Upgrading Bayesian Network Scores for Multi-Relational Data

by

Sajjad Gholami

B.Sc., Amirkabir University of Technology (Tehran Polytechnic), 2014

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2016

Approval

Name:	**Sajjad Gholami**	
Degree:	**Master of Science**	
Title:	***Upgrading Bayesian Network Scores for Multi-Relational Data***	
Examining Committee:	**Chair:**	Dr. Ted Kirkpatrick Associate Professor Depeartment of Computing Science Simon Fraser University

Dr. Oliver Schulte
Senior Supervisor
Professor
Depeartment of Computing Science
Simon Fraser University

Dr. Leonid Chindelevitch
Supervisor
Assistant Professor
Depeartment of Computing Science
Simon Fraser University

Dr. Mark Schmidt
External Examiner
Assistant Professor
Department of Computer Science
University of British Columbia

Date Defended: 20 September 2016

Abstract

A multi-relational bayesian network model provides an integrated statistical analysis of the inter-dependent data resources in a database. A model selection score measures how well a model fits a dataset. We describe a new method for extending, or upgrading, a Bayesian network score designed for single-table i.i.d. data to multi-relational datasets (databases). Our method defines a *gain function* that measures the difference in data fit between two bayesian network structures. Theoretical analysis shows that it satisfies two key properties: (1) Relative consistency: If the bayesian network i.i.d. score is consistent for i.i.d. data, the upgraded version is consistent for multi-relational data. (2) Balance: The model's data likelihood and the model's complexity should be measured on the same scale. A surprising negative finding is that for log-linear relational Bayesian network likelihood scores, it is not possible to achieve these two properties with a model score that is a function of a single model only. Empirical evaluation on six benchmark relational databases shows that our consistent gain function method finds a Bayesian network structure that strikes a balance between overly sparse and overly dense graph structures.

Keywords: Model Selection; Statistical-Relational Learning; Bayesian Networks

Dedication

Dedicated to all the future relational people who know statistics :D

Acknowledgements

Acknowledge committee members, family, friends (fatemeh, zhensong, ramtin, ehsan, mahmoud, kamyar, nazanin, sima, alireza, hassan khosravi, ...)

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

Many organizations maintain their data in a multi-relational database, where multiple tables represent attributes of entities, relationships/links among these entities, and attributes of the relationships. Independent and Identically Distributed (i.i.d.) data represented in a single table can be viewed as a special limiting case of multi-relational data with no relationships [24, 23]. Statistical-relational learning is a recent field at the intersection of databases and machine learning that aims to extend i.i.d. machine learning methods to multi-relational data [10]; this is called *upgrading* the methods [19]. The field of statistical-relational learning has developed various statistical models for relational databases [9]. From the set of these statistical models, Markov Logic Networks (MLNs) and Bayesian Networks (BNs) achieved impressive performance on a variety of statistical-relational learning tasks. MLNs are based on undirected graphical models and BNs are based on the directed graphical models.

A multi-relational graphical model, such as MLN and BN, provides an integrated statistical analysis of the interdependent data resources in the database. Our focus is on bayesian networks; Figure 1.1 shows a simple bayesian network with 4 random variables (nodes) and 4 dependency edges.

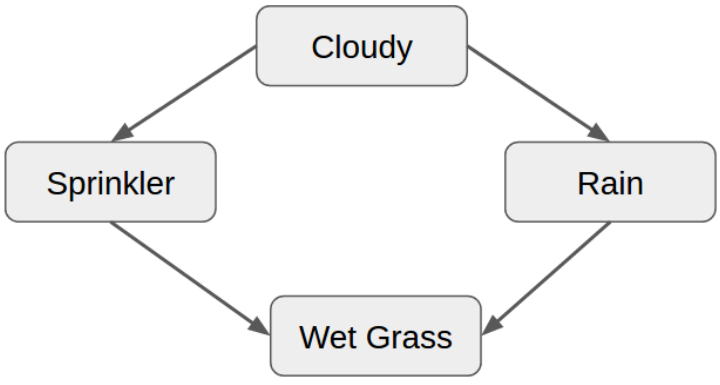

Figure 1.1: Simple Bayesian Network example.

Statistical-relational models have achieved state-of-the-art performance in a number of application domains, such as natural language processing, ontology matching, information extraction, entity resolution, link-based clustering, query optimization, etc. [6, 25, 11]. Database researchers have noted the usefulness of statistical-relational models for knowledge discovery and for representing uncertainty in databases [39, 40]. Multi-relational data have a complex structure, which integrates heterogeneous information about different types of entities and relationships. An integrated statistical analysis of the heterogeneous and independent complex data resources can be provided by statistical-relational models [29].

In this thesis we focus on learning Bayesian network structures, which are directed acyclic graphs whose nodes are random variables. The most widely used approach to Bayesian network structure learning is search+score: find a structure that optimizes a model selection criterion for a given dataset. We propose a general definition for extending bayesian network model selection scores designed for single-table i.i.d. data to relational databases. Our model comparison upgrade method satisfies three desiderata: *Generalization*, *Relative Consistency*, and *Balance*; which will be discussed later on.

Motivation. Generalization is a widely accepted principle for relational learners [19, 17]. Balance is one of the fundamental properties of standard model selection scores [33]. For example, in a Bayesian view, both likelihood and penalty terms are log-probabilities [4]. Consistency has been widely applied as a theoretical criterion in model selection theory [41] for i.i.d. data, and increasingly for relational data as well [34, 42, 38]. Informally, it means that as the amount of available data increases, the method selects a model structure that matches the true data generating process.

Contributions. Our main contributions are twofold, and may be summarized as follows.

1. A novel method for upgrading an i.i.d. BN structure score to relational databases. The method upgrades the score by defining a gain function that compares the relative fit of two graph structures on relational data.
2. A relative consistency proof: if the original score is consistent for i.i.d. data, the upgraded gain function is consistent for relational data. To our knowledge this is the first consistency result for multi-relational structure learning.

Organization. We review background on Bayesian networks and relational data. Then we define our rescaling method for upgrading model selection scores, as well as baseline upgrade methods for comparison. Theoretical analysis demonstrates the consistency of the rescaling method, and the nonconsistency of the baseline scores. Empirical evaluation compares the different Bayesian networks selected with respect to data fit and model complexity. We review related work after we have presented the background and the details of our own approach.

Chapter 2

Background and Notation

In this chapter, we provide the necessary background on relational data, bayesian networks, random variables for relational data, and how to define bayesian networks comprising relational data. In each section we will describe the concept and common accepted terminology for that concept. While we introduce no new terminology, in some cases the same concept has been given different names by different communities. In that case we employ the name that is most suggestive for our purposes. In section 2.1 we review basic probability concepts and notations needed to define a bayesian network. In section 2.2 we discuss about the Bayesian Networks and its concepts, having an example helps to make it more clear for the first view. Then we go more into the First-Order Bayesian Networks in section 2.3, which is the sort of Bayesian Network the rest of our work is on it. We will discuss about the relational data and relational random variables in that section and show how we can show that in Bayesian Networks.

2.1 Probability Distribution

We assume general concepts and notations as the general familiarity with probability theory. We use capital letters, sometimes with an index, to denote **random variables** (e.g. $X, Y, X_1, X_2, \ldots$); the same notation with a lowercase letter denote the value of a random variable (e.g. $x, y, x_1, x_2, \ldots$).

We use boldface to denote sets and lists of objects; for instance, $\boldsymbol{a} = \{a_1, a_2, \ldots, a_n\}$. The notation $|S|$ denotes the cardinality of a set S. Fix a set of random variables $\boldsymbol{X} = \{X_1, X2 \ldots, X_n\}$. The possible values of X_i are enumerated as $\{x_{i_1}, \ldots, x_{i_k}\}$. The notation $P(X_i = x)$, or $P(x)$ if we know X_i is our subject random variable, denotes the probability of random variable X_i taking on value x. We also use the set notation $P(\boldsymbol{X} = \boldsymbol{x}) \equiv P(\boldsymbol{x})$ denotes the **joint probability** of random variables $X_1, X2 \ldots, X_n$ taking on values $x_1, x_2, \ldots, x_n$. The sum of joint probability distribution of a set of random variables $\boldsymbol{X}$ over all possible values $\boldsymbol{x}$ is 1; which means,

$$\sum_{\boldsymbol{x}} P(\boldsymbol{X} = \boldsymbol{x}) = 1. \tag{2.1}$$

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

$$P(\boldsymbol{Y}) = \sum_{z} P(\boldsymbol{Y}, \boldsymbol{Z} = \boldsymbol{z}) \tag{2.2}$$

The **conditional probability** of a subset $\boldsymbol{Y}$ of $\boldsymbol{X}$ given a subset $\boldsymbol{Z}$ of $\boldsymbol{X}$ is the probability of $\boldsymbol{Y}$ occurring when we know $\boldsymbol{Z}$ has occurred. The conditional probability can be obtained by

$$P(\boldsymbol{Y}|\boldsymbol{Z}) = \frac{P(\boldsymbol{Y}, \boldsymbol{Z})}{P(\boldsymbol{Z})} \tag{2.3}$$

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

$$P(\boldsymbol{X}, \boldsymbol{Y}) = P(\boldsymbol{X}) \times P(\boldsymbol{Y}) \tag{2.4}$$

2.2 Bayesian Networks

Machine learning is a subfield of computer science concerned with the design and development of algorithms and techniques that given a dataset and an objective function, search through the hypothesis space or models for the most suitable candidate such that maximizes objective function. Arthur Samuel defined the field of Machine Learning in 1959 as "Machine learning gives computers the ability to learn without explicitly programmed", which is still true from the bird's-eye view over the current machine learning methods. A Bayesian Network is one of the possible solutions for this kind of learning problems. There is no single model as the best solution for all kind of machine learning problems, it is not found by now at least.

A Bayesian Network is a probabilistic graphical model which consists of a set of random variables, as the nodes in the graphical model, and their conditional dependencies, as the edges in the graphical model. Conditional dependencies should make a Directed Acyclic Graph (DAG) in order to make a Bayesian Network structure. For example, as it shows in the figure 2.1 we have 4 random variables: Cloudy, Sprinkler, Rain, WetGrass; Each of those can get binary values "True" and "False". This bayesian network structure could represent the probabilistic relationships between having a Cloudy weather and the grass being wet. All the other relationships between the four random variables could be computed from the structure and the probability tables.

Therefore, a Bayesian Network structure is a directed acyclic graph G (DAG) whose nodes comprise a set of random variables [26]. Depending on context, we interchangeably refer to the nodes and (random) variables of a bayesian network. A Bayesian network B is a structure G together with a set of parameter values. To have a better understanding about the random variables and probability distributions, we first covered a brief overview on the basics and notations of probability distribu-

tions in section 2.1. Each node is associated with a probability function that takes a particular set of values for the node's parent variables, and gives the probability distribution of the variable represented by the node. For example in the figure 2.1 random variable "Sprinkler" takes the probability values of the random variable "Cloudy" as input, and gives the conditional probability distribution for "Sprinkler" node, which is shown in the table besides the node. It is the same for "WetGrass"; except it has two parents, so the conditional probability distribution would depend on Sprinkler and Rain random variables.

