to estimate relational statistics, the frequency with which statistical patterns occur in the database [77]. This kind of statistical model can be applied for database query optimization [24].

4.2 Approach and Contribution

Algorithms of structure learning for directed graphical models with link uncertainty have been previously described [21]. However to my best knowledge, no implementations of such structure learning algorithms for directed graphical models are available. The system FactorBase builds on the state-of-the-art Bayes net learner for relational data. Implementations exist for other types of graphical models, specifically Markov random fields (undirected models) [15] and dependency networks (directed edges with cycles allowed) [58]. Structure learning programs for Markov random fields are provided by Alchemy [15] and Khot et al [44]. Khot et al. use boosting to provide a state-of-the-art dependency network learner. **None** of these programs are able to return a result on half of the datasets because they are too large. I restrict the scope of this chapter to directed graphical models and do not go further into undirected model. For an extensive comparison of the learn-and-join Bayes net learning algorithm with Alchemy please see [76].

As introduced in section 1.2.1, I focus on building a Bayes net model for relational data, using the Parametrized Bayes nets (PBNs) in this chapter. The nodes in a PBN are constructed with functors and first-order variables (e.g., $gender(X)$ may be a node).

Contributions Scalable analysis for relational data with multiple link types is a challenging problem. In this chapter, I describe a method for learning a Bayes net that captures simultaneously correlations between link types, link features, and attributes of nodes. My method conducts bottom-up search through the lattice of table joins hierarchically. Dependencies (Bayes net edges) discovered on smaller joins are propagated to larger joins. The different table joins integrate information about the presence or absence of relationships across multiple tables. This is an extension of the current state-of-the-art Bayes net learning algorithm for relational data with multiple links given link uncertainty[76].

Such a Bayes net provides a succinct graphical representation of complex statistical-relational patterns. Previous work on learning Bayes nets for relational data was restricted to correlations among attributes **given the existence of links** [76]. The larger class of correlations examined in my new approach includes two additional kinds:

1. Dependencies between different types of links.
2. Dependencies among node attributes given the *absence* of a link between the nodes.

4.3 Bayes Nets for Relational Data

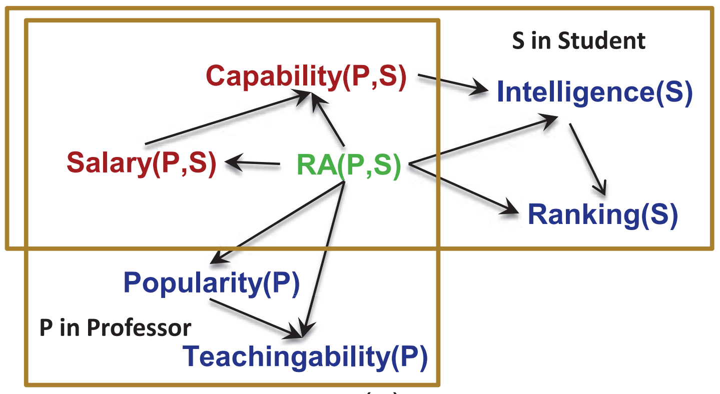
I review some key concepts of the Bayes Nets as I discussed in the section 1.2.1. A **Bayes Net (BN)** is a directed acyclic graph (DAG) whose nodes comprise a set of random variables and conditional probability parameters. For each assignment of values to the nodes, the joint probability is specified by the product of the conditional probabilities, $P(child|parent\_values)$. A **Parametrized random variable** is of the form $f(X_1, \ldots, X_a)$, where the populations associated with the variables are of the appropriate type for the functor. A **Parametrized Bayes Net** (PBN) is a Bayes net whose nodes are Parametrized random variables [65], e.g, Figure 4.1[1]. If a Parametrized random variable appears in a Bayes net, I often refer to it simply as a node.

4.4 Bayes Net Learning With Link Correlation Analysis

Constructing a Bayes Net (BN) for a relational database is very challenging. For single-table data, Bayes Net learning has been considered as a benchmark application for precomputing sufficient statistics [57, 54]. Given an assignment of values to its parameters, a Bayes Net represents a joint distribution over both attributes and relationships in a relational database.

To learn correlations between link types, I need to provide the Bayes net with data about when links are present *and* when they are absent. FactorBase extends the previously existing learn-and-join method (LAJ), which is the state-of-the-art for Bayes net learning in relational databases [76] to support the analysis with link uncertainty. The LAJ method

[1]Repeat the Figure 2.4 for easy review.



(a)

Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1.00
4	T	high	0.45
5	T	high	0.36
3	T	high	0.18
3	T	low	0.20
2	T	low	0.40
1	T	low	0.40
2	T	med	0.22
3	T	med	0.44
1	T	med	0.33

(b)

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	n/a	F	n/a
5	4	T	high
4	5	T	high
2	3	T	high
1	3	T	low
2	2	T	low
2	1	T	low
2	2	T	med
4	3	T	med
3	1	T	med

(c)

Figure 4.1: (a) Bayesian network for the University domain. I omit the *Registered* relationship for simplicity. The network was learned from the University dataset [67]. (b) Conditional Probability table $Capability(P,\mathcal{S})\_CPT$, for the node $Capability(P,\mathcal{S})$. Only value combinations that occur in the data are shown. This is an example of a factor table. (c) Contingency Table $Capability(P,\mathcal{S})\_CT$ for the node $Capability(P,\mathcal{S})$ and its parents. Both CP and CT tables are stored in an RDBMS.

takes as input a contingency table for the entire database, so I can apply it with both **link analysis on** and **link analysis off** to obtain two different BN structures for each database.

The experiment in section 4.5 is the first evaluation of the LAJ method with link analysis on. I use the LAJ implementation provided by system FactorBase. I score all learned graph structures using the same full contingency table with link analysis on, so that the scores are comparable. The idea is that turning link analysis on should lead to a different structure that represents correlations, involving relationship variables, that exist in the data.

4.4.1 Hierarchical Search: learn-and-join method (LAJ)

The key idea of the LAJ algorithm can be explained in terms of the **table join lattice** illustrated in Figure 4.2[2]. The user chooses a single-table Bayes net learner. The learner is applied to table joins of size 1, that is, regular data tables. Then the learner is applied to table joins of size $s, s+1, \ldots$, with the constraint that larger join tables inherit the

[2]Repeat the Figure 3.1 for easy review.

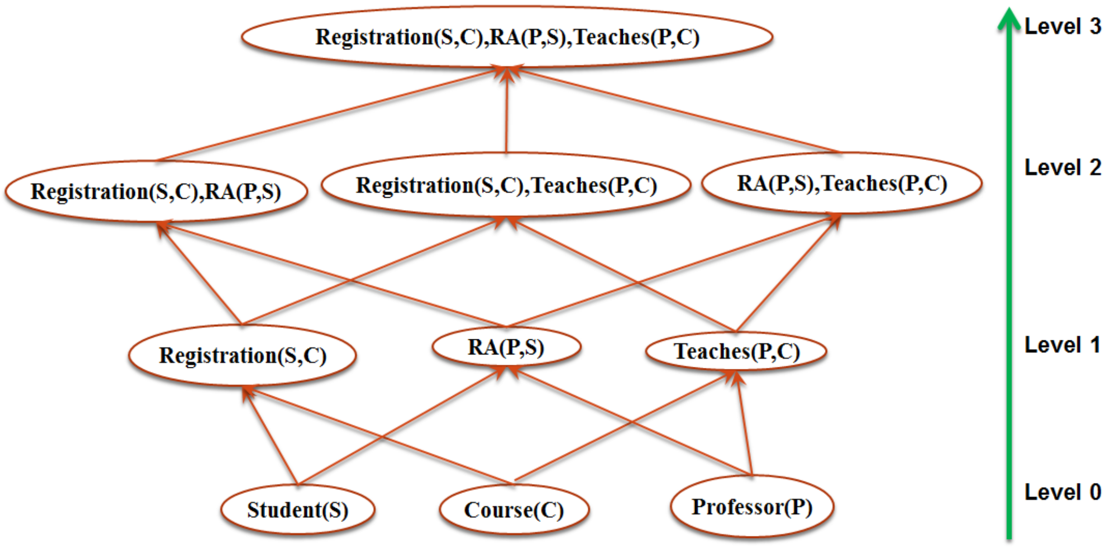


Figure 4.2: A lattice of relationship sets for the university schema of Figure 1.2. The arrows indicate the direction of edge propagation from lower level to upper level.

absence or presence of learned edges from smaller join tables. These edge constraints are implemented by keeping a global cache of forbidden and required edges.

Recall that the (natural) join of two or more tables, written $T_1 \bowtie T_2 \cdots \bowtie T_k$ is a new table that contains the rows in the Cartesian products of the tables whose values match on common fields. A table join corresponds to logical conjunction [88]. Say that a join table J is a **subjoin** of another join table J' if $J' = J \bowtie J^*$ for some join table J^*. If J is a subjoin of J', then the fields (columns) of J are a subset of those in J'. The subjoin relation defines the table join lattice. The moral of the learn-and-join algorithm is that join tables should inherit edges between descriptive attributes from their subjoins. This gives rise to the following constraints for two attributes X_1, X_2 that are both contained in some subjoin of J.

1. X_1 and X_2 are adjacent in a BN B_J for J if and only if they are adjacent in a BN for some subjoin of J.

2. if all subjoin BNs of J orient the link as $X_1 \to X_2$ resp. $X_1 \leftarrow X_2$, then B_J orients the link as $X_1 \to X_2$ resp. $X_1 \leftarrow X_2$.

Algorithm 4 provides pseudocode for the previous learn-and-join algorithm (LAJ) [75].

Examples. Consider the lattice shown in Figure 4.2. Suppose that the Bayes net associated with the relationship $\mathit{Registration}(\mathcal{S}, \mathbb{C})$ contains an edge

$$\mathit{difficulty}(\mathbb{C}) \rightarrow \mathit{intelligence}(\mathcal{S}).$$

Then the edge $\mathit{difficulty}(\mathbb{C}) \rightarrow \mathit{intelligence}(\mathcal{S})$ must be present in the Bayes net associated with the larger relationship set $\mathit{Registration}(\mathcal{S}, \mathbb{C})$, $\mathit{Teaches}(P, \mathbb{C})$. If the edge is contained in neither of the graphs associated with $\mathit{Registration}(\mathcal{S}, \mathbb{C})$, and $\mathit{Teaches}(P, \mathbb{C})$, it must not be present in the graph associated with the $\mathit{Registration}(\mathcal{S}, \mathbb{C})$, $\mathit{Teaches}(P, \mathbb{C})$.

Schema Invariance. The same data set may be represented under different ER schemas for various reasons, such as efficiency, data quality, and usability. Some relational learning algorithms tend to vary quite substantially over the choice of schema, which complicates their off-the-shelf application [64]. I circumvent the limitations of classical likelihood measures by using a relational pseudo-likelihood measure for Bayes nets [74]. The pseudo likelihood is equivalent to an expression defined in terms of a single (hyper) population has the important consequence that it is invariant under syntactic equivalence transformations of the database. For instance, database normalization operations may move information about a descriptive attribute from one table to another [40]. Thus, the objective function of the struture learning algorithm shown in Algorithm 4 is schema invariant.

Algorithm 4: Pseudocode for previous Learn-and-Join Structure Learning for Lattice Search.

```
Input: Database $\mathcal{D}$ with $E_1, ..E_e$ entity tables, $R_1, ...R_r$ Relationship tables,
Output: Bayes Net 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 all entity and relationship tables as variables to $G$. Add a boolean
    indicator for each relationship table to $G$.
 2: Econstraints = $\emptyset$ [Required and Forbidden edges]
 3: for m=1 to e do
 4:    Econstraints += Get-Constraints(PBN($E_m$ , $\emptyset$))
 5: end for
 6: for m=1 to r do
 7:    $N_m$ := natural join of $R_m$ and entity tables linked to $R_m$
 8:    Econstraints += Get-Constraints(PBN($N_m$, Econstraints))
 9: end for
10: for each $N_i$ and $N_j$ with a foreign key in common do
11:    $K_{ij}$ := join of $N_i$ and $N_j$
12:    Econstraints += Get-Constraints(PBN($K_{ij}$, Econstraints))
13: end for
14: return Bayes Net defined by Econstraints.
```

4.4.2 Extend LAJ for Multiple Links

To extend the learn-and-join algorithm for multiple link analysis, I replace the natural join in line 7 by the extended join (more precisely, by the contingency table derived from both positive relationship and negative relationship computed in section 3.4). The natural join contains only tuples that appear in all relationship tables. Compared to the joint contingency table, this corresponds to considering only rows where the par-Var (I also refer such as link indicator) have the value T (e.g. RA(P,$\mathcal{S}$) = T).

When the propositional Bayes net learner is applied to such a table, the link indicator variable appears like a constant. Therefore the BN learner cannot find any correlations between the link indicator variable and other nodes, nor can it find correlations among attributes conditional on the link indicator variable being F. Thus the previous LAJ algorithm finds only correlations between entity attributes conditional on the existence of a relationship. In sum, hierarchical search with link correlations can be described as follows.

1. Run the previous LAJ algorithm (Algorithm 4) using natural joins.
2. Starting with the constraints from step 1, run the LAJ algorithm where joint contingency table replace natural joins. That is, for each relationship set shown in the lattice of Figure 4.2, apply the single-table Bayes net learner to joint contingency table for the relationship set.

4.5 Evaluation

I used the same FactorBase system configurations with previous chapters. For the parameter learning of the PBN, I utilize the Parameter Manager in section 2.4.2. I also make the standard assumption that the database is complete [15, 43]: For each individual (node) observed, the database lists its attributes, and for each pair of observed individuals (nodes), the database specifies which links hold between them. As of the single-table Bayes Net learner, I made use of the following implementation: GES search [9] with the BDeu score as implemented in version 4.3.9-0 of CMU's Tetrad package (structure prior uniform, ESS=10; [87]).

I used seven benchmark real-world databases as shown in Table 4.1. For detailed descriptions and the sources of the databases, please refer to Table 3.2 in section 3.5. I compared the following settings utilizing different setting:

1. The extended LAJ method with link correlations off (Algorithm 4 utilizing natural joins).
2. The extended LAJ method that has the potential to find link correlations (Algorithm 4 utilizing join contingency tables).

Table 4.1: Datasets characteristics.
#Tuples = total number of tuples over all tables in the dataset.