Having a bayesian network structure with the probability tables for all the nodes is like having answer for all the probabilistic queries about random variables in the bayesian network. But the problem arises when we are thinking about the structure. Which random variables do we need for a specific task? Having all the variables we know they can be relate to our problem, how we should define their relationship? Which nodes are directly connected and in which direction? These are the main questions that Structure Learning (Model Selection) for bayesian networks is trying to solve. It used to be done by experts in the field, one who have some overall knowledge about the variables. For example, in a simple example of figure 2.1 one can easily say that "if the sprinkler is on, it doesn't mean that it is rainy or not", means that Sprinkler and Rain are two independent variables in this case. Trying to solve the model selection problem by rule-based Machine Learning techniques is the field of study that wants to do the work of the experts automatically, and sometimes we do not have any experts in some sort of data. Statistical Relational Learning is one of the common approaches to tackle this problem using a score function based on the bayesian network structure to guide the structural search and augment the network.

The parameters of a Bayesian network specify the distribution of a child node given an assignment of values to its parent node. For an assignment of values to its nodes, a bayesian network defines the joint probability as the product, over all nodes, of the conditional probability of the node value given its parent values. Thus the log-joint probability is given by

$$\log P_B(\boldsymbol{X} = \boldsymbol{x}) = \sum_{i=1}^{n} \log P_B(X_i = \boldsymbol{x}_i | \mathbf{Pa}_i^G = \mathbf{pa}_i^G). \tag{2.5}$$

where $\boldsymbol{x}_i$ resp. $\mathbf{pa}_i^G$ is the assignment of values to node X_i resp. the parents of X_i determined by the assignment $\boldsymbol{x}$.

Figure 2.1 shows an example of a first-order bayesian network. Each of the nodes in the bayesian network can get value "True" or "False". The probability of each node value depends on its parents. Probability tables are also shown in Figure 2.1; each table shows the probability of a node (random variable) being "True" or "False" based on its parents (conditional probability).

2.3 First order Bayesian Networks

A First-Order Bayesian network (FOB) [40], aka Parametrized bayesian network [16], is a Bayesian network whose nodes are first-order terms. Following Halpern's well-known random selection se-

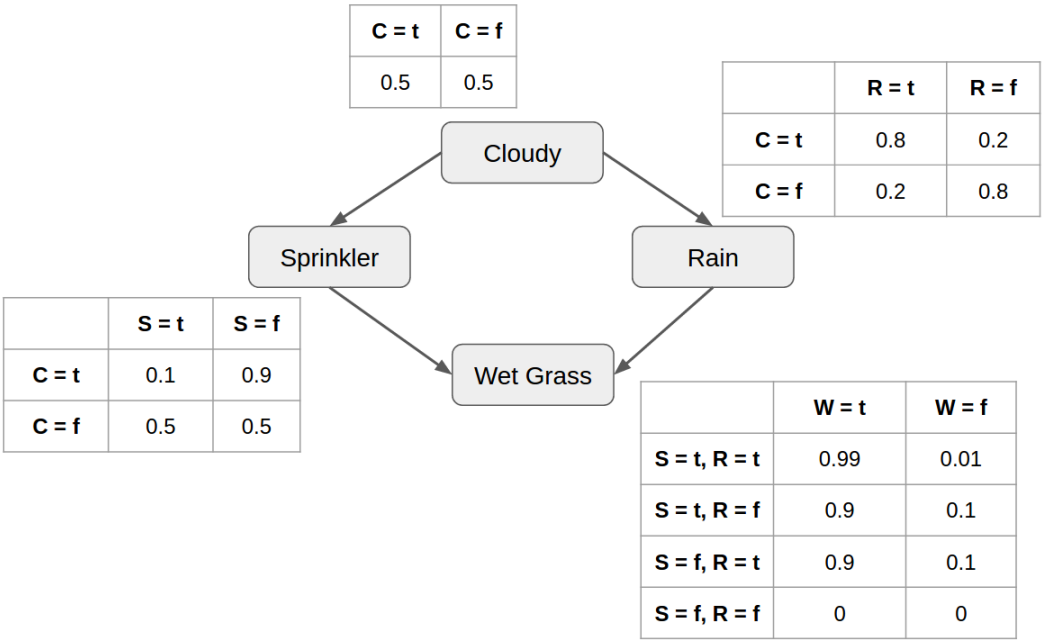

Figure 2.1: Example First-Order Bayesian networks.

mantics for probabilistic logic, a FOB can be viewed as a model of database frequencies [37]; in the terminology of [8], a FOB can be used as a Statistical-Relational Model (SRM). The basic idea is to view a population variable as a random selection of individuals from the variable's population. Functors are then functions of random variables, hence themselves random variables. For example,

$$P(Sprinkler = false, Cloudy = true) = 0.9$$

can be read as "the probability is 0.9 that the sprinkler is off and the weather is cloudy".

2.3.1 Relational Data

In this section we are going to go through the relational data. We first start with the definition and then mention the relation to our work.

As an example we are going to work on it in this section suppose we have the bayesian network of figure 2.2,

$$P(Rating(User, Movie) = 1, Age(User) = 1) = 0.00122$$

can be read as "if we randomly select a user and a movie, the probability is 0.00122 that the user has age level 1 and rates the movie 1". The topic of the next section is quantifying how well the joint distribution $P_B(\cdot)$ represented in an SRM via Equation (2.5) fits the database distribution $P_{\mathcal{D}}(\cdot)$ defined by Equation (2.6).

Example Figure 2.2 shows an example of two small Bayesian networks. The rating value is n/a (for "not applicable") if and only if the user has not rated the movie (cf. [33]). For the 2-node BN, Equation 2.5 yields

$$\begin{aligned} P(Rating(User, Movie) = 1, Age(User) = 1) \\ = 0.00297 \cdot 0.41205 \approx 0.00122. \end{aligned}$$

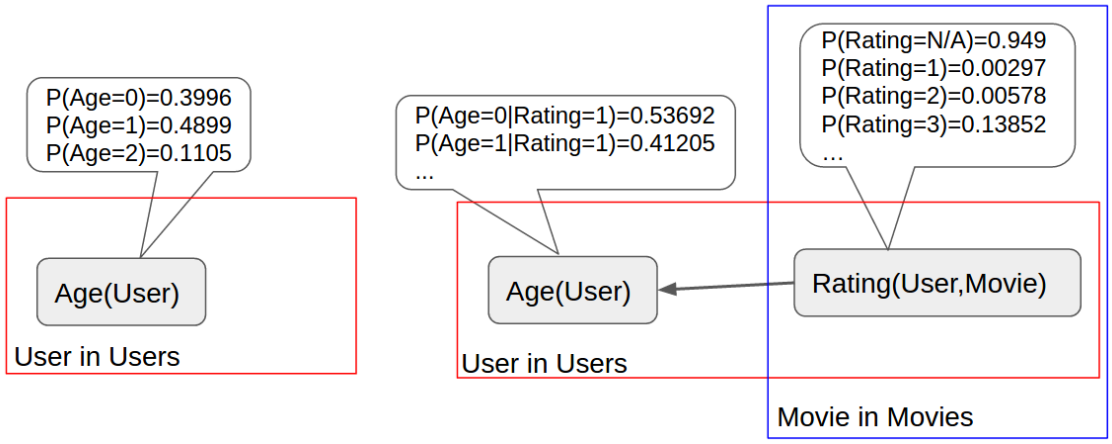

Figure 2.2: Example First-Order Bayesian networks. The type of population variables is shown as in a plate model. Conditional probability parameters were estimated from the MovieLens database.

Relational Data Frequencies

We adopt a function-based formalism for combining logical and statistical concepts [28, 16, 32]. Table 2.1 compares statistical concepts for relational and for i.i.d. data.

	Representation	Population	Instances	Sample Size	Empirical Frequency
i.i.d Data	Single Table	Single	Rows	Single N	Sample Frequency
Relational Data	Multiple Tables	Multiple	Groundings	Multiple N, one for each population	Database Frequency

Table 2.1: Statistical concepts for relational vs. i.i.d. data. With a single population and unary functors only, the relational concepts reduce to the i.i.d. concepts.

A multi-relational model is typically a multi-population model. A **population** is a set of individuals of the same type (e.g., a set of $Actors$, a set of $Movies$). Individuals are denoted by lower-case constants (e.g., $brad\_pitt$ and $braveheart$). A k-ary **functor**, denoted f, f' etc., maps a tuple of k individuals to a value. Propositional or i.i.d. data are represented by unary ($k = 1$) functors that return the value of an attribute (column) for each individual (row) from a single population [24]. Binary functors can be represented as matrices, $k > 2$-ary functors as tensors of order k. The arguments of functors are restricted to appropriate types. Each functor takes values from a

set of constants called the **domain** of the functor. Like [28], we assume that (1) the domain of each functor is finite, and (2) that functor values are disjoint from individuals.

Figure 2.3 provides an example of a toy database. The example follows the closed-world convention: if a relationship between two individuals is not listed, it does not obtain. A (complete) relational dataset or **database** $\mathcal{D}$, specifies:

1. A finite sample population $\mathcal{I}_1, \mathcal{I}_2 \ldots$, one for each type.
2. The values of each functor, for each input tuple of observed sample individuals of the appropriate type.

User		
User_id	**Age**	**Gender**
3	0	M
5	1	F
7	2	M
...		

Rating		
User_id	**Movie_id**	**Rating**
3	The Dictator	1
5	Thor	4
5	The Dictator	3
7	BraveHeart	5
...		

Movie			
Movie_id	**Action**	**Drama**	**Horror**
The Dictator	0	0	0
Thor	1	0	0
BraveHeart	1	1	1
...			

Figure 2.3: Excerpt from a relational dataset/database.

Relational Random Variables

A **population** variable ranges over a population, and is denoted in upper case such as $Actor, Movie, \mathbb{A}, \mathbb{B}$. A (functional) **term** is of the form $f(\tau_1, \ldots, \tau_k)$ where each τ_i is a population variable or a constant/individual of the appropriate type. A term is **ground** if it contains only constants; otherwise it is a **first-order term** with at least one population variable.

A first-order random variable (FORV) is a first-order term [40, 16]. FORV examples are $gender(Actor)$ and $AppearsIn(Actor, Movie)$. When the special syntactic structure of a FORV is not important, we use the traditional random variable notation like X, Y.[1] A FORV can be viewed as a template, instantiated with individual constants, much like an index in a plate model [16]; see Figure 2.2. An instantiation or **grounding** for a list of FORVs simultaneously replaces each population variable in the list by a constant. The **number of possible groundings** of a joint assignment is given by

$$N\left[\boldsymbol{X} = \boldsymbol{x}; \mathcal{D}\right] \equiv N\left[\mathbb{A}_1; \mathcal{D}\right] \times \cdots \times N\left[\mathbb{A}_m; \mathcal{D}\right]$$

[1] Unfortunately the tradition in statistics clashes with the equally strong tradition in logic of using X, Y to denote population variables.

Table 2.2: Computing the database frequency of a joint assignment to first-order random variables on MovieLens database.