Dataset	**#Relationship Tables/ Total**	**#Self Relationships**	**#Tuples**	**#Attributes**
Movielens	1 / 3	0	1,010,051	7
Mutagenesis	2 / 4	0	14,540	11
Financial	3 / 7	0	225,932	15
Hepatitis	3 / 7	0	12,927	19
IMDb	3 / 7	0	1,354,134	17
Mondial	2 / 4	**1**	870	18
UW-CSE	2 / 4	**2**	712	14

4.5.1 Structure Learning Times

Table 4.2 provides the model search time for structure learning with link analysis on and off. Given the sufficient statistics structure learning is fast, even for the largest contingency table IMDb (less than 10 minutes run-time). With link analysis on, structure learning takes more time as it processes more information. As expected in both modes, the run-time for building the contingency tables (refer to the Table 3.3 in section 3.5.2) dominates the structure learning cost. For the Mondial database, there is no case where all relationship variables are simultaneously true, so with link analysis off the contingency table is empty.

Table 4.2: Model Structure Learning Time in seconds.

Dataset	**Link Analysis On**	**Link Analysis Off**
Movielens	1.53	1.44
Mutagenesis	1.78	1.96
Financial	96.31	3.19
Hepatitis	416.70	3.49
IMDb	551.64	26.16
Mondial	190.16	N/A
UW-CSE	2.89	2.47

4.5.2 Statistical Scores

I report two model metrics, the log-likelihood score, and the model complexity as measured by the number of parameters. The **log-likelihood** is denoted as $L(\hat{G}, \mathbf{d})$ where $\hat{G}$ is the BN G with its parameters instantiated to be the maximum likelihood estimates given the dataset $\mathbf{d}$, and the quantity $L(\hat{G}, \mathbf{d})$ is the log-likelihood of $\hat{G}$ on $\mathbf{d}$. I use the relational log-likelihood score defined in [74], which differs from the standard single-table Bayes net likelihood only by replacing counts by frequencies so that scores are comparable across different nodes and databases. To provide information about the qualitative graph structure learned, I report edges learned that point to a relationship variable as a child. Such edges can be learned only with link analysis on. I distinguish edges that link relationship variables—R2R—and that link attribute variables to relationships—A2R. For instance, in the most intuitive Financial dataset, there are two R2R edges and nine A2R edges, as shown in Figure 4.3. And for the IMDb dataset, eleven A2R edges were captured in the learned BN struture with link analysis on which is shown in Figure 4.4.

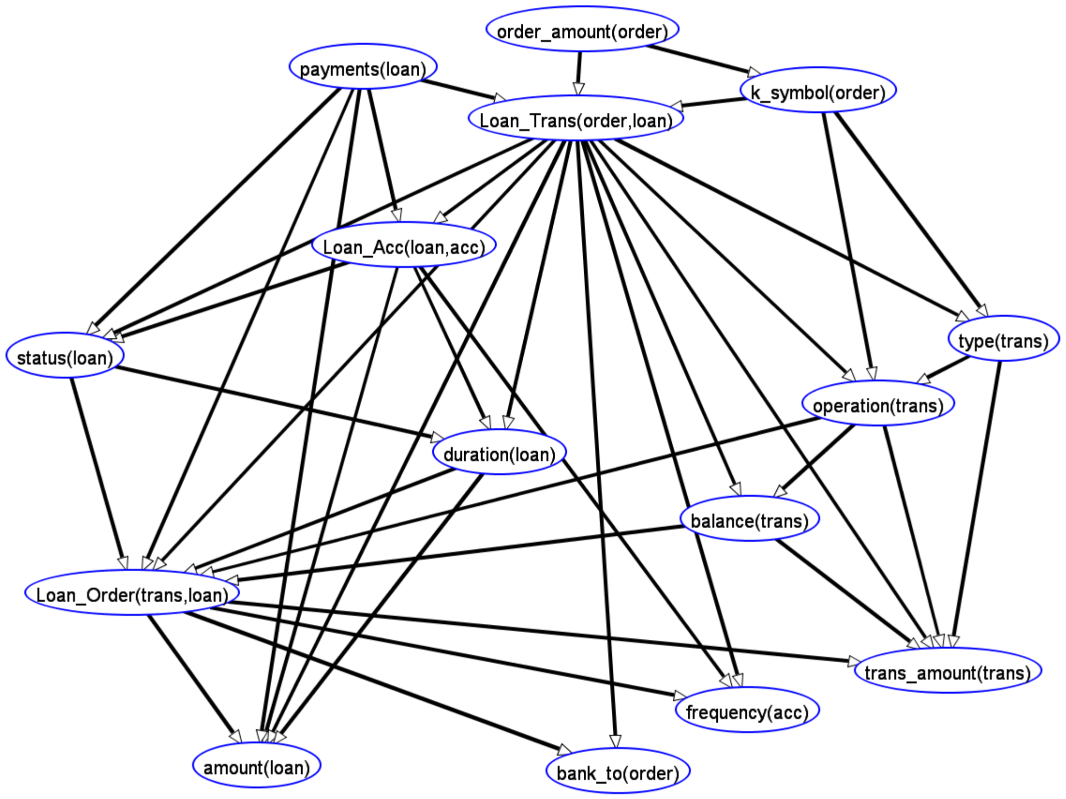


Figure 4.3: The learned Bayes Net for Financial dataset with link analysis on.

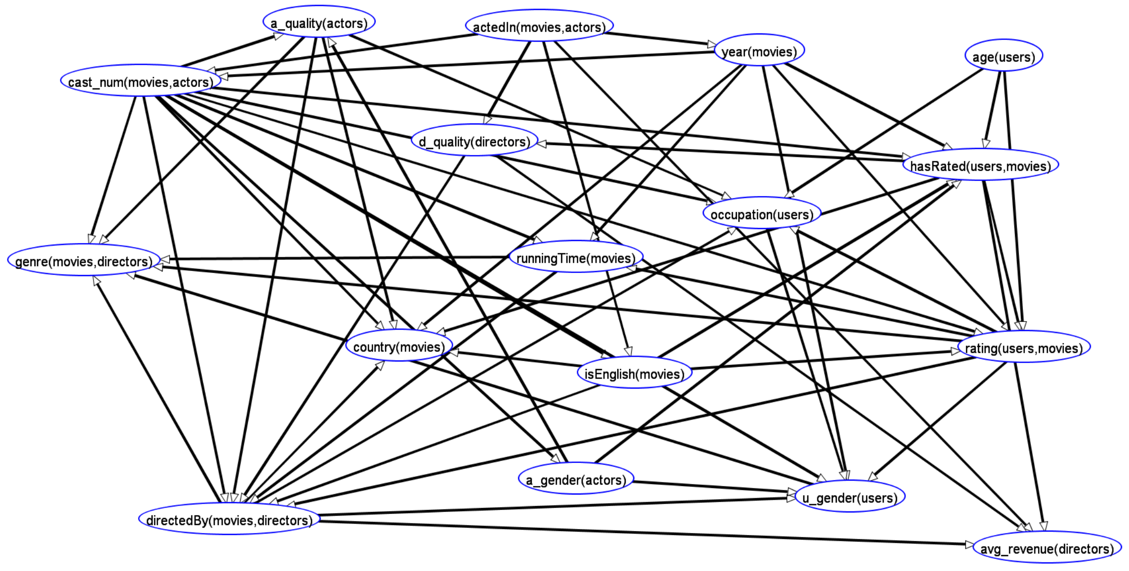


Figure 4.4: The learned Bayes Net for IMDb dataset with link analysis on.

Structure learning can use the new type of dependencies to find a better, or at least different, trade-off between model complexity and model fit. On two datasets (IMDb and Financial), link analysis leads to a superior model that achieves better data fit with fewer parameters. These are also the datasets with the most complex relational schemas (see Table 4.1). On IMDb in particular, considering only positive links leads to a very poor structure with a huge number of parameters. On four datasets, extra sufficient statistics lead to different trade-offs: On MovieLens and Mutagenesis, link analysis leads to better data fit but higher model complexity, and the reverse for Hepatitis and UW-CSE.

4.6 Conclusion

The model described in this chapter captures a wider class of correlations that involve uncertainty about the link structure. I extend the state-of-the-art algorithm for learning correlations among link types and node attributes in relational data that represent complex heterogeneous network with many attributes and link types. The link correlations are represented in a Bayes net structure. This provides a succinct graphical way to display relational statistical patterns and support powerful probabilistic inferences. Statistical measures indicate that Bayes net methods succeed in finding relevant correlations.

Table 4.3: Comparison of Statistical Performance of Bayesian Network Learning.

Movielens	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	-4.68	**164**	0	0
Link Analysis On	**-3.44**	292	0	3

Mutagenesis	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	-6.18	**499**	0	0
Link Analysis On	**-5.96**	721	1	5

Financial	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	-10.96	11,572	0	0
Link Analysis On	**-10.74**	**2433**	2	9

Hepatitis	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	**-15.61**	962	0	0
Link Analysis On	-16.58	**569**	3	6

IMDb	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	-13.63	181,896	0	0
Link Analysis On	**-11.39**	**60,059**	0	11

Mondial	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	N/A	N/A	N/A	N/A
Link Analysis On	-18.2	339	0	4

UW-CSE	log-likelihood	#Parameter	R2R	A2R
Link Analysis Off	**-6.68**	305	0	0
Link Analysis On	-8.13	**241**	0	2

Chapter 5

Instance Level Learning of Relational Dependency Networks

Learning graphical models is one of the main approaches to extending machine learning for relational data. Two major classes of graphical models are Bayesian networks (BNs) [62] and dependency networks (DNs) [33]. In chapter 4, I discussed the generative modeling with Bayes net by learning a model of the joint probability distribution of all input random variables. Dependency networks are well suited for the task of discriminative predicting preferences such as classification problem to predict the label y given all inputs x. In this chapter, I describe a new approach for instance level dependency networks learning: first learn a Bayes net, then convert that network to a dependency network. This hybrid approach combines the speed of learning Bayes net with the advantages of dependency network inference for relational data. The experiments show that the hybrid learning algorithm can produce dependency networks for large and complex databases, up to one million records and 19 predicates. The predictive accuracy of the resulting networks is competitive with those from state-of-the-art function gradient boosting methods but scales substantially better than the boosting methods.

5.1 Introduction

The hybrid approach combines different strengths of Bayesian networks and dependency networks for relational learning. The special strength of Bayesian networks is scalability in learning [59, Sec.8.5.1],[42]. Bayesian networks offer closed-form parameter estimation via the maximum likelihood method, and therefore closed-form model evaluation. Model evaluation is the computationally most expensive part of relational learning, as it requires combining information from different related tables, which involves expensive table joins.

In contrast, a strength of relational dependency networks is that they support inference in the presence of cyclic dependencies[59, 58]. Cyclic dependencies occur when a relational

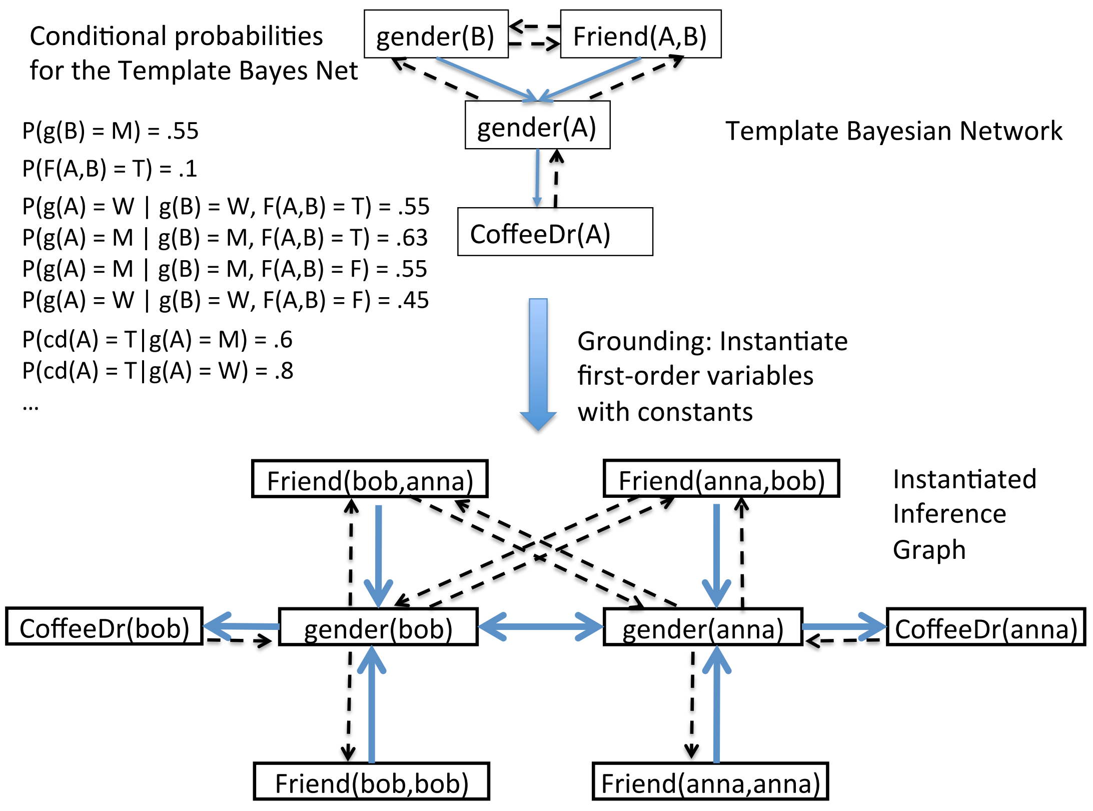

Conditional probabilities
for the Template Bayes Net
gender(B)
Friend(A,B)
gender(A)
CoffeeDr(A)
Template Bayesian Network
P(g(B) = M) = .55
P(F(A,B) = T) = .1
P(g(A) = W | g(B) = W, F(A,B) = T) = .55
P(g(A) = M | g(B) = M, F(A,B) = T) = .63
P(g(A) = M | g(B) = M, F(A,B) = F) = .55
P(g(A) = W | g(B) = W, F(A,B) = F) = .45
P(cd(A) = T|g(A) = M) = .6
P(cd(A) = T|g(A) = W) = .8
…
Grounding: Instantiate
first-order variables
with constants
Friend(bob,anna)
Friend(anna,bob)
Instantiated
Inference
Graph
CoffeeDr(bob)
gender(bob)
gender(anna)
CoffeeDr(anna)
Friend(bob,bob)
Friend(anna,anna)

Figure 5.1: A Bayesian/dependency template network (top) and the instantiated inference graphs (bottom). By convention, predicates (Boolean functors) are capitalized. Edges from the BN template are solid blue, while edges added by the BN-to-DN transformation are dashed black. The edge set in the DN comprises both solid and dashed arrows. Note that although the template BN (top) is acyclic, its instantiation (bottom) features a bi-directed edge between $gender(bob)$ and $gender(anna)$.

dataset features auto-correlations, where the value of an attribute for an individual depends on the values of the same attribute for related individuals. Figure 5.1 provides an example. It is difficult for Bayesian networks to model auto-correlations because by definition, the graph structure of a Bayesian network must be acyclic [16, 86, 25]. Because of the importance of relational auto-correlations, dependency networks have gained popularity since they support reasoning about cyclic dependencies using a directed graphical model.