$X = x$	$n(X = x; D)$	$N(X = x; D)$	$P_D(X = x; D)$
Age(User)=0	376	941	0.3996
Age(User)=0, Rating(User, Movie)=1	2524	1582762	0.0016

where $N[\mathbb{A}; \mathcal{D}]$ is the size of the sample population associated with variable $\mathbb{A}$. Applying a grounding to a joint assignment $\boldsymbol{X} = \boldsymbol{x}$ results in a joint assignment of values to ground terms. A database satisfies (agrees with) the ground assignment or not. The **number of satisfying groundings** of a joint assignment in database $\mathcal{D}$ is denoted by $\mathrm{n}[\boldsymbol{X} = \boldsymbol{x}; \mathcal{D}]$. The **database frequency** [12, 36] is the number of satisfying instances over the number of possible instances:

$$P_{\mathcal{D}}(\boldsymbol{X} = \boldsymbol{x}) = \frac{\mathrm{n}[\boldsymbol{X} = \boldsymbol{x}; \mathcal{D}]}{N[\boldsymbol{X} = \boldsymbol{x}; \mathcal{D}]}. \tag{2.6}$$

The database frequency is the counterpart to the sample distribution for i.i.d. data. Table 2.2 illustrates database frequencies. In the first row, the number of users is 941, and the number of users at age level 0 is 376, so the frequency of age 0 users is 376/941. In the second row, the number of user-movie pairs is 1,582,762. The number of user-movie pairs where the user has age level 0 and assigned a rating of 1 is 2,524. So the database frequency of this event is 2,524/1,582,762.

Chapter 3

Theoretical Results

In this chapter we will first go through the previous works on model selection area, mostly related to relational data, then we will continue discussing our approach and our model definition and how we tried to tackle the problems in the previous works.

3.1 Literature Review

We will review related work on model selection for i.i.d. and for relational data.

3.1.1 Bayesian Network Model Selection for i.i.d. data

Model selection criteria are a major topic in statistics [41]. Model selection criteria for Bayesian Network learning are a classic topic in machine learning; for review see [1]. Our work extends the theoretical analysis of Bayesian Network learning to multi-relational data.

3.1.2 Bayesian Network Model Selection for Multi-Relational data

For relational data, a likelihood function requires aggregating model probabilities for multiple instances of the template Bayesian Network [16]. A number of different aggregation mechanisms have been proposed, resulting in different Bayesian Network likelihood functions.

Likelihood based on Random Instantiations

The *random selection log-likelihood score* is defined as the expected log-likelihood from a random grounding [35]. The random selection log-likelihood can be seen as an application of Halpern's well-known random selection semantics for first-order probabilistic logic [12]. The frequency log-likelihood of Table 3.1 is a closed form for the random selection log-likelihood [36, Prop.4.1]. For model selection, [36] suggests using what we call the normalized-count score, but does not investigate model selection scores in detail.

Likelihood based on Complete Instantiations

This type of likelihood is based on unrolling or grounding the BN [14, 23, 28]. An inference graph contains all possible instances of edges in the first-order template, obtained by replacing population variables with constants. The inference model defines a conditional probability $P(X_{ij}|\mathbf{Pa}_{ij}^G)$, where X_{ij} denotes the j-th grounding of node i in the template BN. This conditional probability aggregates the information from the multiple parent instances of the ground node X_{ij}. Assuming that the ground inference graph is acyclic, a log-likelihood function can be defined by the usual BN formula

$$L(G, \mathcal{D}) \equiv \sum_i \sum_j^{\gamma_i} \ln P(X_{ij}|\mathbf{Pa}_{ij}^G). \tag{3.1}$$

There are two main approaches for defining the conditional probability model $P(X_{ij}|\mathbf{Pa}_{ij}^G)$, depending on how multiple instances of the same parent configuration are included [22]: Using (1) aggregate functions [9] and (2) combining rules [15]. Model selection scores have been defined for both aggregators [9] and combining rules. To our knowledge, the consistency of these model selection scores has not been investigated. Another open problem with the complete instantiation approach is that the ground inference graph may contain cycles even if the template Bayesian Network structure does not [20].

Previous application of the Learn-and-Join algorithm [36] used a BN learner for i.i.d. data as a subroutine for learning a first-order BN. Combined with a log-linear prediction model, the features in the learned BN provide accurate predictions for the values of ground terms. While this upgrades BN learning algorithms, it does not upgrade i.i.d. model comparison criteria. In this paper we combine the LAJ search strategy with upgraded i.i.d. model scores.

3.1.3 Markov Logic Networks

A Markov Logic Network (MLN) can be viewed as a template model for an *un*directed graph.

Model Selection

Because computing the normalization constant (partition function) for the log-linear MLN likelihood function is generally intractable, Markov structure learning often optimizes the pseudo log-likelihood [20, 18]. This is the sum of the log-conditional probabilities of each ground node value, given the values of all other ground nodes. Similar to our normalized log-likelihood scores, the weighted pseudo log-likelihood (WPLL) normalizes the pseudo log-likelihood counts of different target nodes X_{ij} by the total number γ_i of the groundings of template node X_i. Each weight is penalized with a Gaussian shrinkage prior. The closest counterpart in our experiments is the normalized-count AIC score of Table 3.5. The WPLL+penalty score is not consistent for the same reason as the normalized-count AIC score.

Parameter Estimation

Xiang and Neville [42] provide a sophisticated asymptotic analysis of parameter learning for MLNs, given locality assumptions about the correlations between ground nodes. Our paper shares their framework of learning from one network. Similar to our paper, they consider a normalized log-likelihood score. Their main theorem shows that maximizing either the normalized log-likelihood or the pseudo log-likelihood leads to consistent parameter estimates. Different from our paper, they do not consider the consistency of MLN model selection.

3.1.4 Other Models

The Inductive Logic Programming FOIL system [31] defined the information gain that results from adding a new condition (literal) to a first-order rule. The first-order information gain is similar to our approach in that 1) it defines a gain function rather than a score, and 2) the key issue concerns adding population variables. It is different in that 1) it is applied with discriminative models (classification), rather than generative models, and 2) different rule groundings are combined using existential quantification rather than a log-linear model.

A popular approach to modelling relational data is to learn latent factors such that links are independent of each other given the latent factors. Sakai and Yamanishi [34] provide an asymptotic analysis of relational clustering when the number of clusters is selected to optimize normalized minimum description length. In this paper we did not consider latent factor learning, so our consistency result is not derived from conditional independence assumptions.

3.2 Relational Model Comparison

We define the relational model comparison scores that we study in this paper. We begin with general model selection concepts.

3.2.1 Model Score Concepts

A score $S(G, \mathcal{D})$ measures how well a DAG G fits a database $\mathcal{D}$ [3]. A **gain function** $\mathcal{D}(G, G', \mathcal{D})$ measures how much an alternative structure G' improves a current structure G. For every score S, there is an associated gain function defined by $\Delta_S(G, G', \mathcal{D}) = S(G', \mathcal{D}) - S(G, \mathcal{D})$. A score function is **decomposable** if it can be written as a sum of local scores s, each of which is a function only of one node and its parents; see Equation (3.2). Similarly, a gain function is decomposable if the improvement can be written as a sum of local improvements δ; see Equation (3.3). Many structure search algorithms consider adding the addition or deletion of a single edge at a time [3]. Let $X_+ \rightarrow X_i$ be an edge not contained in G, and let G_+ be the graph that adds the edge to G. In that case we write the local gain only as a function of the previous parents and the new parent X_+; see Equation (3.4).

$$S(G, \mathcal{D}) = \sum_i s(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \tag{3.2}$$

$$\Delta(G, G', \mathcal{D}) = \sum_i \delta(X_i, \mathbf{Pa}_i^G, \mathbf{Pa}_i^{G'}, \mathcal{D}) \tag{3.3}$$

$$= \delta(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D}) \tag{3.4}$$

We consider upgrading an i.i.d. score of the form (log-likelihood of data under model) - (penalty = function of number of parameters, sample size). We first discuss upgrading the log-likelihood term.

3.2.2 Relational Model Likelihood Scores

Whereas the independence of i.i.d. data induces a unique product likelihood function, several BN likelihood functions have been proposed for multi-relational data. The reason is that template BN structures represent multiple dependent instantiations, and there are different approaches to aggregating them; see Section 3.1. We focus on the most recent proposal, *log-linear likelihood scores* [35], whose form is similar to the log-linear likelihood of Markov Logic Networks (Section 3.1). A log-linear likelihood score is a factor product that is not necessarily normalized (not summing to 1). Log-linearity has the following advantages [35]. (1) Generalizing the i.i.d. case: The mathematical form is very close to the i.i.d. log-likelihood function for Bayesian networks. (2) Tractability: The score is maximized by the empirical conditional frequencies, as in the i.i.d. case. It can therefore be computed in closed form, given the sufficient statistics in the data. (3) Autocorrelation: The score is well-defined even when the data exhibit cyclic dependencies (see Section 3.1).

We adopt standard notation for BN sufficient statistics [13]. Let $X_i = x_{ik}, \mathbf{Pa}_i^G = \mathbf{pa}_{ij}^G$ be the assignment that sets node i to its k-th value, and its parents to their j-th possible configuration. We associate the following concepts with the ijk assignment.

- $\mathrm{n}_{ijk}^G(\mathcal{D}) \equiv \mathrm{n}\left[X_i = x_{ik}, \mathbf{Pa}_i^G = \mathbf{pa}_{ij}^G; \mathcal{D}\right]$ is the number of groundings that satisfy the ijk assignment.
- $\mathrm{n}_{ij}^G(\mathcal{D}) \equiv \sum_k \mathrm{n}_{ijk}^G(\mathcal{D})$ is the number of groundings that satisfy the j-th parent assignment.
- $\mathrm{n}_i^G(\mathcal{D}) \equiv \sum_j \sum_k \mathrm{n}_{ijk}^G(\mathcal{D})$ is the number of possible groundings for node i.

The number of possible groundings can be directly computed as the product of the sample size $N[\mathbb{A}; \mathcal{D}]$ associated with each population variable $\mathbb{A}$ that is contained in node i or its parents. Since the quantity n_i^G plays the same role as the sample size in i.i.d. data, we refer to it as the **local sample size** for node i. A key difference is that the *local sample size* n_i^G *depends on both the data and the graph structure* [20]. In contrast, the global sample size n in i.i.d. data is the same for all nodes and

Table 3.1: Relational Local (Pseudo) Log-likelihood Scores.