These advantages of the hybrid approach are specific to relational data. For propositional (i.i.d.) data, which can be represented in a single table, there is no problem with cyclic dependencies, and the acyclic constraint of Bayesian networks can actually make probabilistic inference more efficient [37]. Also, the closed-form model evaluation of Bayesian networks is relatively less important in the single-table case, because iterating over the rows of a single data table to evaluate a dependency network is relatively fast compared to iterating over ground atoms in the relational case, where they are stored across several tables.

My approach extends the moralization approach to relational structure learning, which learns an undirected Markov model by first learning a Bayesian network structure, then converts the network structure to a Markov logic structure, without parameters [42, 76]. That approach also combines the scalable learning of Bayesian networks with the support undirected models offer for inference with auto-correlation. The present work extends this by also computing the dependency network parameters from the learned Bayesian network. The previous work used only the Bayesian network structure. The theoretical analysis shows that the learned dependency networks provide different predictions from Markov networks.

Contributions I make three main contributions:

1. A faster approach for learning relational dependency networks: first learn a Bayesian network, then convert it to a dependency network.
2. A closed-form log-linear discriminative model for computing the relational dependency network parameters from Bayesian network structure and parameters.
3. Necessary and sufficient conditions for the resulting network to be **consistent**, defined as the existence of a single joint distribution that induces all the conditional distributions defined by the dependency network [33].

5.2 Bayesian Networks and Relational Dependency Networks

I review dependency networks and their advantages for modelling relational data. I assume familiarity with the basic concepts of Bayesian networks [62].

5.2.1 Dependency networks and Bayesian networks

The structures of both Bayesian networks and dependency networks are defined by a directed graph whose nodes are random variables. Bayesian networks must be acyclic, while dependency networks may contain cycles, including the special case of bi-directed edges. For both networks, the parameters are conditional distributions over the value of a node given its parents. The two types differ in the influence of a node's children, however. In a Bayesian network, a node is only independent of all other nodes given an assignment of values to its parents, its children, and the co-parents of its children, whereas in a dependency network a node is independent given an assignment of values to only its parents. In graphical model terms, the **Markov blanket** of a node in a dependency network, the minimal set of nodes such that assigning them values will make this node independent of the rest of the network, is simply its parents.[1] For a Bayesian network, the Markov blanket is the node's parents, children, and the co-parents of its children.

Consequently, a conditional probability in a dependency network effectively specifies the probability of a node value given an assignment of values to *all* other nodes. Following Heckerman *et al.* [33], I refer to such conditional probabilities as **local probability distributions**.

5.2.2 Relational Dependency Networks

As described in section 1.2, I present the relational case using the parametrized random variable notation [45]. Relational dependency networks [59] extend dependency networks to model distributions over multiple populations. A functor is a symbol denoting a function or predicate. Each functor has a set of values (constants) called the **domain** of the functor. Functors with boolean ranges are called **predicates** and their name is capitalized. I consider only functors with finite domains. An expression $f(\tau_1, \ldots, \tau_k)$, where f is a functor and each τ_i is a first-order variable or a constant, is a **Parametrized Random Variable** (PRV).

A directed acyclic graph whose nodes are PRVs is a **parametrized Bayesian network structure**, while a general (potentially cyclic) directed graph whose nodes are PRVs is a **relational dependency network structure** (RDN). A Bayesian network structure or relational dependency network structure augmented with the appropriate conditional probabilities is respectively a **Bayesian network template** or **relational dependency network template**. Note that the RDN templates that I define in this chapter have the same Markov blanket as the Bayesian network templates from which they are derived but a different edge structure and probabilities. Algorithm 5, defined in Section 5.3, converts the probabilities of the Bayesian template to their counterparts in the relational dependency template.

RDNs extend dependency networks from i.i.d. to relational data via knowledge-based model construction [59]: The first-order variables in a template RDN graph are instantiated for a specific domain of individuals to produce an *instantiated* or *ground* propositional DN graph, the **inference graph**. Figure 5.1 gives a dependency network template and its inference graph. Given an edge in the template RDN, instantiating both the parent and the child of the edge with the same grounding produces an edge in the inference graph. An example local probability distribution for the graph in Figure 5.1 (abbreviating functors) is

$$P(g(anna)|g(bob), CD(anna), F(anna, bob), F(bob, anna), F(anna, anna)).$$

Language Bias. The general definition of a parametrized random variable allows PRVs to contain constants as well as population variables. Another language extension is to

[1]For this reason Hofmann and Tresp originally used the term "Markov blanket networks" for dependency networks [36].

Algorithm 5: Computing Features and Weights for Template Dependency Network.

```
Input: Template Bayesian Network $B$ (Structure and Parameters)
Output: A List of Relevant Features; a Weight for each Feature
1: for each target node $T$ do
2:    initialize $Feature\_Weight\_List(T)$ as the empty list
3:    for each $U$ in $\{T \cup \mathrm{Ch}(T)\}$ do
4:       for each value $u$ of the child node $U$ do
5:          for each vector of parent values $\vec{u}_{pa}$ do
6:             $Feature\ F := (U = u, \mathrm{Pa}(U) = \vec{u}_{pa})$
7:             $FeatureWeight\ w := \ln \theta(U = u|\mathrm{Pa}(U) = \vec{u}_{pa})$
8:             if the Feature $F$ does not contain a false relationship other than $T$ then
9:                add $(F, w)$ to $Feature\_Weight\_List(T)$
10:            end if
11:         end for
12:      end for
13:   end for
14: end for
15: return $Feature\_Weight\_List(T)$
```

allow parametrized random variables to be formed with aggregate functions, as described by Kersting and deRaedt [41]. For example, it is possible to use a functor that returns the number of friends of a generic person $\mathbb{A}$. The main contribution of this paper, the relational BN-to-DN conversion method, can be used whether the parametrized random variables contain constants, aggregates, or only first-order variables. A common restriction to simplify model structure learning is to exclude constants (e.g. [20, 15]). Friedman *et al.* investigated learning directed graphical models with aggregate functions [20].

5.3 Learning Relational Dependency Networks via Bayesian Networks

The algorithm for rapidly learning relational dependency networks (Figure 5.2) begins with any relational learning algorithm for Bayesian networks. Using the resulting Bayesian network as a template, I then apply a simple, fast transformation to obtain a relational dependency template. Finally I apply a closed-form computation to derive the dependency network inference graph parameters from the Bayesian structure and parameters.

5.3.1 BN-to-DN structure conversion

Converting a Bayesian network structure to a dependency network structure is simple: for each node, add an edge pointing to the node from each member of its BN Markov blanket [33]. The result contains bidirectional links between each node, its children, and

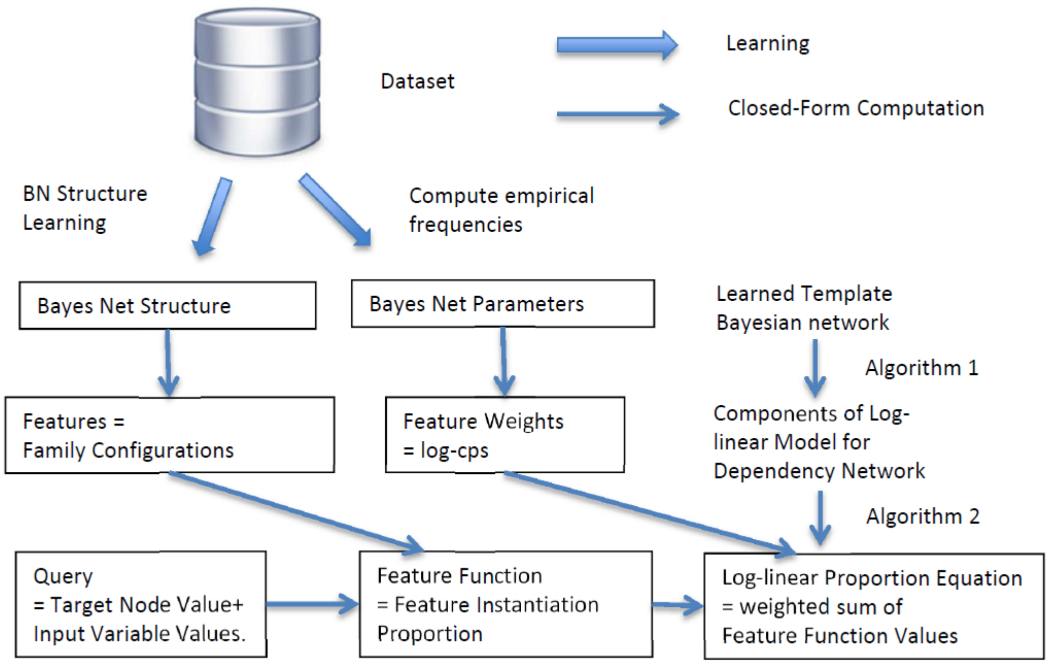

Dataset
Learning
Closed-Form Computation
BN Structure Learning
Compute empirical frequencies
Bayes Net Structure
Bayes Net Parameters
Learned Template Bayesian network
Algorithm 1
Features = Family Configurations
Feature Weights = log-cps
Components of Log-linear Model for Dependency Network
Algorithm 2
Query = Target Node Value+ Input Variable Values.
Feature Function = Feature Instantiation Proportion
Log-linear Proportion Equation = weighted sum of Feature Function Values

Figure 5.2: The program flow for computing local probability distributions from a template Bayesian network. Features and weights are computed from the Bayesian network. Feature function values are computed for each query.

its co-parents (nodes that share a child with this one). This is equivalent to the standard moralization method for converting a BN to an undirected model [15], except that the dependency network contains bi-directed edges instead of undirected edges. Bidirected edges have the advantage that they permit assignment of different parameters to each direction, whereas undirected edges have only one parameter.

5.3.2 BN-to-DN parameter conversion

For propositional data, converting Bayesian network parameters to dependency network parameters is simple: apply the standard BN product formula and solve for the local probability distributions given Bayesian network parameters [72, Ch.14.5.2]. A **family** comprises a node and its parents. A **family configuration** specifies a value for a child node and each of its parents. For example, in the template of Figure 5.1 (top), one family is $\mathit{gender}(\mathbb{A}), \mathit{Friend}(\mathbb{A}, \mathbb{B}), \mathit{gender}(\mathbb{B})$ and one of its eight possible configurations is

$$\mathit{gender}(\mathbb{A}) = \mathrm{M}, \mathit{Friend}(\mathbb{A}, \mathbb{B}) = \mathrm{T}, \mathit{gender}(\mathbb{B}) = \mathrm{M}.$$

The Markov blanket of a target node comprises multiple families, one each for the target node and each of its children, so an assignment of values to the target's Markov blanket defines a unique configuration for each family. Hence in the propositional case the Markov blanket induces a *unique* log-conditional probability for each family configuration. The probability of a target node value given an assignment of values to the Markov blanket is then proportional to *the exponentiated sum of these log-conditional probabilities* [72, Ch.14.5.2].

With relational data, however, different family configurations can be simultaneously instantiated, *multiple times.* I generalize the propositional log-linear equation for relational data by replacing the unique log-conditional probability with the *expected* log-conditional probability that results from selecting an instantiation of the family configuration uniformly at random. The probability of a target node value given an assignment of values to the Markov blanket is then proportional to the exponentiated sum of the expected log-conditional probabilities. I describe the resulting closed-form equation in the next section.

5.4 The Log-linear Proportion Equation

I propose a log-linear equation, the **log-linear proportion equation** (lower right box of Figure 5.2), for computing a local probability distribution for a ground target node, T^*, given (i) a target value t for the target node, (ii) a complete set of values Λ^* for all ground terms other than the target node, and (iii) a template Bayesian network. The template structure is represented by functions that return the set of parent nodes of U, $\mathrm{Pa}(U)$, and the set of child nodes of U, $\mathrm{Ch}(U)$. The parameters of the template are represented by the conditional probabilities of a node U having a value u conditional on the values of its parents, $\theta(U = u|\mathrm{Pa}(U) = \vec{u}_{pa})$. A grounding γ substitutes a constant for each member of a list of first-order variables, $\{\mathbb{A}_1 = a_1, \ldots, \mathbb{A}_k = a_k\}$. Applying a grounding to a template node defines a fully ground target node: $\mathit{gender}(\mathbb{A})\{\mathbb{A} = \mathit{sam}\} = \mathit{gender}(\mathit{sam})$. These are combined in the following log-linear equation to produce a local probability distribution:

Definition 1 (The Log-Linear Proportion Equation)**.**

$$P(T^* = t|\Lambda^*) \quad \propto \quad \exp$$
$$\sum_{U} \sum_{u,\vec{u}_{pa}} \quad [\ln \theta(U = u|\mathrm{Pa}(U) = \vec{u}_{pa})] \quad \cdot \quad \mathrm{p^r}\left[\gamma; U = u, \mathrm{Pa}(U) = \vec{u}_{pa}; T^* = t, \Lambda^*\right]$$

where

U	varies over	$\{T\} \cup \mathrm{Ch}(T)$;
the singleton value u	varies over	*the range of* U;
the vector of values $\vec{u}_{pa}$	varies over	*the product of the ranges of* U'*s parents,*
		constrained to value t *for occurrences of* T;
$T^* = T\gamma$	is	*is the target grounding of template node* T;
and p^{r}	is	*the feature function, the family proportion.*

The family proportion p^{r} is computed as follows:

1. For a given family configuration ($U = u, \mathrm{Pa}(U) = \vec{u}_{pa}$), let the **family count**

$$\mathrm{n}\left[\gamma; U = u, \mathrm{Pa}(U) = \vec{u}_{pa}; T^* = t, \Lambda^*\right]$$

   be the number of instantiations that (a) satisfy the family configuration and the ground node values specified by $T^* = t, \Lambda^*$, and (b) are consistent with the equality constraint defined by the grounding γ.

2. The **relevant family count** n^r is 0 if the family configuration contains a false relationship (other than the target node), else equals the family count. It is common in statistical-relational models to restrict predictors to existing relationships only [21, 72].

3. The **family proportion** is the relevant family count, divided by the total sum of all relevant family counts for the given family:

$$\mathrm{p}^{\mathrm{r}}\left[\gamma; U = u, \mathrm{Pa}(U) = \vec{u}_{pa}; T^* = t, \Lambda^*\right] = \frac{\mathrm{n}^{\mathrm{r}}\left[\gamma; U = u, \mathrm{Pa}(U) = \vec{u}_{pa}; T^* = t, \Lambda^*\right]}{\sum_{u', \vec{u}'_{pa}} \mathrm{n}^{\mathrm{r}}\left[\gamma; U = u', \mathrm{Pa}(U) = \vec{u}'_{pa}; T^* = t, \Lambda^*\right]}$$

In the experiments, family counts and proportions are computed using exact counting methods (see Section 5.4.3 below).

5.4.1 Example and Pseudocode

Table 5.1 illustrates the computation of these quantities for predicting the gender of a new test instance (*sam*). Algorithm 5 shows pseudocode for the closed-form transformation of Bayesian network structure and parameters into features and weights for the dependency network. Algorithm 6 shows pseudocode for computing the scores defined by the log-linear Equation (1), given a list of weighted features and a target query.

The inner sum of Equation (1) computes the expected log-conditional probability for a family with child node U, when I randomly select a relevant grounding of the first-order variables in the family.

Algorithm 6: Computing local probability distributions, the parameters of the Inference Dependency Network.

```
Input: Feature-Weight List of Dependency Network, Query $P(T^* = t|\Lambda^*) =?$. $T$ is a
       template node, $T^* = T\gamma$ is the target grounding.
Output: Normalized log-linear score
1: initialize $score(T^* = t) := 0$
2: for each Feature $F = (U = u, \mathrm{Pa}(U) = \vec{u}_{pa})$ in $Feature\_Weight\_List(T)$ do
3:     Let $w$ be the weight listed for feature $F$
4:     {Next compute feature function.}
5:     $RelFamCnt(F) := \mathrm{n}^{\mathrm{r}}[\gamma; \mathsf{U} = u, \mathrm{Pa}(\mathsf{U}) = \vec{u}_{pa}; T^* = t, \Lambda^*]$
6:     $TotalRelFamCnt(U) := \sum_{u', \vec{u}'_{pa}} \mathrm{n}^{\mathrm{r}}\left[\gamma; \mathsf{U} = u', \mathrm{Pa}(\mathsf{U}) = \vec{u}'_{pa}; T^* = t, \Lambda^*\right]$
7:     $FamilyProportion$ $\mathrm{p}^{\mathrm{r}}(F) := RelFamCnt(F)/TotalRelFamCnt(U)$
8:     $score(T^* = t)$ += $\mathrm{p}^{\mathrm{r}} \cdot w$
9: end for
10: return Normalized scores for target node.
```

5.4.2 Discussion and Motivation

I discuss the key properties of the local distribution model, Equation (1).

Log-Linearity. The survey by Kimmig *et al.* [45] shows that most statistical-relational methods define log-linear models. The general form of a discriminative log-linear model [85] is that the conditional probability of a target variable value given input variable values is proportional to an exponentiated weighted sum of feature functions. A feature function maps a complete assignment of ground node values (= target value + input variables) to a real number. Khazemi *et al.* have shown that many relational aggregators can be represented by a log-linear model with suitable features [39]. Equation (1) instantiates this well-established log-linear schema as follows: The features of the model are the family configurations $(U = u, \mathrm{Pa}(U) = \vec{u}_{pa})$ where the child node is either the target node or one of its children. The feature weights are the log-conditional BN probabilities defined for the family configuration. The input variables are the values specified for the ground (non-target) nodes by the conjunction Λ^*. The feature functions are the family proportion p^{r}. Like other log-linear relational models, Equation 1 enforces parameter tying, where different groundings of the same family configuration receive the same weight [45].

Standardization. Using proportions as feature functions has the desirable consequence that the range of all feature functions is standardized to [0,1]. It is well-known that the number of instantiation counts in relational data can differ for different families, depending on the population variables they contain. This ill-conditioning causes difficulties for log-linear models because families with more population variables can have an exponentially

Table 5.1: Applying the log-linear proportion equation with the Bayesian network of Figure 5.1 to compute $P(gender(sam) = \mathrm{W}|\Lambda^*)$ and $P(gender(sam) = \mathrm{M}|\Lambda^*)$. Each row represents a feature/family configuration. For the sake of the example I suppose that the conjunction Λ^* specifies that Sam is a coffee drinker, has 60 male friends, and 40 female friends. CP is the conditional probability BN parameter of Figure 5.1 and $w \equiv \ln(CP)$.

Child Value u	Parent State $\vec{u}_{pa}$	CP	w	$\mathrm{p^r}$	$w \times \mathrm{p^r}$
$g(sam) = \mathrm{W}$	$g(B) = \mathrm{W}$, $F(sam, B) = \mathrm{T}$	0.55	-0.60	0.4	-0.24
$g(sam) = \mathrm{W}$	$g(B) = \mathrm{M}$, $F(sam, B) = \mathrm{T}$	0.37	-0.99	0.6	-0.60
$CD(sam) = \mathrm{T}$	$g(sam) = \mathrm{W}$	0.80	-0.22	1.0	-0.22
$CD(sam) = \mathrm{F}$	$g(sam) = \mathrm{W}$	0.20	-1.61	0.0	0.00
Sum ($\exp(Sum) \propto P(gender(sam) = \mathrm{W}|\Lambda^*)$)					-1.06
$g(sam) = \mathrm{M}$	$g(B) = \mathrm{W}$, $F(sam, B) = \mathrm{T}$	0.45	-0.80	0.4	-0.32
$g(sam) = \mathrm{M}$	$g(B) = \mathrm{M}$, $F(sam, B) = \mathrm{T}$	0.63	-0.46	0.6	-0.28
$CD(sam) = \mathrm{T}$	$g(sam) = \mathrm{M}$	0.60	-0.51	1.0	-0.51
$CD(sam) = \mathrm{F}$	$g(sam) = \mathrm{M}$	0.40	-0.92	0.0	0.00
Sum ($\exp(Sum) \propto P(gender(sam) = \mathrm{M}|\Lambda^*)$)					-1.11

higher impact on the score prediction [53]. Intuitively, counts tacitly conflate number of instantiations with degree of information. Proportions avoid such ill-conditioning.

Generalizing the Propositional Case. A useful general design principle is that relational learning should have propositional learning as a special case [49, 46]: When I apply a relational model to a single i.i.d. data table, it should give the same result as the propositional model. Equation 1 satisfies this principle. In the propositional case, an assignment of values to all nodes other than the target node specifies a *unique* value for each family configuration. This means that all the family counts n^r are either 0 or 1, hence all relevant proportions p^r are 0 or 1, depending on whether a family configuration matches the query or not. For a simple illustration, consider the edge $gender(\mathbb{A}) \to CoffeeDr(\mathbb{A})$. Since this edge concerns only the *Person* domain associated with the single population variable $\mathbb{A}$, I may view this edge as a propositional subnetwork. Suppose the query is $P(gender(sam) = \mathrm{W}|CoffeeDr(sam) = \mathrm{T})$. The only family configurations with nonzero counts are $gender(sam) = \mathrm{W}$ (count 1) and $CoffeeDr(sam) = \mathrm{T}), gender(sam) = \mathrm{W}$ (count 1). Equation (1) gives

$$P(g(sam) = \mathrm{W}|CD(sam) = \mathrm{T}) \propto$$
$$\exp\{\ln P(g(sam) = \mathrm{W}) + \ln P(CD(sam) = \mathrm{T})|g(sam) = \mathrm{W})\}.$$

This agrees with the propositional BN formula for a local conditional probability, which is the product of the BN conditional probabilities for the target node given its children, and the target node's children given their parents. This formula can be derived from the BN product rule for defining a joint probability [72, Ch.14.5.2]. In the simple two-node example, it can be derived immediately from Bayes' theorem:

$$P(g(sam) = \mathrm{W} | CD(sam) = \mathrm{T}) \propto$$
$$P(CD(sam) = \mathrm{T}) | g(sam) = \mathrm{W}) \times P(g(sam) = \mathrm{W}),$$

which agrees with the solution above derived from Equation (1). It may seem surprising that in predicting gender given coffee drinking, the model should use the conditional probability of coffee drinking given gender. However, Bayes' theorem states that $P(X|Y)$ is proportional to $P(Y|X)$. In the example, given that the BN model specifies that women are more likely to be coffee drinkers than men, the information that Sam is a coffee drinker raises the probability that Sam is a woman.

5.4.3 Complexity of Algorithms 5 and 6

The loops of Algorithm 5 enumerate every family configuration in the template Bayesian network exactly once. Therefore *computing features and weights takes time linear in the number of parameters of the Bayesian network.*

Evaluating the log-linear equation, as shown in Algorithm 6, requires finding the number of instantiations that satisfy a conjunctive family formula, given a grounding. This is an instance of the general problem of computing the number of instantiations of a formula in a relational structure. Computing this number is a well-studied problem with highly efficient solutions [89, 78].

A key parameter is the number m of first-order variables that appear in the formula. A loose upper bound on the complexity of counting instantiations is d^m, where d is the maximum size of the domain of the first-order variables. Thus counting instantiations has parametrized polynomial complexity [19], meaning that if m is held constant, counting instantiations requires polynomially many operations in the size of the relational structure (i.e., the size of $T^* = t, \Lambda^*$ in Equation (1)). For varying m, the problem of computing the number of formula instantiations is #P-complete [16, Prop.12.4].

5.5 Consistency of the Derived Dependency Networks

A basic question in the theory of dependency networks is the *consistency* of the local probabilities. Consistent local probabilities ensure the existence of a single joint probability distribution p that induces the various local conditional probability distributions P for each node

$$P(T^* = t|\Lambda^*) \propto p(T^* = t, \Lambda^*)$$

for all target nodes T^* and query conjunctions Λ^* [33].

I present a precise condition on a template Bayesian network for its resulting dependency network to be consistent and the implications of those conditions. I define an **edge-consistent template Bayesian network** to be a network for which every edge has the same set of population variables on both nodes.

Theorem 1. *A template Bayesian network is edge-consistent if and only if its derived dependency network is consistent.*

The proof of this result is complex, so I present it in an appendix. Intuitively, in a joint distribution, the correlation or potential of an edge is a single fixed quantity, whereas in Equation (1), the correlation is adjusted by the size of the relational neighbourhood of the target node, which may be either the child or the parent of the edge. If the relational neighborhood size of the parent node is different from that of the child node, the adjustment makes the conditional distribution of the child node inconsistent with that of the parent node. The edge-consistency characterization shows that the inconsistency phenomenon is properly relational, meaning it arises when network structure contains edges that relate parents and children from different populations.

The edge-consistency condition required by this theorem is quite restrictive: Very few template Bayesian networks will have exactly the same set of population variables on both sides of each edge.[2] Therefore relational template Bayesian networks, which have multiple population variables, will most often produce inconsistent dependency networks. Previous work has shown that dependency networks learned from data are almost always inconsistent but nonetheless provide accurate predictions using ordered pseudo-Gibbs sampling [33, 59, 52] or Generative Stochastic Networks [6, Sec.3.4].

5.6 Empirical Evaluation: Design and Datasets

There is no obvious baseline method for the RDN learning method because mine is the first work that uses the approach of learning an RDN via a Bayesian network. Instead I benchmark against the performance of a different approach for learning RDNs, which uses an ensemble learning approach based on functional gradient boosting. Boosted functional gradient methods have been shown to outperform previous methods for learning relational dependency networks [43, 58].

[2]A commonly used weaker condition is range-restriction: that the population variables in the child node should be contained in the population variables of its parents [41], but not vice versa as with edge-consistency.

Datasets I used six benchmark real-world databases with the FactorBase system. Summary statistics are given in Table 5.2. For more details please see the references in [76].

Methods Compared Functional gradient boosting is a state-of-the-art method for applying discriminative learning to build a generative graphical model. The local discriminative models are ensembles of relational regression trees [43]. Functional gradient boosting for relational data is implemented in the Boostr system [44]. For functors with more than two possible values, I followed [43] and converted each such functor to a set of binary predicates by introducing a predicate for each possible value. I compared the following methods:

RDN_Bayes my method: Learn a Bayesian network, then convert it to a relational dependency network.

RDN_Boost The RDN learning mode of the Boostr system [58].

MLN_Boost The MLN learning mode of the Boostr system. It takes a list of target predicates for analysis. I provide each binary predicate in turn as a single target predicate, which amounts to using MLN learning to construct an RDN. This RDN uses a log-linear model for local probability distributions that is derived from Markov Logic Networks.

I used the default Boostr settings. I experimented with alternative settings but they did not improve the performance of the boosting methods.

Prediction Metrics I follow [43] and evaluate the algorithms using conditional log likelihood (CLL) and area under the precision-recall curve (AUC-PR). AUC-PR is appropriate when the target predicates features a skewed distribution as is typically the case with relationship predicates. For each fact $T^* = t$ in the test dataset, I evaluate the accuracy of the predicted local probability $P(T^* = t|\Lambda^*)$, where Λ^* is a complete conjunction for all ground terms other than T^*. Thus Λ^* represents the values of the input variables as specified by the test dataset. CLL is the average of the logarithm of the local probability for each ground truth fact in the test dataset, averaged over all test predicates. For the gradient boosting method, I used the AUC-PR and likelihood scoring routines included in Boostr.

Both metrics are reported as means and standard deviations over all binary predicates. The learning methods were evaluated using 5-fold cross-validation. Each database was split into 5 folds by randomly selecting entities from each entity table, and restricting the relationship tuples in each fold to those involving only the selected entities (i.e., subgraph sampling [76]). The models were trained on 4 of the 5 folds, then tested on the remaining one.

Table 5.2: Learning Time. The total learning time for constructing a relational dependency network from an input database. Only partial boosting learning times are reported for the larger databases MovieLens(1M) and IMDb—see text for details. Spread is reported as coefficient of variation (CV—standard deviation / mean). PRV = Parametrized Random Variable.