Name	Symbol	Definition
Count	$L(X_i, \mathbf{Pa}_i^G, \mathcal{D})$	$\sum_j \sum_k \mathrm{n}_{ijk}^G(\mathcal{D}) \cdot \log_2 \left(\frac{\mathrm{n}_{ijk}^G(\mathcal{D})}{\mathrm{n}_{ij}^G(\mathcal{D})} \right)$
Frequency	$\overline{L}(X_i, \mathbf{Pa}_i^G, \mathcal{D})$	$1/\mathrm{n}_i^G(\mathcal{D}) \times L(X_i, \mathbf{Pa}_i^G, \mathcal{D})$

Family Configuration	n_{ijk}	n_{ij}	n_i	n_{ijk}/n_i	*CP*	L	$\overline{L}$
Age(User)=0	376	—	941	0.3996	0.3996	-497.6217	-0.5288
Age(User)=0, Rating(User,Movie)=1	2524	4703	1582762	0.0016	0.5367	-2266.2224	-0.0014

Table 3.2: A population-adding edge to expand a structure G to a larger structure G_+. CP-table for G and G_+. The parameter values are maximum likelihood estimates computed from the Movielens database. The L and $\overline{L}$ columns show the contributions of the family configuration in each row (part of the sum for the total likelihood values).

for all graphs. The question of how to compare models with different local sample sizes is the main question of this paper.

Table 3.1 gives the formulas for two previously proposed relational log-likelihood scores. The **count log-likelihood** has the same form as the local log-likelihood for i.i.d. data, but replaces counts in a single data table by counts in a database. The **frequency likelihood** [35] normalizes the count score by the local sample size (see also [42]). This is equivalent to replacing counts in a single data table by frequencies in a database.

For i.i.d. data, adding an edge to a graph G can only increase the log-likelihood score. A big difference to the relational case is that *adding an edge can decrease the relational count log-likelihood.* This occurs when the new edge **adds a population variable** that was not contained in the child node or the previous parents; see Figure 2.2 and Table 3.2. In contrast, adding an edge cannot decrease the frequency log-likelihood. Table 3.2 illustrates the computations of the different log-likelihood scores.

3.2.3 The Normalized Gain Function

Here and below, we write G_+ for the DAG that results from adding a generic edge $X_+ \to X_i$ to DAG G. Our upgrade method for defining a relational gain function is as follows.

1. Compute the likelihood differential using the frequency likelihood $\overline{L}$.
2. Normalize the penalty term differential by the *larger* sample size $\mathrm{n}_i^+(\mathcal{D})$.
3. The **normalized gain** is computed as (1) minus (2), likelihood differential minus penalty differential.

Table 3.3 gives the normalized gain formulas for upgrading the standard AIC and BIC scores [1]. (We omit a constant factor of 2 that does not affect model comparisons.) The upgrade method can be applied with other i.i.d. scores as well. We focus on AIC and BIC because of their relatively simple form. Table 3.6 and Table 3.7 show example values for the normalized gains. These are compared with gains based on natural single model scores, which we introduce in the next section.

Local Gain Function	Definition
$\Delta\overline{L}(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D})$	$\overline{L}(X_i, \mathbf{Pa}_i^G \cup X_+, \mathcal{D}) - \overline{L}(X_i, \mathbf{Pa}_i^G, \mathcal{D})$
$\Delta\overline{AIC}(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D})$	$\Delta\overline{L}(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D}) - \frac{\Delta pars(X_i, \mathbf{Pa}_i^G, X_+)}{\mathrm{n}_i^+(\mathcal{D})}$
$\Delta\overline{BIC}(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D})$	$\Delta\overline{L}(X_i, \mathbf{Pa}_i^G, X_+, \mathcal{D}) - \frac{\frac{1}{2}\log_2(\mathrm{n}_i^+(\mathcal{D}))\Delta pars(X_i, \mathbf{Pa}_i^G, X_+)}{\mathrm{n}_i^+(\mathcal{D})}$

Table 3.3: Our Proposed Normalized Relational Model Gain Upgrade for AIC and BIC. $\mathrm{n}_i^+(\mathcal{D}) \equiv \mathrm{n}_i^{G_+}(\mathcal{D})$ denotes the local sample size for the expanded graph. $\Delta pars(X_i, \mathbf{Pa}_i^G, X_+) = \#pars(X_i, \mathbf{Pa}_i^{G^+}) - \#pars(X_i, \mathbf{Pa}_i^G)$

3.3 Comparison Multi-Relational Model Scores

The baseline methods define relational scores by normalizing the likelihood term and/or the penalty term. This upgrade scheme defines 4 different relational versions of a model selection score; see Table 3.4. Normalizing the penalty term but not the likelihood term is clearly inadequate. Table 3.5 gives the formulas for the remaining 3 different relational versions of AIC and BIC. Table 3.6 and Table 3.7 show example values.

The next proposition establishes that the normalized gain function defines a new concept, in that it cannot be represented as a difference of single model selection scores. In fact, we give a stronger result: no scaled version of the normalized gain function can be represented as a difference associated with a single model selection score. A scaling corresponds to a *unit change* that can be represented as a positive linear transformation [α (normalized gain) $+ c$] where $\alpha > 0$ may depend only on the local sample sizes of the structures compared.

Proposition 1. *There is no relational model selection score such that its associated model gain function is a unit change applied to the normalized gain function.*

Table 3.4: Available options for each score, depending on whether rescaling the likelihood count and/or the penalty term.

Likelihood \ Penalty	Count	Normalized
Count	S; not consistent; underfits	—
Normalized	$\widetilde{S}$; not consistent; underfits	$\overline{S}$; not consistent; overfits

Score	AIC_i	BIC_i
Count-Count	$AIC(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv L(X_i, \mathbf{Pa}_i^G, \mathcal{D}) - \#pars(X_i, \mathbf{Pa}_i^G)$	$BIC(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv L(X_i, \mathbf{Pa}_i^G, \mathcal{D}) - \frac{1}{2}\log_2(\mathrm{n}_i^G(\mathcal{D})) \cdot \#pars(X_i, \mathbf{Pa}_i^G)$
Normalized-Normalized	$\overline{AIC}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv \frac{1}{\mathrm{n}_i^G(\mathcal{D})} AIC(X_i, \mathbf{Pa}_i^G, \mathcal{D})$	$\overline{BIC}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv \frac{1}{\mathrm{n}_i^G(\mathcal{D})} BIC(X_i, \mathbf{Pa}_i^G, \mathcal{D})$
Normalized-Count	$\widetilde{AIC}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv \overline{L}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) - \#pars(X_i, \mathbf{Pa}_i^G)$	$\widetilde{BIC}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) \equiv \overline{L}(X_i, \mathbf{Pa}_i^G, \mathcal{D}) - \frac{1}{2}\log_2(\mathrm{n}_i^G(\mathcal{D})) \cdot \#pars(X_i, \mathbf{Pa}_i^G)$

Table 3.5: Relational Local Model Selection Scores, count and frequency versions. Normalized scores divide count scores by the local sample size $\mathrm{n}_i^G(\mathcal{D})$.

					AIC			
	#groundings	#parameters	L	$\overline{L}$	coun-count	normalized-normalized	normalized-count	normalized gain
G	941	2	-1302.7	-1.384	-1304.7	-1.3865	-3.38	—
G+	1582762	12	-1863691.4	-1.177	-1863703.4	-1.1775	-13.18	—
GAIN		10	-1862388.7	0.207	-1862398.7	0.2090	-9.79	0.20684

Table 3.6: Example Values for the Scores and Gain Functions defined for AIC in this section. For the structures of Figure 2.2.

We omit the proof. Intuitively, the reason is that a balanced gain function has to adjust the score of the smaller graph G depending on whether the new edge $X_+ \rightarrow X_i$ adds a population variable or not. This is impossible if the score for G is fixed. *The proposition shows that no balanced log-linear gain function is representable in terms of a log-linear single model selection score.* Our argument for this conclusion is as follows. The normalized gain is balanced because the likelihood scores and the penalty terms are standardized to the same scale. Therefore any balanced log-linear gain function should be a scaling (unit change) of the normalized gain. (Much like any valid temperature scale is a unit change of the centigrade scale.) Thus Proposition 1 should apply to any balanced log-linear gain function.

Example Consider the *rescaled gain function* $[\mathrm{n}_i^+(\mathcal{D}) \times$ (normalized gain)$]$. This is equivalent to the following upgrade procedure. (1) Multiply the count likelihood for the smaller graph G by the factor $\frac{\mathrm{n}_i^+(\mathcal{D})}{\mathrm{n}_i^G(\mathcal{D})}$. Since the term $\frac{\mathrm{n}_i^+(\mathcal{D})}{\mathrm{n}_i^G(\mathcal{D})}$ represents the number of possible groundings added by the expansion G_+, this puts the count likelihoods for both graphs on the same scale. (2) Return the resulting score differential. For instance, the rescaled gain function for AIC is given by the formula

$$L(X_i, \mathbf{Pa}_i^G \cup X_+, \mathcal{D}) - \left(\frac{\mathrm{n}_i^+(\mathcal{D})}{\mathrm{n}_i^G(\mathcal{D})} \times L(X_i, \mathbf{Pa}_i^G, \mathcal{D})\right) - \Delta pars(X_i, \mathbf{Pa}_i^G, X_+).$$

The rescaled gain function is balanced, since counts and parameters are measured on the same scale. Proposition 1 entails that the rescaled gain function cannot be represented by a single model

					BIC			
	#groundings	#parameters	L	$\overline{L}$	coun-count	normalized-normalized	normalized-count	normalized gain
G	941	2	-1302.7	-1.384	-1312.5	-1.3948	-11.26	—
G+	1582762	12	-1863691.4	-1.177	-1863814.9	-1.1776	-124.74	—
GAIN		10	-1862388.7	0.207	-1862502.4	0.2173	-113.48	0.20678

Table 3.7: Example Values for the Scores and Gain Functions defined for BIC in this section. For the structures of Figure 2.2.

selection score. Conversely, the log-linear single model selection scores of Table 3.5 are not balanced. As we show in the next section, their lack of balance leads to nonconsistency.

3.4 Consistency Analysis

This section shows that the normalized gain function is relatively consistent, [preserves consistency] but the model score functions are not. The reason is that the former are balanced and the latter are not. We observe that (*) *for edges that do not add population variables, the normalized gain, the count-count, and the normalized-normalized comparisons are equivalent.* This is because in this case the local sample sizes are the same for both structures (i.e., $\mathrm{n}_i^G(\mathcal{D}) = \mathrm{n}_i^+(\mathcal{D})$). Therefore our analysis focuses on the case where the model comparisons differ: edges that do add population variables.

3.4.1 Local Consistency

A large sample analysis considers a sequence of samples from the true data generating distribution that grow without bound. A model selection method is consistent if the sequence of models that it selects converges to one that is correct for the data generating distribution. Following previous work on consistency for relational data [34, 42, 38], we formalize these concepts as follows. In the limit the sample size for each population $\mathcal{I}_i$ goes to infinity, which like [34] we denote as

$$\boldsymbol{N}(\mathcal{D}) \to \infty.$$

Like [42], we make the identifiability assumption that

$$P_{\mathcal{D}}(\cdot) \to P_w(\cdot) \equiv p \text{ as } \boldsymbol{N}(\mathcal{D}) \to \infty.$$

Here w represents a complete relational structure (network) from which samples are drawn. We abbreviate the frequency distribution associated with the complete structure as p, for brevity and compatibility with [3] where p is used for the data generating distribution. Arbitrarily large samples can be generated either by sampling with replacement, or by sampling from infinite populations. For discussion of network sampling, see [38, 7]. The definition of the relational frequency $P_w(\cdot)$ with infinite populations is given in [12]. Chickering and Meek analysed BN model selection scores in terms of **local consistency**, which we adapt for score gain functions as follows.