			RDN_Bayes		RDN_Boost		MLN_Boost	
Dataset	kTuple	PRVs	(s)	CV	(s)	CV	(s)	CV
UW	0.6	14	**14**	0.00	237	0.06	329	0.16
Mondial	0.9	18	1836	0.07	**369**	0.06	717	0.05
Hepatitis	11.3	19	5434	0.01	6648	0.02	**3197**	0.04
Mutagenesis	24.3	11	**11**	0.00	1342	0.04	1040	0.02
MovieLens(0.1M)	83.4	7	**8**	0.07	3019	0.04	3292	0.01
MovieLens(1M)	1010.1	7/6	**8**	0.09	32230	0.04	25528	0.04
IMDb	15538.4	17/13	**9346**	0.22	78129	0.04	29704	0.03

5.7 Results

I report learning times and accuracy metrics. In addition to these quantitative assessments, I inspect the learned models to compare the model structures. Finally I make suggestions for combining the strenghts of boosting with the strengths of Bayesian network learning.

5.7.1 Learning Times

Table 5.2 shows learning times for the methods. The Bayesian network learning simultaneously learns a joint model for all parametrized random variables (PRVs). Recall that Boolean PRVs are predicates. For the boosting method, I added together the learning times for each target PRV. On MovieLens(1M), the boosting methods take over 2 days to learn a classifier for the relationship B_U2Base, so I do not include learning time for this predicate for any boosting method. On the largest database, IMDb, the boosting methods cannot learn a local distribution model for the three relationship predicates with my system resources, so I only report learning time for descriptive attributes by the boosting methods. Likewise, the accuracy results in Tables 5.3 and 5.4 include only measurements for descriptive attributes on the datasets IMDb and MovieLens(1M).

Consistent with other previous experiments on Bayesian network learning with relational data [42, 76], Table 5.2 shows that RDN_Bayes scales very well with the number of data tuples: even the MovieLens dataset with 1 M records can be analyzed in seconds. RDN_Bayes is less scalable with the number of PRVs, since it learns a joint model over all PRVs simultaneously, although the time remains feasible (1–3 hours for 17–19 predicates; see also [76]). By contrast, the boosting methods scale well with the number of predicates, which is consistent with findings from propositional learning [33]. Gradient boosting scales much worse with the number of data tuples.

Table 5.3: Conditional Log-Likelihood: Mean (top), Std. Dev. (bottom)

Method	UW	Mond.	Hepa.	Muta.	MovieLens (0.1M)	MovieLens (1M)	IMDb
RDN_Boost	-0.30	-0.48	-0.48	-0.36	-0.50	**-0.22**	**-0.49**
MLN_Boost	-0.14	-0.40	-0.49	-0.23	-0.50	-0.23	**-0.49**
RDN_Bayes	**-0.01**	**-0.25**	**-0.39**	**-0.22**	**-0.30**	-0.28	-0.51
RDN_Boost	0.02	0.03	0.01	0.02	0.01	0.00	0.00
MLN_Boost	0.01	0.05	0.01	0.02	0.01	0.00	0.00
RDN_Bayes	0.00	0.06	0.10	0.07	0.00	0.00	0.00

Table 5.4: Area Under Precision-Recall Curve: Mean (top), Std. Dev. (bottom).

Method	UW	Mond.	Hepa.	Muta.	MovieLens (0.1M)	MovieLens (1M)	IMDb
RDN_Boost	0.42	0.27	0.55	0.71	0.50	0.88	0.63
MLN_Boost	0.68	0.44	0.55	**0.86**	0.50	0.88	0.63
RDN_Bayes	**0.89**	**0.79**	0.55	0.50	**0.65**	**1.00**	**0.85**
RDN_Boost	0.00	0.00	0.01	0.02	0.01	0.00	0.01
MLN_Boost	0.01	0.04	0.01	0.04	0.01	0.00	0.01
RDN_Bayes	0.00	0.07	0.11	0.10	0.02	0.00	0.00

5.7.2 Accuracy

Whereas learning times were evaluated on all PRVs, unless otherwise noted, I evaluate accuracy on all the binary predicates only (e.g., *gender*, *Borders*) because the boosting methods are based on binary classification. By the conditional likelihood metric (Table 5.3), the Bayesian network method performs best on four datasets, comparably to MLN_Boost on Mutagenesis, and slightly worse than both boosting methods on the two largest datasets. By the precision-recall metric (Table 5.4), the Bayesian network method performs substantially better on four datasets, identically on Hepatitis, and substantially worse on Mutagenesis.

Combining these results, for most of the datasets the Bayesian network method has comparable accuracy and much faster learning. This is satisfactory because boosting is a powerful method that achieves accurate predictions by producing a tailored local model for each target predicate. By contrast, Bayesian network learning simultaneously constructs a joint model for all predicates, and uses simple maximum likelihood estimation for parameter values. I conclude that *Bayesian network learning scales much better to large datasets, and provides competitive accuracy in predictions.*

5.7.3 Comparison of Model Structures

Boosting is known to lead to very accurate classification models in general [7]. For propositional data, a Bayesian network classifier with maximum likelihood estimation for parameter values is a reasonable baseline method [27], but I would expect less accuracy than from a boosted ensemble of regression trees. Therefore the predictive performance of the RDN models is not due to the log-linear equation (1), but due to the **more powerful features** that Bayesian network learning finds in relational datasets. These features involve longer chains of relationships than I observe in the boosting models.[3] The ability to find complex patterns involving longer relationship chains comes from the lattice search strategy, which in turn depends on the scalability of model evaluation in order to explore a complex space of relationship chains. Table 5.5 reports results that quantitatively confirm this analysis.

Table 5.5: Difference in Markov blankets between RDN_Bayes and RDN_Boost. Δx = (x for RDN_Bayes - x for RDN_Boost). RDN_Bayes predicts a target more successfully because it uses more predicates and those predicates contain more first-order variables.

Database	Target	Δ Predicates	Δ Vars.	Δ CLL	Δ AUC-PR
Mondial	religion	11	1	0.58	0.30
IMDb	gender	4	2	0.30	0.68
UW-CSE	student	4	1	0.50	0.55
Hepatitis	sex	4	2	0.20	0.25
Mutagenesis	ind1	5	1	0.56	0.22
MovieLens	gender	1	1	0.26	0.26

For each database, I selected the target PRV where RDN-Bayes shows the greatest predictive advantage over RDN-Boost (shown as Δ CLL and Δ AUC-PR). I then compute how many more PRVs the RDN-Bayes model uses to predict the target predicate than the RDN-Boost model, shown as Δ Predicates. This number can be as high as 11 more PRVs (for Mondial). I also compare how many more population variables are contained in the Markov blanket of the RDN-Bayes model, shown as Δ Variables. In terms of database tables, the number of population variables measures how many related tables are used for prediction in addition to the target table. This number can be as high as 2 (for IMDb and Hepatitis). To illustrate Figure 5.3 shows the parents (Markov blanket) of target node $gender(\mathbb{U})$ from IMDb in the RDN-Boost and RDN-Bayes models. The RDN-Bayes model introduces 4 more parents and 2 more variables, *Movie* and *Actor*. These two variables correspond to a relationship chain of length 2. Thus BN learning discovers that the gender of a user can be predicted by the gender of actors that appear in movies that the user has rated.

[3]Kok and Domingos emphasize the importance of learning clauses with long relationship chains [47].

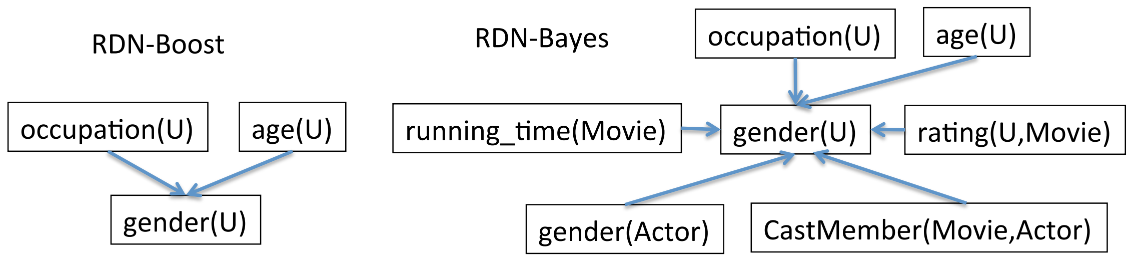


Figure 5.3: The parents of target *gender*($\mathbb{U}$) in the models discovered by RDN_Boost (left) and RDN_Bayes (right). The RDN-Bayes model discovers that the gender of a user can be predicted by the gender of actors that appear in movies that the user has rated.

5.8 Conclusion

Relational dependency networks offer important advantages for modelling relational data. They can be learned quickly by first learning a Bayesian network, then performing a closed-form transformation of the Bayesian network to a dependency network. The key question is how to transform BN parameters to DN parameters. I introduced a relational generalization of the standard propositional BN log-linear equation for the probability of a target node conditional on an assignment of values to its Markov blanket. The new log-linear equation uses a sum of expected values of BN log-conditional probabilities, with respect to a random instantiation of first-order variables. This is equivalent to using feature instantiation proportions as feature functions. The main theorem provided a necessary and sufficient condition for when the local log-linear equations for different nodes are mutually consistent. On six benchmark datasets, learning RDNs via BNs scaled much better to large datasets than state-of-the-art functional gradient boosting methods, and provided competitive accuracy in predictions.

Chapter 6

Conclusion and Future Work

In this chapter, I summarize the system, algorithms, models and results discussed in my thesis, and I list the future directions which I am planning to work on as follows:

FactorBase System In chapter 2, I described FactorBase, a system that leverages the existing capabilities of an SQL-based RDBMS to support statistical-relational learning with respect to representational tasks and computational tasks. While FactorBase provides good solutions for each of these system capabilities in isolation, the ease with which large complex statistical-relational objects can be integrated via SQL queries is a key feature. Because information about random variables, sufficient statistics, and models is all represented in relational database tables, a machine learning application can access and combine the information in a uniform way via SQL queries.

While my implementation has used simple SQL plus indexes, there are opportunities for optimizing RDBMS operations for the workloads required by statistical-relational structure learning. These include view materialization and the key scalability bottleneck of computing multi-relational sufficient statistics. NoSQL databases can exploit a flexible data representation for scaling to very large datasets. However, SRL requires count operations for random complex join queries, which is a challenge for less structured data representations.

Another important goal is a single end-to-end RDBMS package for both learning and inference that integrates FactorBase with inference systems such as BayesStore, Tuffy and DeepDive [73]. This end-to-end package will be much more friendly for the consumers without much relational learning background. The Spark SQL module offers tighter integration between relational processing and a highly extensible optimizer tailored for the complex queries [3]. Porting FactorBase system to Spark implementation would enhance the ability of Count Manager for processing very large-scale data sets which are beyond the capability of single RDBMS. Another advantage of a Spark implementation is that Spark leverages Hadoop, which is distributed file system framework for very large data sets. However, the current performance bottleneck for computing sufficient statistics is main memory not external memory. On the other hand, incorporating the PS paradigm within Model Manager would support the distributed parameter learning for many industrial scenarios (e.g., one need to deal with very big model containing millions to billions of parameters) [51].

Sufficient Statistics across Multiple Relationships As described in Chapter 3, the Virtual Join algorithm efficiently computes query counts which may involve any number of *positive and negative* relationships. These sufficient statistics support a scalable statistical analysis of associations among both relationships and attributes in a relational database. The Virtual Join algorithm scales well with the number of rows, but not with the number of columns and relationships in the database. This limitation stems from the fact that the contingency table size grows exponentially with the number of random variables in the table.

I applied the algorithm to construct a large table for *all* variables in the database. I emphasize that this is not the only way to apply the algorithm. The algorithm efficiently finds cross-table statistics for any set of variables, not only for the complete set of all variables in the database. An alternative for using counting for relational model searches is to apply the Virtual Join only up to a pre-specified relatively small relationship chain length which could be determined by a learning algorithm or specified by the user. Another possibility is to use post-counting [54]: Rather than pre-compute a large contingency table prior to learning, compute many small contingency tables for small subsets of variables on demand during learning. In a post-counting approach, generating a contingency table for a target set of variables is a service that can be called dynamically during the execution of a learning program.

While I have focused on statistical analysis, another potential application is in probabilistic first-order inference [65, 8]. Such inferences often require sufficient statistics with regard to counting the satisfied groundings of a first-order formula defined with respect to one or more specified individuals (e.g., the number of user $Jack's$ male friends). The Virtual Join algorithm could be applied to compute sufficient statistics for specified individuals. In addition, the traditional ETL + single table machine learning training based model can take advantage of the sufficient statistics to gain better performance.

Generative Modelling with Link Uncertainty I discussed two kinds of modelling approach: class-level generative modelling and instance-level discriminative learning in the context of relational data in chapter 4 and chapter 5, respectively. The generative modeling with Bayes net learns a model of the joint probability distribution of all input random variables. This type of model supports class-level queries like, "What fraction of the grades are awarded to highly intelligent students?" The model described in Chapter 4 captures a wider class of correlations that involve uncertainty about the link structure for complex heterogeneous network with many attributes and link types. By capturing the relevant correlations, the model achieves better data fit on standard statistical metrics.

Instance Level Discriminative Learning Dependency networks are well suited for the task of discriminative learning such as classification problem to predict the label y given all inputs x. This type of model supports instance-level queries like, "What is the probability of *Jack* being a highly intelligent student given the grades of his registered courses?" As introduced in chapter 5, the hybrid approach combines the speed of learning Bayes net with the advantages of dependency network inference for relational data. Empirical experiments showed the predictive accuracy and scalability of my BN-to-DN hybrid discriminative learning algorithm. The state-of-the-art boosting approach to constructing a dependency network by learning a collection of discriminative models is very different from learning a Bayesian network. There are various options for hybrid approaches that combine the strengths of both, for instance:

1. Fast Bayesian network learning can be used to select features. Discriminative learning methods should work faster restricted to the BN Markov blanket of a target node.

2. The Bayesian network can provide an initial dependency network structure. Gradient boosting can then be used to fine-tune local distribution models.

Integrated Statistical Analysis for Complex Heterogeneous Data Many real world datasets are relational and most real world applications are characterized by the presence of uncertainty and complex relational structures. In the thesis, I focus on statistical modelling of the interactions between different descriptive attributes and the link itself for complex heterogeneous and richly interconnected data. The SQL-based FactorBase system provides integrated statistical analysis out-of-the-box for challenging applications on two folds: the class-level generative modelling with Bayes net and the instance-level discriminative learning with relational dependency networks. All statistical objects are stored as first-class citizens as well as raw data; one can access and combine the information in a uniform way via SQL queries. The Virtual Join algorithm solves the scalability bottleneck for using cross-table sufficient statistics in relational model searching. The model captures a wider class of correlations that involve uncertainty about the link structure.

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] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 123–132, New York, NY, USA, 2012. ACM.

[3] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Hua, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational data processing in Spark. In *SIGMOD Conference, To Appear*, 2015.

[4] Fahiem Bacchus. *Representing and Reasoning with Probabilistic Knowledge: A Logical Approach to Probabilities.* MIT Press, Cambridge, MA, USA, 1990.

[5] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling Up Machine Learning: Parallel and Distributed Approaches.* Cambridge University Press, New York, NY, USA, 2011.

[6] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *ICML 2014*, pages 226–234, 2014.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[8] Hari Charan Cheekati, Swaroop Goli, and Deepak Venugopal. MSpark: A scalable lifted inference pipeline for MLNs. In IJCAI-16 workshop on Statistical Relational Artificial Intelligence, 2016.

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

[10] Apache Spark Project Contributors. Apache Spark. http://spark.apache.org/.

[11] Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth Gibson, and Eric P Xing. High-performance distributed ml at scale through parameter server consistency models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[12] Luc de Raedt. *Logical and Relational Learning.* Cognitive Technologies. Springer, 2008.

[13] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1232–1240, 2012.

[14] Amol Deshpande and Samuel Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, pages 73–84. ACM, 2006.

[15] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence.* Morgan and Claypool Publishers, 2009.

[16] Pedro Domingos and Matthew Richardson. Markov logic: A unifying framework for statistical relational learning. In *Introduction to Statistical Relational Learning* [23].

[17] Saso Dzeroski and Nada Lavrac. *Relational Data Mining.* Springer, Berlin, 2001.

[18] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD Conference*, pages 325–336, 2012.

[19] Jörg Flum and Martin Grohe. *Parameterized complexity theory*, volume 3. Springer, 2006.

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

[21] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Benjamin Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning* [23], chapter 5, pages 129–173.

[22] Lise Getoor and Ben Taskar. Introduction. In Getoor and Taskar [23], pages 1–8.

[23] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.

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

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

[26] Goetz Graefe, Usama M. Fayyad, and Surajit Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. In *KDD*, pages 204–208, 1998.

[27] Daniel Grossman and Pedro Domingos. Learning Bayesian network classifiers by maximizing conditional likelihood. In *ICML*, page 46, New York, NY, USA, 2004. ACM.

[28] 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.

[29] Mark A. Hall. *Correlation-based Feature Selection for Machine Learning.* PhD thesis, Department of Computer Science, The University of Waikato, 1999.

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

[31] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[32] D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In Getoor and Taskar [23].

[33] 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.

[34] Joseph M. Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. The MADlib analytics library: Or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, August 2012.

[35] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. More effective distributed ML via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1223–1231, 2013.

[36] Reimar Hofmann and Volker Tresp. Nonlinear markov networks for continuous variables. pages 521–527. MORGAN KAUFMANN PUBLISHERS, 1998.

[37] Geoff Hulten, David Maxwell Chickering, and David Heckerman. Learning bayesian networks from dependency networks: A preliminary study. In *in Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, Key West, FL*, 2003.

[38] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD Conference*, pages 687–700, 2008.

[39] Seyed Mehran Kazemi, David Buchman, Kristian Kersting, Sriraam Natarajan, and David Poole. Relational logistic regression. In *Principles of Knowledge Representation and Reasoning KR*, 2014.

[40] William Kent. A simple guide to five normal forms in relational database theory. *Commun. ACM*, 26(2):120–125, 1983.

[41] Kristian Kersting and Luc de Raedt. Bayesian logic programming: Theory and tool. In *Introduction to Statistical Relational Learning* [23], chapter 10, pages 291–318.

[42] 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.

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

[44] Tushar Khot, Jude Shavlik, and Sriraam Natarajan. Boostr. http://pages.cs.wisc.edu/ tushar/Boostr/.

[45] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015.

[46] Arno J Knobbe. *Multi-relational data mining*, volume 145. Ios Press, 2006.

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

[48] Tim Kraska, Ameet Talwalkar, John C. Duchi, Rean Griffith, Michael J. Franklin, and Michael I. Jordan. MLbase: A distributed machine-learning system. In *CIDR*, 2013.

[49] 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.

[50] Nada Lavravc, Matic Perovvsek, and Anvze Vavpetivc. Propositionalization online. In *ECML*, pages 456–459. Springer, 2014.

[51] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, pages 583–598, 2014.

[52] D. Lowd. Closed-form learning of Markov networks from dependency networks. In *UAI*, 2012.

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

[54] Qiang Lv, Xiaoyan Xia, and Peide Qian. A fast calculation of metric scores for learning Bayesian network. *Int. J. of Automation and Computing*, 9:37–44, 2012.

[55] Brian Milch, Bhaskara Marthi, Stuart J. Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: probabilistic models with unknown objects. In *IJCAI-05*, pages 1352–1359, 2005.

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

[57] Andrew W. Moore and Mary S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *J. Artif. Intell. Res. (JAIR)*, 8:67–91, 1998.

[58] 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.

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

[60] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Felix: Scaling Inference for Markov Logic with an Operator-based Approach. *ArXiv e-prints*, August 2011.

[61] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

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

[63] Verónika Peralta. Extraction and integration of MovieLens and IMDb data. Technical report, Technical Report, Laboratoire PRiSM, 2007.

[64] Jose Picado, Parisa Ataei, Arash Termehchy, and Alan Fern. Schema independent and scalable relational learning by castor. *PVLDB*, 9(13):1589–1592, 2016.

[65] David Poole. First-order probabilistic inference. In *IJCAI*, 2003.

[66] Zhensong Qian and Oliver Schulte. Learning bayes nets for relational data with link uncertainty. In M. Croitoru et al., editor, *GKR*, volume 8323 of *Lecture Notes in Artificial Intelligence*, pages 123–137. Springer, 2014.

[67] Zhensong Qian and Oliver Schulte. The BayesBase system, 2015. `http://www.cs.sfu.ca/~oschulte/BayesBase/BayesBase.html`.

[68] Zhensong Qian and Oliver Schulte. Factorbase: Multi-relational model learning with sql all the way. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015.

[69] Zhensong Qian and Oliver Schulte. FACTORBASE: SQL for multi-relational model learning. LearningSys, NIPS Workshop, 2015.

[70] Zhensong Qian, Oliver Schulte, and Yan Sun. Computing multi-relational sufficient statistics for large databases. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1249–1258. ACM, 2014.

[71] Zhensong Qian, Oliver Schulte, and Yan Sun. Computing multi-relational sufficient statistics for large databases. *CoRR*, abs/1408.5389, 2014.

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

[73] Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. Deepdive: Declarative knowledge base construction. *SIGMOD Record*, 45(1):60–67, 2016.

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

[75] Oliver Schulte. Challenge paper: Marginal probabilities for instances and classes. ICML-SRL Workshop on Statistical Relational Learning., June 2012.

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

[77] Oliver Schulte, Hassan Khosravi, Arthur Kirkpatrick, Tianxiang Gao, and Yuke Zhu. Modelling relational statistics with bayes nets. In *Inductive Logic Programming (ILP)*, 2012.

[78] Oliver Schulte, Hassan Khosravi, Arthur Kirkpatrick, Tianxiang Gao, and Yuke Zhu. Modelling relational statistics with bayes nets. *Machine Learning*, 94:105–125, 2014.

[79] Oliver Schulte and Zhensong Qian. Factorbase: SQL for learning a multi-relational graphical model. *arXiv preprint*, August 2015.

[80] Oliver Schulte, Zhensong Qian, Arthur E. Kirkpatrick, Xiaoqian Yin, and Yan Sun. Fast learning of relational dependency networks. *CoRR*, abs/1410.7835, 2014.

[81] Oliver Schulte, Zhensong Qian, Arthur E. Kirkpatrick, Xiaoqian Yin, and Yan Sun. Fast learning of relational dependency networks. *Machine Learning*, pages 1–30, 2016.

[82] Sameer Singh and Thore Graepel. Automated probabilistic modeling for relational data. In *CIKM*, pages 1497–1500. ACM, 2013.

[83] Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1-2):703–710, 2010.

[84] Yizhou Sun and Jiawei Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*, volume 3. Morgan & Claypool Publishers, 2012.

[85] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In *Introduction to Statistical Relational Learning* [23], chapter 4, pages 93–127.

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

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

[88] J. D. Ullman. *Principles of Database Systems.* W. H. Freeman & Co., 2 edition, 1982.

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

[90] Trevor Walker, Ciaran O'Reilly, Gautam Kunapuli, Sriraam Natarajan, Richard Maclin, David Page, and Jude W. Shavlik. Automating the ILP setup task: Converting user advice about specific examples into general background knowledge. In *ILP*, pages 253–268, 2010.

[91] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M Hellerstein. BayesStore: managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, volume 1, pages 340–351, 2008.

[92] Michael L. Wick, Andrew McCallum, and Gerome Miklau. Scalable probabilistic databases with factor graphs and MCMC. In *PVLDB*, volume 3, pages 794–804, 2010.

[93] SK Michael Wong, Cory J Butz, and Yang Xiang. A method for implementing a probabilistic model as a relational database. In *UAI*, pages 556–564, 1995.

[94] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *ICDE*, 2004.

[95] Nan Zhu, Lei Rao, and Xue Liu. PD2F: Running a parameter server within a distributed dataflow framework, 2015. LearningSys, NIPS Workshop.

Appendix A

Proof of Consistency Characterization

This appendix presents a proof of Theorem 1. The theorem says that a dependency network derived from a template Bayesian network is consistent if and only if the Bayesian network is edge-consistent. I begin by showing that Bayesian network edge-consistency is sufficient for dependency network consistency. This is the easy direction. That edge-consistency is also necessary requires several intermediate results.

A.1 Edge-Consistency is Sufficient for Consistency

Edge consistency entails that each grounding of a node determines a unique grounding of both its parents and its children in the Bayesian network. Thus the ground dependency network is composed of disjoint dependency networks, one for each grounding. Each of the ground disjoint dependency networks is consistent, so a joint distribution over all can be defined as the product of the joint probabilities of each ground dependency network. The formal statement and proof is as follows.

Proposition 3. *If a template Bayesian network is edge-consistent, then the derived dependency network is consistent.*

Proof. Heckermann *et al.* [33] showed that a dependency network is consistent if and only if there is a Markov network with the same graphical structure that agrees with the local conditional distributions. I argue that given edge-consistency, there is such a Markov network for the derived dependency network. This Markov network is obtained by moralizing and then grounding the Bayesian network [15]. Given edge-consistency, for each ground target node, each family of the ground target node has a unique grounding. Thus the relevant family counts are all either 1 or 0 (0 if the family configuration is irrelevant). The Markov network is now defined as follows: Each grounding of a family in the template Bayesian network is a clique. For an assignment of values $U^* = u, \mathrm{Pa}(U)^* = \vec{u}_{pa}$ to a ground family, the clique potential is 1 if the assignment is irrelevant, and $\theta(U = u|\mathrm{Pa}(U) = \vec{u}_{pa})$ otherwise. It is easy to see that the conditional distributions induced by this Markov network agree with those defined by Equation 1, given edge-consistency. □

A.2 Edge-Consistency is Necessary for Consistency

This direction requires a mild condition on the structure of the Bayesian network: it must not contain a redundant edge [62]. An edge $T_1 \to T_2$ is redundant if for every value of the parents of T_2 excluding T_1, every value of T_1 is conditionally independent of every value of T_2. Less formally, given the other parents, the node T_1 adds no probabilistic information about the child node T_2. Throughout the remainder of the proof, I assume that the template Bayesian network contains no redundant edges. The proof is based on establishing the following theorem.

Theorem 2. *Assume that a template BN contains at least one edge e_1 such that the parent and child do not contain the same set of population variables. Then there exists an edge e_2 (which may be the same as or distinct from e_1) from parent T_1 to child T_2, ground nodes T_1^* and T_2^*, and a query conjunction Λ^* such that: the ground nodes T_1^* and T_2^* have mutually inconsistent conditional distributions $\theta(T_1^*|\Lambda^*)$ and $\theta(T_2^*|\Lambda^*)$ as defined by Equation 1.*

The query conjunction Λ^* here denotes a complete specification of all values for all ground nodes except for T_1^* and T_2^*. Theorem 2 entails the necessity direction of Theorem 1 by the following argument. Suppose that there is a joint distribution p that agrees with the conditional distributions of the derived dependency network. Then for every query conjunction Λ^*, and for every assignment of values t_1 resp. t_2 to the ground nodes, I have that $p(T_1^* = t_1|T_2^* = t_2, \Lambda^*)$ and $p(T_2^* = t_2|T_1^* = t_1, \Lambda^*)$ agree with the log-linear equation 1. Therefore, the conditional distributions $p(T_1^*|T_2^*, \Lambda^*)$ and $p(T_2^*|T_1^*, \Lambda^*)$ must be mutually consistent. Theorem 2 asserts that for every (non-redundant) edge-inconsistent template BN, I can find a query conjunction and two ground nodes such that the conditional distributions of the ground nodes given the query conjunction are not mutually consistent. Therefore there is no joint distribution that is consistent with all the conditional distributions defined by the log-linear equations, which establishes the necessity direction of the main theorem 1.

A.2.1 Properties of the template BN and the input query Λ^*

I begin by establishing some properties of the template BN and the query conjunction that are needed in the remainder of the proof.

The inconsistency of the BN networks arises when a parent and a child ground node have different relevant family counts. The next lemma shows that this is possible exactly when the template BN is properly relational, meaning it relates parents and children from different populations.