Definition 1. *Let p be the data generating distribution. A local gain function is **locally consistent** if, the following hold in the sample size limit as $\boldsymbol{N}(\mathcal{D}) \to \infty$, for any single-edge expansion G_+:*

1. *If X_+ is not independent of X_i given $\mathbf{Pa}_i^G$ in p, then*
   $\Delta(G, G_+, \mathcal{D}) > 0$.

2. *If X_+ is independent of X_i given $\mathbf{Pa}_i^G$ in p, then* $\Delta(G, G_+, \mathcal{D}) < 0$.

The first clause requires that edges that represent valid dependencies receive a positive gain. The second clause requires that redundant edges that do no represent a valid dependency receive a negative gain. An upgrade method is relatively locally consistent if the upgraded i.i.d. gain function (which may be derived from a single score) is locally consistent for relational data, whenever it is locally consistent for i.i.d. data.

Theorem 1. *The normalized gain function (Section 3.2.3) is relatively locally consistent. None of the relational model scores (Section 3.3) are relatively locally consistent.*

Proof. Case 1: The additional edge $X_+ \rightarrow X_i$ adds no population variables. For such edges, the count-count upgrade score has the same mathematical form as the original i.i.d. score. Therefore the arguments of [3] apply, and the score is locally consistent. Since by (*), for such edges, the normalized gain is equivalent to the count-count upgrade score, the normalized gain is locally consistent as well.

Case 2: The additional edge $X_+ \rightarrow X_i$ adds a population variable. For concreteness, assume that the edge is of the form $g(\mathbb{A}, \mathbb{B}) \rightarrow f(\mathbb{A})$, so the added sample size is $N[\mathbb{B}; \mathcal{D}]$. Consider a transformed database $\mathcal{D}'$ where $f(\mathbb{A})$ is replaced by $f'(\mathbb{A}, \mathbb{B})$, with an inert second argument: $f'(a, b) = f(a)$. Then in $\mathcal{D}'$, each satisfying assignment in $\mathcal{D}$ for the family of $f(\mathbb{A})$ is multiplied by $N[\mathbb{B}; \mathcal{D}]$. Therefore the sufficient statistics are related as follows:

$$\mathrm{n}_{ijk}^{G}(\mathcal{D}') = N[\mathbb{B}; \mathcal{D}] \times \mathrm{n}_{ijk}^{G}(\mathcal{D})$$

and

$$\mathrm{n}_{ijk}^{G_+}(\mathcal{D}') = \mathrm{n}_{ijk}^{G_+}(\mathcal{D}).$$

These equalities imply that the normalized gain for $\mathcal{D}$ is the same as that for $\mathcal{D}'$. Now in $\mathcal{D}'$, the new edge adds no population variables, and so by Case 1, the normalized gain is locally consistent for $\mathcal{D}'$. Therefore it is locally consistent for $\mathcal{D}$. Hence in either case, the normalized gain is locally consistent.

For the non-consistency, we omit a formal proof due to space constraints. Instead, we give the intuitive reason. By under(over)fitting we mean producing an overly sparse (dense) DAG.

Count-Count Score S The count likelihood typically decreases for an edge that adds a population variable, even if the edge represents a true dependency. We expect to observe *underfitting*. [discuss the example table.]

Normalized-Count $\widetilde{S}$ The weight of the likelihood term does not increase with sample size, so an edge that represents a true dependency may not be added even in the sample size limit. We expect to observe *underfitting*.

Normalized-Normalized $\overline{S}$ The normalized penalty term for the expanded structure G_+ is down-weighted more than the normalized penalty term for the simpler structure G. So a redundant edge may be added even in the sample size limit. We expect to observe *overfitting.*

From the insights in this section, we derive a number of empirical hypotheses about the relationships between the upgrade methods.

3.4.2 Hypotheses

We assess these hypotheses in experiments that are described in the following section.

1. A normalized-count score selects smaller graphs than the other upgrade methods.
2. A normalized-normalized score selects larger graphs than the other upgrade methods.
3. A count-count score does not select edges that add population variables.
4. The difference between upgrade methods concern only edges that add population variables.

Chapter 4

Empirical Results

We describe the system and the datasets we used. Code was written in 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 data is available on-line (pointer removed for blind review).

4.1 Datasets

We used seven benchmark real-world databases from the CTU Prague Relational Learning Repository [21]. For detailed descriptions and the sources of the databases, please see references [36, 30]. Table 4.1 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[1] with data from the Internet Movie Database (IMDb)[2] following [27].

[1] www.grouplens.org, 1M version
[2] www.IMDb.com, July 2013

Dataset	**#Relationship Tables/ Total**	**#Tuples**	**#Attributes**
University	2	171	12
Movielens	1 / 3	1,010,051	7
Mutagenesis	2 / 4	14,540	11
Financial	3 / 7	225,932	15
Hepatitis	3 / 7	12,927	19
IMDb	3 / 7	1,354,134	17

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

4.2 Methods Compared

The previously existing learn-and-join method (LAJ) is the state of the art for Bayes net learning in relational databases [36, 30]. We used the LAJ implementation provided by its creators. The LAJ method conducts a search through the lattice of relational paths. At each lattice point, an i.i.d. Bayesian network learner is applied, and learned edges are propagated from shorter paths to longer paths. We reconfigured the LAJ algorithm by changing the `score` class, for each of the 2 gains defined in Table 3.3 and each of the 6 scores defined in Table 3.5.

4.3 Results

For each learned graph G, we use maximum likelihood estimates to obtain a Bayesian network B to be evaluated. To measure how close the joint distribution represented by a learned BN is to the database distribution, we employ the standard Kulback-Leibler divergence metric [5] (KLD). The KLD is given by

$$P_{\mathcal{D}}||P_B = \sum_{\boldsymbol{X}=\boldsymbol{x}} P_{\mathcal{D}}(\boldsymbol{X}=\boldsymbol{x}) \ln \frac{P_{\mathcal{D}}(\boldsymbol{X}=\boldsymbol{x})}{P_B(\boldsymbol{X}=\boldsymbol{x})}$$

where the summation ranges over all assignments to node values. Figure 4.1 shows the KL divergence and number of parameters for upgrading the AIC resp. BIC scores. We discuss several findings that confirm the hypotheses of Section 3.4.

The normalized-count and normalized-normalized upgrade methods are inadequate

Hypothesis 1 is fully confirmed: The normalized-count scores select very sparse structures (parameter numbers from 10 to 100). The only edges in the selected structures are deterministic relationships between a relationship indicator and an associated descriptive attribute. An example would be the edge $HasRated(Movie, User) \rightarrow Rating(Movie, User)$. (The rating value is n/a (for "not applicable") if and only if $HasRated(Movie, User)$ is false.)

Hypothesis 2 is also fully confirmed: The normalized-normalized scores select very dense structures (parameter numbers from 100 to 1M). The many edges found by a normalized-normalized score lead to almost no improvement in KLD compared to the normalized gain function. Given their theoretical shortcomings (lack of balance and consistency), we conclude that these two score types are clearly inadequate and focus on comparing the count-count and normalized gain upgrade methods.

Large Sample Size Comparison for the Count-Count and Normalized-Gain Methods

Recall that the count-count score uses the grounding count for the likelihood function and the number of parameters for the penalty term. The KLD of both upgrade methods is very close on all datasets. On MovieLens and Financial, the AIC count-count score uses more parameters than the

normalized gain function, and on IMDb it uses fewer. On Hepatitis, the BIC count-count score uses significantly fewer parameters.

Figures 4.2 and 4.3 consider the structures learned with increasing sample size. The figures are constructed as follows. We duplicate entities by a factor of $m = 1, 5, 10, 20$. For example on IMDb, there would be 20 copies of BraveHeart with the same attributes and relationships. This can be viewed as a sampling-with-replacement variation of the design in [11, 37], where the input database is also treated as representing a complete population. In terms of the model comparison scores, the local sample sizes are multiplied by a uniform *magnification factor*, which we denote as $\boldsymbol{N} \times m$.

As the magnification factor m increases, the sample size tends towards infinity. The measurements reported therefore also illustrate the consistency analysis of Section 3.4. To simplify the presentation, we focus on the BIC score. Unlike the AIC score [41, Sec.6.7], the BIC score is consistent for i.i.d. data [4]. Therefore Theorem 1 entails that it is also consistent for multi-relational data when upgraded with the normalized gain method. Figure 4.3 shows the convergence to the database distribution (KLD = 0) on our benchmark databases for the normalized-gain BIC upgrade. The University dataset requires a higher magnification factor for convergence as the original dataset is much smaller than the others. The count-count score converges to the database distribution at a similar rate (Figure 4.2). The exception is the Financial database where the count-count score stops improving at KLD = 0.08, compared to KLD = 0.04 for the normalized gain with sample magnification 20. While the KLD values are similar for the two upgrade methods (except for Financial), there are differences in the structures learned that we discuss next.

Edges Selected by the Normalized Gain Method but not by the Count-Count scores Figure 4.4 fully confirms Hypothesis 3: *the count-count BIC score selects no edges that add population variables.* Together, Figures 4.4 and 4.1 confirm Hypothesis 4: whenever the normalized gain BIC upgrade find no such edges, it discovers the same structure as the count-count BIC upgrade (databases University and Mutagenesis). The fact that the count-count BIC score does not select dependencies represented by edges that add population variables, even with increasing sample size illustrates its nonconsistency.

To understand the additional edges selected by the normalized gain BIC further, we assessed them in the light of domain knowledge. Figure 4.5 shows some of the plausible edges discovered by the Normalized BIC gain score with a magnification of 20. University: The grade of a student in a random course predicts her intelligence. MovieLens: The rating that a user assigns to a random movie predicts his or her gender and age. IMDb: If an actor appears in a movie whose language is English, she is more likely to be rated as high quality. UW: The years of a person in the program are predicted by whether his advisor has a position. Financial: The frequency of transactions for an account is predicted by facts about loans associated with the account. Based on domain understanding, it is plausible that these dependencies reflect genuine associations in the domain. However, the count-count method is incapable of finding them.

Why Do Few Edges Add Population Variables? Figure 4.4 shows that the normalized gain function selects few edges that add population variables even in the sample size limit. The reason for this is that our datasets contain many unary attributes and relatively few binary functors that represent relationships. *If two unary attributes contain different population variables, they are marginally independent.* For example, the nodes $gender(Actor)$ and $genre(Movie)$ are marginally independent because the attributes of a randomly selected actor are independent of those of a randomly selected movie. The nodes may be conditionally dependent given that the actor appeared in the movie, that is, given $AppearsIn(Actor, Movie) = \mathrm{T}$. However, a BN score tends to penalize edges that connect marginally independent nodes. Instead, it favors a v-structure

$$gender(Actor) \rightarrow AppearsIn(Actor, Movie) \leftarrow genre(Movie)$$

for representing this (in)dependence pattern. No edge in this v-structure adds population variables. This observation shows another reason why adding population variables is helpful: once such an edge $X_+ \rightarrow X$ is added, unary attributes can become conditionally dependent given X_+. For example, once $Rating(User, Movie)$ is added as a parent of $gender(User)$, the relevance of $Drama(Movie)$ is assessed conditional on the fact that the user has rated the movie.