Lemma 1. *The following conditions are equivalent for a template edge $T_1 \to T_2$.*

1. *The parent and child do not contain the same population variables.*
2. *It is possible to find a grounding γ for both parent and child, and an assignment Λ^* to all other nodes, such that the relevant family count for the T_2 family differs for $T_1^* = \gamma T_1$ and $T_2^* = \gamma T_2$.*

Proof. If the parent and child contain the same population variables, then there is a 1-1 correspondence between groundings of the child and groundings of the parents. Hence the count of relevant family groundings is the same for each, no matter how parents and child are instantiated. If the parent and child do not contain the same population variables, suppose without loss of generality that the child contains a population variable $\mathbb{A}$ not contained in the parent. Choose a common grounding γ for the parents and child node. For the ground child node, γT_2, let γ be the only family grounding that is relevant, so the relevant count is 1. For the ground parent node, there is at least one other grounding of the child node T_2' different from γT_2 since T_2 contains another population variables. Thus it is possible to add another relevant family grounding for γT_1, which means that the relevant count is at least 2. □

The proof proceeds most simply if I focus on template edges that relate different populations and no common children.

Definition 2. *An template edge $T_1 \to T_2$ is* ***suitable*** *if*

1. *The parent and child do not contain the same population variables.*
2. *The parent and child have no common edge.*

The next lemma shows that focusing on suitable edges incurs no loss of generality.

Lemma 2. *Suppose that a template BN contains an edge such that the parent and child do not contain the same population variables. Then the template BN contains a suitable edge.*

Proof. Suppose that there is an edge satisfying the population variable condition. Suppose that the parent and child share a common child. Since the edge satisfies the condition, the set of population variables in the common child differs from at least one of T_1, T_2. Therefore there is another edge from one of $T_1 \to T_2$ as parent to a new child that satisfies the population variable condition. If this edge is not suitable, there must be another shared child. Repeating this argument, I eventually arrive at an edge satisfying the population variable condition where the child node is a sink node without children. This edge is suitable. □

Consider a suitable template edge $T_1 \to T_2$ that produces a bidirected ground edge $T_1^* \leftrightarrow T_2^*$. For simplicity I assume that T_1 and T_2 are binary variables with domain $\{\mathrm{T}, \mathrm{F}\}$. (This incurs no loss of generality as I can choose a database Λ^* in which only two values occur.) Let $\mathrm{Pa}(T_2)$ be the parents of T_2 other than T_1. Since the template edge is not redundant [62], there is a parent value setting $\mathrm{Pa}(T_2) = \mathbf{pa}$ such that T_1 and T_2 are conditionally dependent given $\mathrm{Pa}(T_2) = \mathbf{pa}$. This implies that the conditional distribution of T_1 is different for each of the two possible values of T_2:

$$\frac{\theta(T_2 = \mathrm{F}|T_1 = \mathrm{F}, \mathbf{pa})}{\theta(T_2 = \mathrm{T}|T_1 = \mathrm{F}, \mathbf{pa})} \neq \frac{\theta(T_2 = \mathrm{F}|T_1 = \mathrm{T}, \mathbf{pa})}{\theta(T_2 = \mathrm{T}|T_1 = \mathrm{T}, \mathbf{pa})}. \tag{A.1}$$

Let Λ^* denote an assignment of values to all ground nodes other than the target nodes T_1^* and T_2^*. I assume that the input query Λ^* assigns different relevant family counts N_1 to T_1^* and N_2 to T_2^*. This is possible according to Lemma 1.

Lowd's Equation and Relevant Family Counts

The log-linear equation 1, specifies the conditional distribution of each target node given Λ^* and a value for the other target node. I keep the assignment Λ^* fixed throughout, so for more compact notation, I abbreviate the conditional distributions as

$$p({T_1}^* = t_1 | {T_2}^* = t_2) \equiv P({T_1}^* = t_1 | {T_2}^* = t_2, \Lambda^*)$$

and similarly for $P({T_1}^* = t_1 | {T_2}^* = t_2, \Lambda^*)$.

On the assumption that the dependency network is consistent, there is a joint distribution over the target nodes conditional on the assignment that agrees with the conditional distribution:

$$\frac{p({T_1}^* = t_1, {T_2}^* = t_2)}{p({T_2}^* = t_2)} = p({T_1}^* = t_1 | T_2)^*$$

and also with the conditional $p({T_2}^* = t_2 | {T_1}^* = t_1)$.

Lowd [52] pointed out that this joint distribution satisfies the equations

$$\frac{p(\mathrm{F},\mathrm{F})}{p(\mathrm{T},\mathrm{F})} \cdot \frac{p(\mathrm{T},\mathrm{F})}{p(\mathrm{T},\mathrm{T})} = \frac{p(\mathrm{F},\mathrm{F})}{p(\mathrm{T},\mathrm{T})} = \frac{p(\mathrm{F},\mathrm{F})}{p(\mathrm{F},\mathrm{T})} \cdot \frac{p(\mathrm{F},\mathrm{T})}{p(\mathrm{T},\mathrm{T})} \tag{A.2}$$

Since the ratio of joint probabilities is the same as the ratio of conditional probabilities for the same conditioning event, consistency entails the following constraint on conditional probabilities via Equation (A.2):

$$\frac{p({T_2}^* = \mathrm{F} | {T_1}^* = \mathrm{F})}{p({T_2}^* = \mathrm{T} | {T_1}^* = \mathrm{F})} \cdot \frac{p({T_1}^* = \mathrm{F} | T_2^* = \mathrm{T})}{p({T_1}^* = \mathrm{T} | {T_2}^* = \mathrm{T})} = \frac{p({T_1}^* = \mathrm{F} | {T_2}^* = \mathrm{F})}{p({T_1}^* = \mathrm{T} | {T_2}^* = \mathrm{F})} \cdot \frac{p({T_2}^* = \mathrm{F} | {T_1}^* = \mathrm{T})}{p({T_2}^* = \mathrm{T} | {T_1}^* = \mathrm{T})} \tag{A.3}$$

I refer to Equation A.3 as *Lowd's equation.* The idea of the proof is to show that Lowd's equations are satisfied only if the relevant family counts for the target nodes are the same. According to the log-linear equation, each conditional probability is proportional to a product of BN parameters. The first step is to show that in Lowd's equation, all BN parameter terms cancel out except for those that are derived from the family that comprises T_1^* and their T_2^* and their common grounding.

Lemma 3. *The conditional probabilities for the target nodes can be written as follows:*

$$P({T_2}^* = t_2 | T_1^* = t_1, \Lambda^*) \propto \theta(T_2 = t_2 | T_1 = t_1, \mathbf{pa})^{(N/N_2 + M_{T_2 = t_2}/N_2)} \cdot \pi_{T_2 = t_2} \tag{A.4}$$

where $M_{T_2=t_2}$ and $\pi_{T_2=t_2}$ depend only on t_2 and not on t_1 and

$$P({T_1}^* = t_1 | T_2^* = t_2, \Lambda^*) \propto \theta(T_2 = t_2 | T_1 = t_1, \mathbf{pa})^{(N/N_1 + M_{T_1 = t_1}/N_1)} \cdot \pi_{T_1 = t_1} \tag{A.5}$$

where $M_{T_1=t_1}$ and $\pi_{T_1=t_1}$ depend only on t_1 and not on t_2.

Proof Outline. This is based on analysing the different types of families that appear in the log-linear equation and their groundings. I omit this straightforward analysis to simplify the proof; the details are available from [80].

Lemma 4. *Suppose that conditions* (A.4) *and* (A.5) *of Lemma 3 hold. Then Lowd's Equation* (A.3) *holds if and only if* $N_1 = N_2$.

Proof. Observe that in Equation (A.3), each term on the left has a corresponding term with the same value for the target node assignment and the opposing conditioning assignment. For instance, the term $p({T_2}^* = \mathrm{F}|{T_1}^* = \mathrm{F})$ on the left is matched with the term $p({T_2}^* = \mathrm{F}|{T_1}^* = \mathrm{T})$ on the right. This means that the products in the log-linear expression are the same on both sides of the equation except for those factors that depend on *both* t_1 and t_2. Continuing the example, the factors

$$\theta(T_2 = \mathrm{F}|T_1 = \mathrm{F}, \mathbf{pa})^{(M_\mathrm{F}/N_2)} \cdot \pi_{T_2=t_2}$$

on the left equal the factors

$$\theta(T_2 = \mathrm{F}|T_1 = \mathrm{T}, \mathbf{pa})^{(M_{T_1=t_1}/N_2)} \cdot \pi_{T_2=t_2}$$

on the right side of the equation. They therefore cancel out, leaving only the term

$$\theta(T_2 = \mathrm{F}|T_1 = \mathrm{F}, \mathbf{pa})^{N/N_2}$$

on the left and the term

$$\theta(T_2 = \mathrm{F}|T_1 = \mathrm{F}, \mathbf{pa})^{N/N_2}$$

on the right. Lowd's equation can therefore be reduced to an equivalent constraint with only such BN parameter terms. For further compactness I abbreviate such terms as follows

$$\theta(t_2|t_1) \equiv \theta(T_2 = t_2|T_1 = t_1, \mathbf{pa}).$$

With this abbreviation, the conditions of Lemma 3 entail that Lowd's equation A.3 reduces to the equivalent expressions.

$$\frac{\theta(\mathrm{F}|\mathrm{F})^{N/N_2}}{\theta(\mathrm{T}|\mathrm{F})^{N/N_2}} \cdot \frac{\theta(\mathrm{T}|\mathrm{F})^{N/N_1}}{\theta(\mathrm{T}|\mathrm{T})^{N/N_1}} = \frac{\theta(\mathrm{F}|\mathrm{F})^{N/N_1}}{\theta(\mathrm{F}|\mathrm{T})^{N/N_1}} \cdot \frac{\theta(\mathrm{F}|\mathrm{T})^{N/N_2}}{\theta(\mathrm{T}|\mathrm{T})^{N/N_2}} \tag{A.6}$$

$$(\frac{\theta(\mathrm{F}|\mathrm{F})}{\theta(\mathrm{T}|\mathrm{F})})^{(N/N_2 - N/N_1)} = (\frac{\theta(\mathrm{F}|\mathrm{T})}{\theta(\mathrm{T}|\mathrm{T})})^{(N/N_2 - N/N_1)} \tag{A.7}$$

By the nonredundancy assumption (A.1) on the BN parameters, I have

$$\frac{\theta(\mathrm{F}|\mathrm{F})}{\theta(\mathrm{T}|\mathrm{F})} \neq \frac{\theta(\mathrm{F}|\mathrm{T})}{\theta(\mathrm{T}|\mathrm{T})}$$

so Equation A.7 implies that

$$N_1 = N_2,$$

which establishes the lemma. □

Theorem 2 now follows as follows: Lemma 1 entails that if the dependency network is consistent, the log-linear equations satisfy Lowd's equation with the bidirected ground edge $T_1^* \leftrightarrow T_2^*$ and the query conjunction Λ^* that satisfies the BN non-redundancy condition. Lemmas A.3 and 2 show that if the template BN is relational, it must contain a suitable edge

$T_1 \rightarrow T_2$. Lemma 4 together with Lowd's equation entails that the relevant counts for T_1^* and T_2^* must then be the same. But the query conjunction Λ^* was chosen so that the relevant counts are different. This contradiction shows that Lowd's equation is unsatisfiable, and therefore no joint distribution exists that is consistent with the BN conditional distributions specified by the log-linear Equation 1. Since Theorem 2 entails Theorem 1, the proof is complete.

Appendix B

B.1 SQL Queries for Achema Analyzer

```
DROP SCHEMA IF EXISTS @database@_AchemaAnalyzer;
CREATE SCHEMA  @database@_AchemaAnalyzer;

CREATE SCHEMA  if not exists @database@_BN;
CREATE SCHEMA  if not exists @database@_CT;

USE @database@_AchemaAnalyzer;
SET storage_engine=INNODB;

CREATE TABLE Schema_Key_Info AS SELECT TABLE_NAME, COLUMN_NAME,
REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME, CONSTRAINT_NAME FROM
INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE (KEY_COLUMN_USAGE.TABLE_SCHEMA =
'@database@') ORDER BY TABLE_NAME;

CREATE TABLE Schema_Position_Info AS SELECT COLUMNS.TABLE_NAME,
COLUMNS.COLUMN_NAME,
COLUMNS.ORDINAL_POSITION FROM
INFORMATION_SCHEMA.COLUMNS,
INFORMATION_SCHEMA.TABLES
WHERE
(COLUMNS.TABLE_SCHEMA = '@database@'
    AND TABLES.TABLE_SCHEMA = '@database@'
    AND TABLES.TABLE_NAME = COLUMNS.TABLE_NAME
    AND TABLES.TABLE_TYPE = 'BASE TABLE')
ORDER BY TABLE_NAME;

CREATE TABLE NoPKeys AS SELECT TABLE_NAME FROM
Schema_Key_Info
WHERE
TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
```

```
        Schema_Key_Info
    WHERE
        CONSTRAINT_NAME LIKE 'PRIMARY');

CREATE table NumEntityColumns AS
SELECT
    TABLE_NAME, COUNT(DISTINCT COLUMN_NAME) num
FROM
    Schema_Key_Info
WHERE
    CONSTRAINT_NAME LIKE 'PRIMARY'
        OR REFERENCED_COLUMN_NAME IS NOT NULL
GROUP BY TABLE_NAME;

CREATE TABLE TernaryRelations as SELECT TABLE_NAME FROM
NumEntityColumns
WHERE
num > 2;

CREATE TABLE KeyColumns AS SELECT * FROM
(Schema_Key_Info
NATURAL JOIN Schema_Position_Info)
WHERE
TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        NoPKeys)
    AND TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        TernaryRelations);

CREATE TABLE AttributeColumns AS SELECT TABLE_NAME, COLUMN_NAME FROM
Schema_Position_Info
WHERE
(TABLE_NAME , COLUMN_NAME) NOT IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM
        KeyColumns)
    and TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        NoPKeys)
    and TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        TernaryRelations);
```

```
ALTER TABLE AttributeColumns ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME);

CREATE TABLE InputColumns AS SELECT * FROM
KeyColumns
WHERE
CONSTRAINT_NAME = 'PRIMARY'
ORDER BY TABLE_NAME;

CREATE TABLE ForeignKeyColumns AS SELECT * FROM
KeyColumns
WHERE
REFERENCED_COLUMN_NAME IS NOT NULL
ORDER BY TABLE_NAME;

ALTER TABLE ForeignKeyColumns
 ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME,REFERENCED_TABLE_NAME);

CREATE TABLE EntityTables AS SELECT distinct TABLE_NAME, COLUMN_NAME FROM
KeyColumns T
WHERE
1 = (SELECT
        COUNT(COLUMN_NAME)
    FROM
        KeyColumns T2
    WHERE
        T.TABLE_NAME = T2.TABLE_NAME
            AND CONSTRAINT_NAME = 'PRIMARY');

ALTER TABLE EntityTables ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME);

CREATE TABLE SelfRelationships AS
 SELECT DISTINCT RTables1.TABLE_NAME AS TABLE_NAME,
RTables1.REFERENCED_TABLE_NAME AS REFERENCED_TABLE_NAME,
RTables1.REFERENCED_COLUMN_NAME AS REFERENCED_COLUMN_NAME FROM
KeyColumns AS RTables1,
KeyColumns AS RTables2
WHERE
(RTables1.TABLE_NAME = RTables2.TABLE_NAME)
    AND (RTables1.REFERENCED_TABLE_NAME = RTables2.REFERENCED_TABLE_NAME)
    AND (RTables1.REFERENCED_COLUMN_NAME = RTables2.REFERENCED_COLUMN_NAME)
    AND (RTables1.ORDINAL_POSITION < RTables2.ORDINAL_POSITION);

ALTER TABLE SelfRelationships ADD PRIMARY KEY (TABLE_NAME);

CREATE TABLE Many_OneRelationships AS SELECT KeyColumns1.TABLE_NAME FROM
KeyColumns AS KeyColumns1,
KeyColumns AS KeyColumns2
WHERE
(KeyColumns1.TABLE_NAME , KeyColumns1.COLUMN_NAME) IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM
        InputColumns)
    AND (KeyColumns2.TABLE_NAME , KeyColumns2.COLUMN_NAME) IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM
        ForeignKeyColumns)
    AND (KeyColumns2.TABLE_NAME , KeyColumns2.COLUMN_NAME) NOT IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM
        InputColumns);

CREATE TABLE PVariables AS SELECT CONCAT(EntityTables.TABLE_NAME, '0') AS Pvid,
EntityTables.TABLE_NAME,
0 AS index_number FROM
EntityTables
UNION
SELECT
CONCAT(EntityTables.TABLE_NAME, '1') AS Pvid,
EntityTables.TABLE_NAME,
1 AS index_number
FROM
EntityTables,
SelfRelationships
WHERE
EntityTables.TABLE_NAME = SelfRelationships.REFERENCED_TABLE_NAME
    AND EntityTables.COLUMN_NAME = SelfRelationships.REFERENCED_COLUMN_NAME ;

ALTER TABLE PVariables ADD PRIMARY KEY (Pvid);

CREATE TABLE RelationTables AS SELECT DISTINCT ForeignKeyColumns.TABLE_NAME,
ForeignKeyColumns.TABLE_NAME IN (SELECT
        TABLE_NAME
    FROM
        SelfRelationships) AS SelfRelationship,
ForeignKeyColumns.TABLE_NAME IN (SELECT
        TABLE_NAME
    FROM
        Many_OneRelationships) AS Many_OneRelationship FROM
ForeignKeyColumns;

ALTER TABLE RelationTables ADD PRIMARY KEY (TABLE_NAME);

CREATE TABLE 1Variables AS
```

```
 SELECT CONCAT('`', COLUMN_NAME, '(', Pvid, ')', '`') AS 1VarID,
COLUMN_NAME,
Pvid,
index_number = 0 AS main FROM
PVariables
    NATURAL JOIN
AttributeColumns;

ALTER TABLE 1Variables ADD PRIMARY KEY (1VarID);
ALTER TABLE 1Variables ADD UNIQUE(Pvid,COLUMN_NAME);

CREATE TABLE ForeignKeys_pvars AS SELECT ForeignKeyColumns.TABLE_NAME,
ForeignKeyColumns.REFERENCED_TABLE_NAME,
ForeignKeyColumns.COLUMN_NAME,
Pvid,
index_number,
ORDINAL_POSITION AS ARGUMENT_POSITION FROM
ForeignKeyColumns,
PVariables
WHERE
PVariables.TABLE_NAME = REFERENCED_TABLE_NAME;

ALTER TABLE ForeignKeys_pvars ADD PRIMARY KEY (TABLE_NAME,Pvid,ARGUMENT_POSITION);

CREATE table Relationship_MM_NotSelf AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars1.TABLE_NAME,
            '(',
            ForeignKeys_pvars1.Pvid,
            ',',
            ForeignKeys_pvars2.Pvid,
            ')',
            '`') AS orig_RVarID,
    ForeignKeys_pvars1.TABLE_NAME,
    ForeignKeys_pvars1.Pvid AS Pvid1,
    ForeignKeys_pvars2.Pvid AS Pvid2,
    ForeignKeys_pvars1.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars2.COLUMN_NAME AS COLUMN_NAME2,
    (ForeignKeys_pvars1.index_number = 0
        AND ForeignKeys_pvars2.index_number = 0) AS main
FROM
    ForeignKeys_pvars AS ForeignKeys_pvars1,
    ForeignKeys_pvars AS ForeignKeys_pvars2,
    RelationTables
WHERE
    ForeignKeys_pvars1.TABLE_NAME = ForeignKeys_pvars2.TABLE_NAME
```

```
        AND RelationTables.TABLE_NAME = ForeignKeys_pvars1.TABLE_NAME
        AND ForeignKeys_pvars1.ARGUMENT_POSITION
             < ForeignKeys_pvars2.ARGUMENT_POSITION
        AND RelationTables.SelfRelationship = 0
        AND RelationTables.Many_OneRelationship = 0;

CREATE table Relationship_MM_Self AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars1.TABLE_NAME,
            '(',
            ForeignKeys_pvars1.Pvid,
            ',',
            ForeignKeys_pvars2.Pvid,
            ')',
            '`') AS orig_RVarID,
    ForeignKeys_pvars1.TABLE_NAME,
    ForeignKeys_pvars1.Pvid AS Pvid1,
    ForeignKeys_pvars2.Pvid AS Pvid2,
    ForeignKeys_pvars1.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars2.COLUMN_NAME AS COLUMN_NAME2,
    (ForeignKeys_pvars1.index_number = 0
        AND ForeignKeys_pvars2.index_number = 1) AS main
FROM
    ForeignKeys_pvars AS ForeignKeys_pvars1,
    ForeignKeys_pvars AS ForeignKeys_pvars2,
    RelationTables
WHERE
    ForeignKeys_pvars1.TABLE_NAME = ForeignKeys_pvars2.TABLE_NAME
        AND RelationTables.TABLE_NAME = ForeignKeys_pvars1.TABLE_NAME
        AND ForeignKeys_pvars1.ARGUMENT_POSITION
          < ForeignKeys_pvars2.ARGUMENT_POSITION
        AND ForeignKeys_pvars1.index_number < ForeignKeys_pvars2.index_number
        AND RelationTables.SelfRelationship = 1
        AND RelationTables.Many_OneRelationship = 0;

CREATE table Relationship_MO_NotSelf AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars.REFERENCED_TABLE_NAME,
            '(',
            PVariables.Pvid,
            ')=',
            ForeignKeys_pvars.Pvid,
            '`') AS orig_RVarID,
    ForeignKeys_pvars.TABLE_NAME,
    PVariables.Pvid AS Pvid1,
    ForeignKeys_pvars.Pvid AS Pvid2,
    KeyColumns.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars.COLUMN_NAME AS COLUMN_NAME2,
    (PVariables.index_number = 0
        AND ForeignKeys_pvars.index_number = 0) AS main
FROM
    ForeignKeys_pvars,
    RelationTables,
    KeyColumns,
    PVariables
WHERE
    RelationTables.TABLE_NAME = ForeignKeys_pvars.TABLE_NAME
        AND RelationTables.TABLE_NAME = PVariables.TABLE_NAME
        AND RelationTables.TABLE_NAME = KeyColumns.TABLE_NAME
        AND RelationTables.SelfRelationship = 0
        AND RelationTables.Many_OneRelationship = 1;

CREATE table Relationship_MO_Self AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars.REFERENCED_TABLE_NAME,
            '(',
            PVariables.Pvid,
            ')=',
            ForeignKeys_pvars.Pvid,
            '`') AS orig_RVarID,
    ForeignKeys_pvars.TABLE_NAME,
    PVariables.Pvid AS Pvid1,
    ForeignKeys_pvars.Pvid AS Pvid2,
    KeyColumns.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars.COLUMN_NAME AS COLUMN_NAME2,
    (PVariables.index_number = 0
        AND ForeignKeys_pvars.index_number = 1) AS main
FROM
    ForeignKeys_pvars,
    RelationTables,
    KeyColumns,
    PVariables
WHERE
    RelationTables.TABLE_NAME = ForeignKeys_pvars.TABLE_NAME
        AND RelationTables.TABLE_NAME = PVariables.TABLE_NAME
        AND RelationTables.TABLE_NAME = KeyColumns.TABLE_NAME
        AND PVariables.index_number < ForeignKeys_pvars.index_number
        AND RelationTables.SelfRelationship = 1
        AND RelationTables.Many_OneRelationship = 1;

CREATE TABLE Relationship AS SELECT * FROM
```

```
Relationship_MM_NotSelf
UNION SELECT
*
FROM
Relationship_MM_Self
UNION SELECT
*
FROM
Relationship_MO_NotSelf
UNION SELECT
*
FROM
Relationship_MO_Self;

ALTER TABLE Relationship ADD PRIMARY KEY (orig_RVarID);
ALTER TABLE ‘Relationship‘ ADD COLUMN ‘RVarID‘
VARCHAR(10) NULL , ADD UNIQUE INDEX ‘RVarID_UNIQUE‘ (‘RVarID‘ ASC) ;


CREATE TABLE 2Variables AS SELECT CONCAT('‘',
        COLUMN_NAME,
        '(',
        Pvid1,
        ',',
        Pvid2,
        ')',
        '‘') AS 2VarID,
COLUMN_NAME,
Pvid1,
Pvid2,
TABLE_NAME,
main FROM
Relationship
    NATURAL JOIN
AttributeColumns;

ALTER TABLE 2Variables ADD PRIMARY KEY (2VarID);</p>
```

B.2 Tables Dependency in the Random Variable Database *VDB*

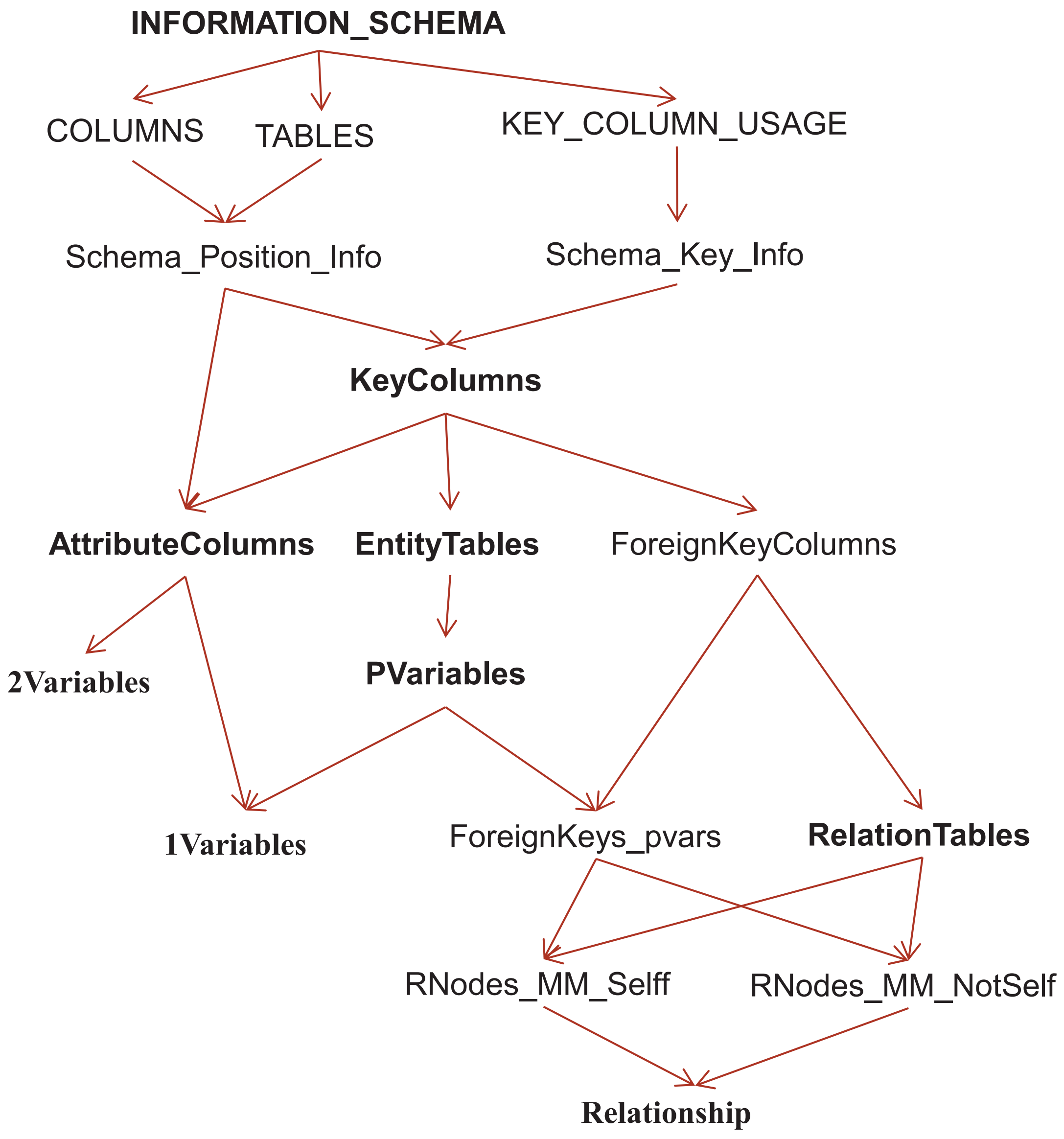


Figure B.1: Tables Dependency in the Random Variable Database *VDB*.

B.3 Schema for Random Variable Database

Table B.1: Schema for Random Variable Database

Table Name	Schema
AttributeColumns	TABLE_NAME, COLUMN_NAME
Domain	COLUMN_NAME, VALUE
Pvariables	Pvid, TABLE_NAME
1Variables	1VarID, COLUMN_NAME, Pvid
2Variables	2VarID, COLUMN_NAME, Pvid1, Pvid2, TABLE_NAME
Relationship	RVarID, TABLE_NAME, Pvid1, Pvid2, COLUMN_NAME1, COLUMN_NAME2