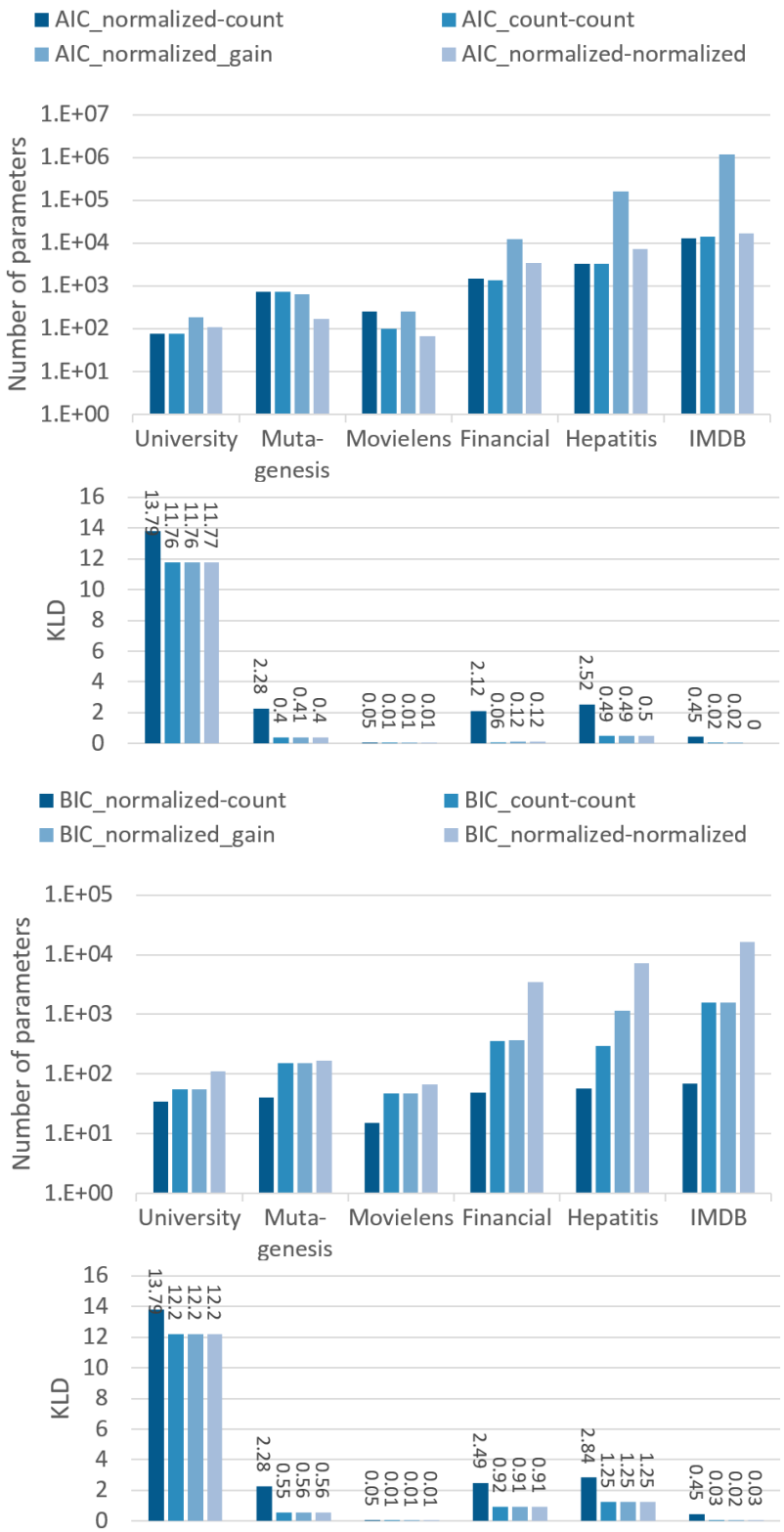

Figure 4.1: KLD with database distribution and Number of Parameters for different relational score upgrade methods. The number of parameters is shown on log-scale. Top: AIC upgrades. Bottom: BIC upgrades.

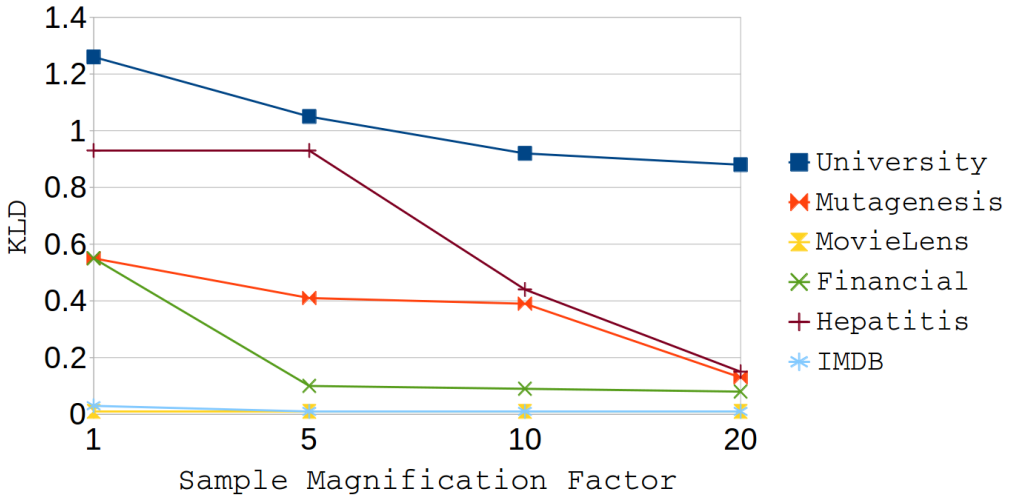

Figure 4.2: KLD for the BIC count-count score

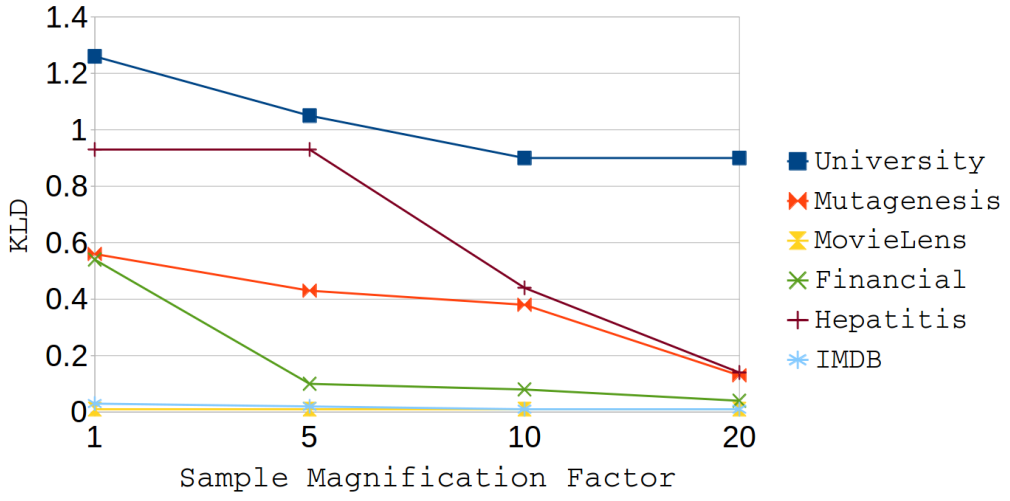

Figure 4.3: KLD for the BIC normalized gain

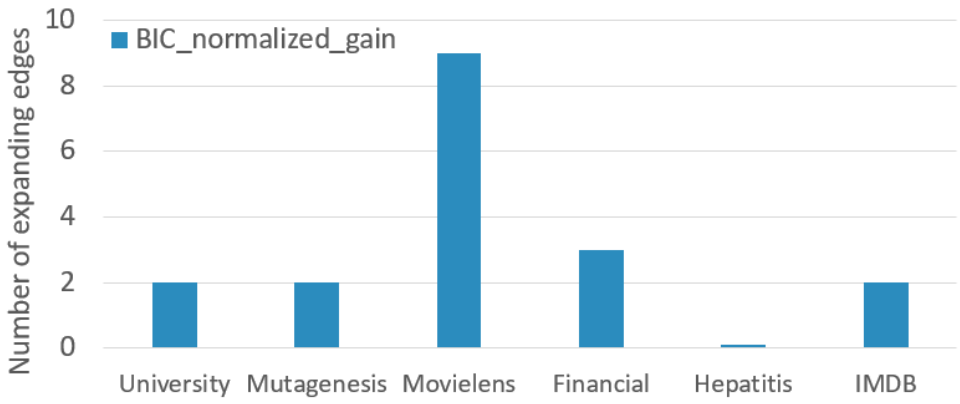

Figure 4.4: The number of edges that add population variables, for the normalized gain upgrade method. This number is 0 for the count-count method on every dataset. (Sample sizes magnified by a factor of 20).

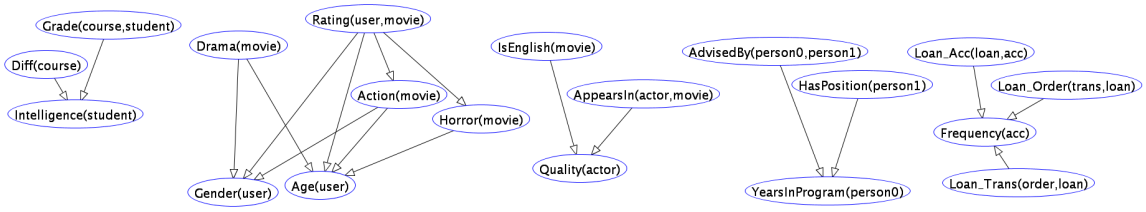

Figure 4.5: Examples of edges selected by the normalized gain function that introduce additional population variables. Left to right: University, MovieLens, IMDb, Mutagenesis, Financial.

Chapter 5

Conclusions

The experimental results support the normalized gain as the best upgrade method. The normalized-normalized and normalized-count upgrade methods are clearly inadequate. The count-count method is incapable of selecting edges that add population variables. We provided qualitative evidence that such edges can be informative. While the quantitative statistical evidence for their importance is mixed on our datasets, such a strong a priori bias against certain types of edges is clearly undesirable. In addition, the normalized gain is the only upgrade method in our set that has the important properties of balance and (relative) consistency.

We described a novel method for upgrading an i.i.d. Bayesian network score for multi-relational data. The normalized gain function measures the difference in data fit between two structures. Normalized gain functions preserve two key properties of i.i.d. BN scores: consistency and balance, meaning that the model's data likelihood and the model's complexity are measured on the same scale. A surprising negative result is that these properties cannot be achieved with log-linear likelihood scores that are a function of a single model only. Empirical results demonstrate a special strength of the normalized gain upgrade method: finding arcs whose source node contains population variables that are not included in the child node (e.g. $Rating(User, Movie) \rightarrow age(User)$). A promising avenue for future work is to extend our approach to other statistical-relational models that use complete instantiation likelihoods (see Section 3.1). It should be possible to develop a gain function for the log-linear likelihood of Markov Logic Networks. As for other Bayesian network relational models, Kersting and deRaedt suggest translating aggregation-based likelihoods into (the equivalent of) a FOB by adding complex aggregate terms as nodes [15]. Recent results on representing combining rules in a logistic regression model suggest that similar complex terms can be defined for combining rules [2]. Adding nodes that represent complex terms is a promising approach to unifying our framework with aggregate functions/combining rules. This has the potential to address both the question of consistency and the issue of cyclicity.

Generalizing i.i.d. model selection scores designed for i.i.d. data is an important yet complex topic for relational learning. The normalized gain is a novel upgrade method that preserves important theoretical properties of i.i.d. model selection scores, and shows good empirical performance.

Bibliography

[1] R. Bouckaert. *Bayesian belief networks: from construction to inference*. PhD thesis, U. Utrecht, 1995.

[2] David Buchman and David Poole. Representing aggregators in relational probabilistic models. In *AAAI*, pages 3489–3495, 2015.

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

[4] David Maxwell Chickering and Christopher Meek. Finding optimal Bayesian networks. In *UAI*, pages 94–102, 2002.

[5] L. de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *JMLR*, pages 2149–2187, 2006.

[6] Pedro Domingos and Matthew Richardson. Markov logic: A unifying framework for statistical relational learning. In *Introduction to Statistical Relational Learning* [10].

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

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

[9] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Benjamin Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning* [10], chapter 5, pages 129–173.

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

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

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

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

[14] D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In Getoor and Taskar [10].

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

[16] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *Machine Learning*, pages 1–45, 2014.

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

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

[19] 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.

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

[21] Jan Motl and Oliver Schulte. The CTU prague relational learning repository. *CoRR*, abs/1511.03086, 2015.

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

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

[24] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[25] 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.

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

[27] Veronika Peralta. Extraction and integration of MovieLens and IMDB data. Technical report, Laboratoire PRiSM, Universite de Versailles, 2007.

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

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

[30] Zhensong Qian, Oliver Schulte, and Yan Sun. Computing multi-relational sufficient statistics for large databases. In *CIKM*, pages 1249–1258, 2014.

[31] J. Ross Quinlan and R. Mike Cameron-Jones. Foil: A midterm report. In *ECML*, volume 667, pages 3–20. Springer, 1993.

[32] Stuart Russell. Unifying logic and probability. *Communications of the ACM*, 58(7):88–97, 2015.

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

[34] Yoshiki Sakai and Kenji Yamanishi. An nml-based model selection criterion for general relational data modeling. In *Big Data*, pages 421–429. IEEE, 2013.

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

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

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

[38] Cosma Rohilla Shalizi and Alessandro Rinaldo. Consistency under sampling of exponential random graph models. *Annals of statistics*, 41(2):508, 2013.

[39] Sameer Singh and Thore Graepel. Automated probabilistic modelling for relational data. In *CIKM*, 2013.

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

[41] David Williams. *Weighing the Odds*. Cambridge University Press, 2001.

[42] Rongjing Xiang and Jennifer Neville. Relational learning with one network: An asymptotic analysis. In *Artificial Intelligence and Statistics*, pages 779–788, 2011.

Appendix A

Global Caching

In this chapter we will see the code and contribution I had to cache the scores in a global environment. We were working on our specific implementation of the model selection for bayesian networks code which was a derivation of tetrad project [1]. BayesBase[2] is the implementation we were used for model selection; that is based on the search+score method of GES [4]. One of the problem I found in the efficiency of the code was that there is no storing space to store the score we got for a specific structure. Because GES searches through the lattice of different bayesian structures and it is possible to have a look again on a same structure. To fix the problem, we implemented a global caching system:

```
public class RunBB {
        ...
        private static Map<Node, Map<Set<Node>, Double[]>>
           globalScoreHash =  new WeakHashMap<Node,
           Map<Set<Node>, Double[]>>();
        ...
}
```

which stores an array of Double values for each node and its parents. We can consider it as a mapping system between the structure and an array of Double values. At first the only needed value for each structure was the score of the structure based on AIC or BIC scores. After a which when we find out that "There is no single score for each structure", the main sentence of the whole thesis, we find out that we have to re-implement some parts of the GES algorithm having not only the actual value of the score, but also having the log-likelihood value and the penalty term for the structure. Therefore, I implemented a 3-value array for the score, penalty, and number of groundings and change the part of the code it was trying to compare two structure to find the better one such a way to use these three values:

```
public final class Ges3 implements GraphSearch, GraphScorer {
    ...
    private Double[] scoreGraphChange(Node y, Set<Node>
       parents1,Set<Node> parents2, Map<Node, Map<Set<Node>,
       Double[]>> globalScoreHash) {
```

[1] http://www.phil.cmu.edu/tetrad/publications.html
[2] http://www.cs.sfu.ca/~oschulte/BayesBase/BayesBase.html

```
        ...
        if(RunBB.currScoreName == "LL" && RunBB.currScoreType ==
           "_gain"){
                bumps[0] = spg1[0]*spg1[2] - spg2[0]*spg2[2];
        }else if(RunBB.currScoreType.endsWith("_gain"))
                bumps[0] = (spg1[0] + spg1[1]/spg1[2] -
                   spg1[1]/spg2[2])*spg1[2] - spg2[0]*spg1[2];
                   // bumps[0] is the gain diference
        else
                bumps[0] = spg1[0]*spg1[2] - spg2[0]*spg1[2];
        bumps[1] = spg1[0]*spg1[2] - spg2[0]*spg1[2];  //
           bums[1] is the score diference (using in fes -> score
           += bump)
        return bumps;
    }
    ...
}
```

Based on the score caching system and the new score comparison criteria, I needed to go through all the code and change every single place that we were computing or using structure scores; all the changes are available in the actual code, I do not want to go deeply inside the code.

Appendix B

Scores class

This chapter contains the scores implementation which make a Scores class.

```
package edu.cmu.tetrad.search;


import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import main.Config;
import main.RunBB;
import edu.cmu.tetrad.util.ProbUtils;
import edu.cmu.tetrad.data.DataSet;
import edu.cmu.tetrad.data.DiscreteVariable;
import edu.cmu.tetrad.graph.Node;

/**
 * @author Sajjad. I implemented all the needed scores using
    tetrad code and Diljot global implementation.
 */
public class Scores implements LocalDiscreteScore {
        private DataSet dataSet;
        private double samplePrior = 10;
        private double structurePrior = 1.0;
        private String scoreName = "AIC";
        private String scoreType = "c";
        private String databaseName;
```

```
public Scores(DataSet dataSet, double samplePrior,
   double structurePrior, String scoreName, String
   scoreType) {
        if (dataSet == null) {
                throw new NullPointerException();
        }
        this.dataSet = dataSet;
        this.samplePrior = samplePrior;
        this.structurePrior = structurePrior;
        this.scoreName = scoreName;
        this.scoreType = scoreType;
        Config conf = new Config();
        databaseName = conf.getProperty("dbname");
}

public Double[] localScore(int i, int parents[], Node y,
                Set<Node> parentNodes,
                Map<Node, Map<Set<Node>, Double[]>>
                   globalScoreHash) {

        Double[] oldscore = null;
        if (globalScoreHash.containsKey(y)) {
                if
                   (globalScoreHash.get(y).containsKey(parentNodes))
                   {
                        oldscore =
                           globalScoreHash.get(y).get(parentNodes);
                }
        }

        if (oldscore != null &&
           !Double.isNaN(oldscore[0])) {

                // =================Writing to the
                   file.======================
                // We just use this file to compare the
                   scores later after execution
                // is complete.
                // The following code will write the
                   contents of the hit to file
                try {
                        File file = new
                           File("Hash-Hits");

                        if (!file.exists()) {
                                file.createNewFile();
                        }
```

```
                    FileWriter fileWriter = new
                       FileWriter(file.getName(),
                       true);
                    BufferedWriter bufferWriter =
                       new
                       BufferedWriter(fileWriter);
                    bufferWriter.write("Node:" + y +
                       "\nParents: "
                                    + parentNodes +
                                       "\nScore:" +
                                       oldscore);
                    bufferWriter.write("\n_____________________________
                    bufferWriter.close();
            } catch (Exception e) {
                    System.out.println("Error
                       Writing");
            }
            // ============================Writing to
               file
            // complete===============

            return oldscore;
    }

    // Number of categories for i.
    int r = numCategories(i);

    // Numbers of categories of parents.
    int dims[] = new int[parents.length];

    for (int p = 0; p < parents.length; p++) {
            dims[p] = numCategories(parents[p]);
    }

    // Number of parent states.
    int q = 1;
    for (int p = 0; p < parents.length; p++) {
            q *= dims[p];
    }

    // Conditional cell counts of data for i given
       parents(i).
    long n_ijk[][] = new long[q][r];
    long n_ij[] = new long[q];
    long n_ik[] = new long[r];
    long n_ijk1[][] = new long[q][r];
```

```
int values[] = new int[parents.length];
System.out.println("SampleSize for computing
   BDeu Score: "+ sampleSize());
for (int n = 0; n < sampleSize(); n++) {
        for (int p = 0; p < parents.length; p++)
           {
                int parentValue =
                   dataSet().getInt(n,
                   parents[p]);

                if (parentValue == -99) {
                      throw new
                         IllegalStateException("Please
                         remove or impute "+
                         "missing values.");
                }
                values[p] = parentValue;
        }

        int childValue = dataSet().getInt(n, i);

        if (childValue == -99) {
                throw new
                   IllegalStateException("Please
                   remove or impute missing " +
                   "values (record "+ n + "
                   column " + i + ")");
        }
        n_ijk[getRowIndex(dims,
           values)][childValue] +=
           dataSet().getMultiplier(n)*RunBB.resampelingFactor;
           // instead of the above loop @Saj
}


//================* @Sajjad (06Jun15)
   *==================
File f;
FileWriter fw;
BufferedWriter bw;
File f2;
FileWriter fw2;
BufferedWriter bw2;
Double spg[] = new Double[3];
Double score = null;
Double penalty = null;
Double grounding = null;
```

```
        try {
                f = new File(databaseName +
                   "_History.txt");
                if(!f.exists())
                        f.createNewFile();
                fw = new FileWriter(f.getName(), true);
                bw = new BufferedWriter(fw);

                f2 = new File(databaseName +
                   "/StructuresInfo.csv");
                if(!f2.exists())
                        f2.createNewFile();
                fw2 = new FileWriter(f2, true);
                bw2 = new BufferedWriter(fw2);

                double N_i = 0.0; //Used for penalty
                   term.
                for (int j = 0; j < q; j++) {
                        for (int k = 0; k < r; k++) {
                                n_ij[j] += n_ijk[j][k];
                        }
                        N_i += n_ij[j]; ///// N_i equals
                           to number of groundings.
                }
                for (int k = 0; k < r; k++){
                        for(int j=0; j<q; j++){
                                n_ik[k] += n_ijk[j][k];
                        }
                }

                // Based on "Scoring functions for
                   learning Bayesian networks" slides
        score = 0.0;
        grounding = 1.0;
        double wpll = 0.0;
        double w = 0.0;
        double prior = 0.0;
        double conditional = 0.0;
        double alpha = 0.01;
        double paramNum = (double)(r-1)*(double)q;

    // Based on "Scoring functions for learning
       Bayesian networks" slides
        double loglikelihood = 0.0;
        for (int j = 0; j < q; j++) {
```

```
                        if (n_ij[j] != 0){ //if parent count is
                           not 0, find corresponding likeilhood
                           term
                            for (int k = 0; k < r; k++) {
                                if (n_ijk[j][k] != 0){ //if
                                   parent-child count is not 0,
                                   find corresponding likeilhood
                                   term
                                        double theta_ijk =
                                           (double)n_ijk[j][k]/(double)n_ij[j];
                                        loglikelihood +=
                                           n_ijk[j][k] *
                                           log2(theta_ijk); //
                                           LL(B|T) = SS(N_ijk *
                                           log(N_ijk/N_ij)) ||
                                           This is equal to Lc
                                           in the paper
//                                      bw.write("\t" +
   loglikelihood + "\n");
                                }
                            }
                        }
                }


                grounding = (double)N_i;
                boolean sw = false;
                        if(grounding > 2200.0){
//                              grounding = 2280.0;
                                sw = true;
                        }

                switch(scoreName.toUpperCase()){
                case "AIC":
                        if(scoreType == "_count"){ //AIC_count
                                score = loglikelihood - paramNum;
                        penalty = paramNum;
                        }else if
                           (scoreType.startsWith("_normalized")){
                           //AIC_normalized
                                score = loglikelihood/N_i -
                                   paramNum/N_i;
                                penalty = paramNum;
                        }else if (scoreType == "_weighted"){
                           //AIC_weighted
                                score = loglikelihood/N_i -
                                   paramNum;
```

```
                penalty = paramNum;
        }
        break;
case "BIC":
        if(scoreType == "_count"){ //BIC_count
                score = loglikelihood -
                   0.5*log2(N_i)*paramNum;
        penalty = 0.5*log2(N_i)*paramNum;
        }else if
           (scoreType.startsWith("_normalized")){
           //BIC_normalized
                score = loglikelihood/N_i -
                   0.5*log2(N_i)*paramNum/N_i;
                penalty = 0.5*log2(N_i)*paramNum;
        }else if (scoreType == "_weighted"){
           //BIC_weighted
                score = loglikelihood/N_i -
                   0.5*log2(N_i)*paramNum;
                penalty = 0.5*log2(N_i)*paramNum;
        }
        break;
case "LL":
        if(scoreType == "_gain"){
                score = loglikelihood/N_i;
                penalty = 0.0;
        }
        break;
}

int alaki = 0;
if(node2string(y).equals("teachingability(prof0)")
   &&
   parents2string(parentNodes).equals("popularity(prof0)"))
        alaki = 10;

bw.write("Structure:\n\tNode: " + y +
   "\n\tParents: "+ parentNodes + "\n");
bw.write("# groundings: " + grounding + ",
   Penalty: " + penalty + ", r (node states): "
   + r + ", q (parents states): " + q + ",
   paramNum: " + paramNum + "\n");
bw.write(">>>>>>> " + scoreName + "_" +
   scoreType + ": " + score +
   "\n============================\n");
bw2.write(scoreName+scoreType +","+
   node2string(y) +","+
   parents2string(parentNodes) +","+ score +","+
```

```
   loglikelihood +","+ grounding +","+ penalty
   +","+ r +","+ q +","+ paramNum +"\n");
bw.close();
bw2.close();

} catch (Exception e) {
        System.out.println("Error Writing");
}
//================**********************************************

// localScoreCache.add(i, parents, score);
if (globalScoreHash.containsKey(y)) {
        globalScoreHash.get(y).put(parentNodes,
           new
           Double[]{score,penalty,grounding});
} else {
        globalScoreHash.put(y, new
           HashMap<Set<Node>, Double[]>());
        globalScoreHash.get(y).put(parentNodes,
           new
           Double[]{score,penalty,grounding});
}

// ===============File =================
// /Another file keeps track of the times when
   we don't get a hit in the
// cache.
try {
        File file = new File("NoHit");

        if (!file.exists())
                file.createNewFile();

        FileWriter fileWriter = new
           FileWriter(file.getName(), true);
        BufferedWriter bufferWriter = new
           BufferedWriter(fileWriter);
        bufferWriter.write("Node:" + y +
           "\nParents: "
                        + parentNodes +
                           "\nScore:" + score);
        bufferWriter
                        .write("\n_____________________________
        bufferWriter.close();
} catch (Exception e) {
        System.out.println("Error Writing");
}
```

```
            // ====================================End
               writing===============
            spg[0] = score;
            spg[1] = penalty;
            spg[2] = grounding;
            return spg;
    }

    public DataSet getDataSet() {
            return dataSet;
    }

    private double log2(double x) {
    return Math.log(x) / Math.log(2);
}

    private int getRowIndex(int[] dim, int[] values) {
            int rowIndex = 0;
            for (int i = 0; i < dim.length; i++) {
                    rowIndex *= dim[i];
                    rowIndex += values[i];
            }
            return rowIndex;
    }

    private int sampleSize() {
            return dataSet().getNumRows();
    }

    private int numCategories(int i) {
            return ((DiscreteVariable)
               dataSet().getVariable(i)).getNumCategories();
    }

    private DataSet dataSet() {
            return dataSet;
    }

    public double getStructurePrior() {
            return structurePrior;
    }

    public double getSamplePrior() {
            return samplePrior;
    }

    public void setStructurePrior(double structurePrior) {
```

```
                this.structurePrior = structurePrior;
        }

        public void setSamplePrior(double samplePrior) {
                this.samplePrior = samplePrior;
        }

        @Override
        public Double[] localScore(int i, int[] parents) {
                // TODO Auto-generated method stub
                return new Double[3];
        }

        public String parents2string(Set<Node> parentNodes){
                String out = "";
                for(Node n: parentNodes){
                        out += node2string(n) + " ";
                }
                return out.trim();
        }

        public String node2string(Node node){
                String out = node.getName();
                out = out.replace(',', '-');
                return out;
        }
}
```

Appendix C

Sort-Merge implementation

Sort-Merge implementation for computing CP tables makes the code run so much faster than it was before. For some big databases like IMDB_MovieLens computing the complete graph scores was a hassle; We could not even try that. Once I ran it for around 4 days and it could not compute the CP table even for the first node. With the new implementation, we can easily run the complete graph computation code for IMDB_MovieLens and it takes about a day to compute all the CP tables. This part used to be done by SQL join query on MySQL, now it is done manually by implementing Sort-Merge. Here is the code:

```
public class CP {
    ...
    public static void updateParentSum(String table1, String
       table2, String table3, String node, String order,
       Connection conn) throws SQLException, IOException {
                File ftemp = new File("sort_merge.csv");
                if (ftemp.exists())
                        ftemp.delete();
                File file = new File("sort_merge.csv");
                BufferedWriter output = new BufferedWriter(new
                   FileWriter(file));

                Statement st1 = conn.createStatement();
                Statement st2 = conn.createStatement();

                long time1 = System.currentTimeMillis();
                ResultSet rst1 = st1.executeQuery("select
                   distinct mult, " + node + "," + order + "
                   from " + table1 + " order by " + order + "
                   ;");
                ResultSet rst2 = st2.executeQuery("select
                   distinct mult, " + order + ", ParentSum from
                   " + table2 + " order by " + order + " ;");
                long time2 = System.currentTimeMillis();
```

```
// finding the no. of rows in each table
int size1 = 0, size2 = 0;
while (rst1.next())     size1++;
while (rst2.next())     size2++;

// finding the no of columns in a table
ResultSetMetaData rsmd = (ResultSetMetaData)
   rst1.getMetaData();
int no_of_colmns = rsmd.getColumnCount();

int i = 1;
int j = 1;// index variables for both tables
rst1.absolute(1);
rst2.absolute(1);
long time3 = System.currentTimeMillis();
// merging starting here
while (i <= size1 && j <= size2) {
        long val1 = 0, val2 = 0;
        for (int k = 3; k <= no_of_colmns; k++) {
                try {
                        val1 =
                           Long.parseLong(rst1.getString(k));
                        val2 =
                           Long.parseLong(rst2.getString(k-1));
                } catch
                   (java.lang.NumberFormatException
                   e) {
                } finally {
                        if
                           (rst1.getString(k).compareTo(rst2.get
                           - 1)) > 0) {
                                val1 = 1;
                                val2 = 0;
                        } else if
                           (rst1.getString(k).compareTo(rst2.get
                           - 1)) < 0) {
                                val1 = 0;
                                val2 = 1;
                        }
                }

                if (val1 < val2) {
                        String quer =
                           rst1.getString(1);
                        for (int c = 2; c <=
                           no_of_colmns; c++) {
```

```
                                        quer = quer +
                                           "$" +
                                           rst1.getString(c);
                                }
                                quer += "$" +
                                   rst2.getString(no_of_colmns);
                                output.write((quer) +
                                   "\n");
                                i++;
                                break;
                        }

                        else if (val1 > val2) {
                                j++;
                                break;
                        }
                }
                if (val1 == val2) {
                        String query = "" +
                           Long.parseLong(rst1.getString(1));
                        for (int c = 2; c <=
                           no_of_colmns; c++) {
                                query = query + "$" +
                                   rst1.getString(c);
                        }
                        query += "$" +
                           rst2.getString(no_of_colmns);
                        output.write(query + "\n");
                        i++;
                        //j++; COMMENTED BY SAJ
                }
                rst1.absolute(i);
                rst2.absolute(j);
        }

        if (i > 1)
                rst1.absolute(i - 1);
        else
                rst1.beforeFirst();
        while (rst1.next()) {
                String query = rst1.getString(1);
                for (int c = 2; c <= no_of_colmns; c++) {
                        query = query + "$" +
                           rst1.getString(c);
                }
                if(rst2.next())
```

```
                        query += "$" +
                            rst2.getString(no_of_colmns);
                    output.write((query) + "\n");
                }
                output.close();
                long time4 = System.currentTimeMillis();
                // System.out.print("\t insert
                    time:"+(time4-time3));

                st2.execute("drop table if exists " + table3 +
                    "; ");
                st2.execute("create table " + table3 + " like "
                    + table1 + " ;");
                st2.execute("LOAD DATA LOCAL INFILE
                    'sort_merge.csv' INTO TABLE " + table3
                            + " FIELDS TERMINATED BY '$'
                                LINES TERMINATED BY '\\n'
                                ;");

                st2.execute("Alter table " + table3 + " add
                    `local_mult` decimal(42,0);" );


                rst1.close();
                rst2.close();
                st1.close();
                st2.close();

                long time5 = System.currentTimeMillis();
                // System.out.print("\t export csv file to
                    sql:"+(time5-time4));
                System.out.println("\ntotal time: " + (time5 -
                    time1) + "\n");

                if (ftemp.exists())
                        ftemp.delete();
    }
    ...
}
```