Learning Class-Level Bayes Nets for Relational Data

Oliver Schulte, Hassan Khosravi, Bahareh Bina, Flavia Moser, Martin Ester
{oschulte, hkhosrav, bb18,fmoser, ester}@cs.sfu.ca
School of Computing Science
Simon Fraser University
Vancouver-Burnaby, B.C., Canada

Abstract

Many databases store data in relational format, with different types of entities and information about links between the entities. The field of statistical-relational learning (SRL) has developed a number of new statistical models for such data. In this paper we focus on learning class-level or first-order dependencies, which model the general database statistics over attributes of linked objects and links (e.g., the percentage of A grades given in computer science classes). Class-level statistical relationships are important in themselves, and they support applications like policy making, strategic planning, and query optimization. Most current SRL methods find class-level dependencies, but their main task is to support instance-level predictions about the attributes or links of specific entities. We focus only on class-level prediction, and describe algorithms for learning class-level models that are orders of magnitude faster for this task. Our algorithms learn Bayes nets with relational structure, leveraging the efficiency of single-table nonrelational Bayes net learners. An evaluation of our methods on three data sets shows that they are computationally feasible for realistic table sizes, and that the learned structures represent the statistical information in the databases well. After learning compiles the database statistics into a Bayes net, querying these statistics via Bayes net inference is faster than with SQL queries, and does not depend on the size of the database.

1 Introduction

Many real-world applications store data in relational format, with different tables for entities and their links. Standard machine learning techniques are applied to data stored in a single table, that is, in nonrelational, propositional or "flat" format [18]. The field of statistical-relational learning (SRL) aims to extend machine learning algorithms to relational data [8, 3]. One of the major machine learning tasks is to use data to build a *generative statistical model* for the variables in an application domain [8]. In the single-table learning setting, the goal is often to represent predictive dependencies between the attributes of a single individual (e.g., between the intelligence and ranking of a student). In the SRL setting, the goal is often to represent, in addition, dependencies between attributes of different individuals that are related or linked to each other (e.g., between the intelligence of a student and the difficulty of a course given that the student is registered in the course).

Task Description: Modelling Class-Level Dependencies Many SRL models distinguish two different levels, a class or type dependency model G_M and an instance dependency model G_I [7, 14, 19]. In a graphical SRL model, the nodes in the instance dependency model represent attributes of individuals or relationships. The nodes in the class dependency model correspond to attributes of the tables. The use of the term "class" here is unrelated to classification; [16] views it as analogous to the concept of class in object-oriented programming. An example of a class-level probabilistic dependency is "among students with high intelligence, the rate of GPA = 4.0 is 80%". An instance-level prediction would be "given that Jack is highly intelligent, the probability that his GPA is 4.0 is 80%". Thus class-level dependencies are concerned with the *rates* at which events occur, or at which properties hold within a class, whereas instance-level dependencies are concerned with specific events or the properties of specific entities [11, 1]. In terms of predicate logic, the class-level model features terms that involve first-order variables (e.g., $intelligence(S)$, where S is a variable ranging over a domain like $Students$), whereas the instance-level graph features terms that involve constants (e.g., $intelligence(jack)$ where $jack$ is a constant that denotes a particular student) [3, 14, 5].

Typically, SRL systems instantiate a class-level model G_M with the specific entities, their attributes and their relationships in a given database to obtain an instance dependency graph G_I. An important purpose of the instance graph is to support predictions about the attributes of individual entities. A key issue for making such predictions is the *combining problem*: how to combine information from different related entities to predict a property of a target entity. Current SRL model construction algorithms learn class-level dependencies and instance-level predictions at the same time.

What is new about our approach is that we focus on class-level variables only rather than making predictions about individual entities. We apply Bayes net technology to design new algorithms especially for learning class-level dependencies. In experiments our algorithms run at two orders of magnitude faster than a benchmark SRL method. Our models thus trade-off tractability of learning with the ability to answer queries about individual entities.

Applications. Examples of applications that provide motivation for the class-level statistical models include the following.

1. *Policy making and strategic planning.* A university administrator may wish to know which program characteristics attract high-ranking students in general, rather than predict the rank of a specific student in a specific program.

2. *Query optimization* is one of the applications of SRL where a statistical model predicts a probability for given table join conditions that can be used to infer the size of the join result [9]. Estimating join sizes is a key problem for database query optimization. In queries that involve several tables being joined together, the ideal scenario is to have smaller intermediate joins [17]. A class-level statistical model may be used for estimating frequency counts in the database to select smaller joins, and so to optimize the speed of answering queries by taking more efficient intermediate steps. For example, suppose we wish to predict the size of the join of a $\mathit{Student}$ table with a $\mathit{Registered}$ table that records which students are registered in which courses, where the selection condition is that the student have high intelligence. In a logical query language like the domain relational calculus [26], this join would be expressed by the conjunctive formula $\mathit{Registered}(S, C), \mathit{intelligence}(S) = \mathit{high}$. A query to a Join Bayes Net can be used to estimate the frequency with which this conjunction is true in the database, which immediately translates into an estimate of the size of the join that corresponds to the conjunction. The join conditions often do not involve specific individuals.

3. *Private Data.* In some domains, information about individuals is protected due to confidentiality concerns. For example, [6] analyzes a database of police crime records. The database is anonymized and it would be unethical for the data mining researcher to try and predict which crimes have been committed by which individuals. However, it is appropriate and important to look for general risk factors associated with crimes, for example spatial patterns [6]. Under the heading of privacy-preserving data mining, researchers have devoted much attention to the problem of discovering class-level patterns without compromising sensitive information about individuals [27].

Approach. We apply Bayes nets (BNs) to model class-level dependencies between attributes that appear in separate tables. Bayes nets [20] have been one of the most widely studied and applied generative model classes. A BN is a directed acyclic graph whose nodes represent random variables and whose edges represent direct statistical associations. Our class-level Bayes nets contain nodes that correspond to the descriptive attributes of the database tables, plus Boolean nodes that indicate the presence of a relationship; we refer to these as Join Bayes nets (JBNs). To apply machine learning algorithms to learn the structure of a Join Bayes Net from a database, we need to define an *empirical database distribution* over values for the class-level nodes that is based on the frequencies of events in the database. In a logical setting, the question is how to assign database frequencies for a conjunction of atomic statements such as $\mathit{intelligence}(S) = \mathit{high}$, $\mathit{Registered}(S, C)$, $\mathit{difficulty}(C) = \mathit{high}$. We follow the definition that was established in fundamental AI research concerning the combination of logic and statistics, especially the classic work of Halpern and Bacchus [11, 1]. This research generalized the concept of single-table frequencies to the relational domain: the frequency of a first-order formula in a relational database is the number of instantiations of the variables in the formula that satisfy the formula in the database, divided by the number of all possible instantiations. In the example above, the instantiation frequency would be the number of student-course pairs where the student is highly intelligent, the course is highly difficult, and the student is registered in the course, divided by all possible student-course pairs. In terms of table joins, the instantiation frequency is the number of tuples in the join that corresponds to the conjunction, divided by the maximum size of the join.

Our learn-and-join algorithm aims to find a model of the database distribution. It upgrades a single table BN learner, which can be chosen by the user, to carry out relational learning by decomposing the learning problem for the entire database into learning problems for smaller tables. The basic idea is to apply the BN learner repeatedly to tables and join tables from the database, and merge the resulting graphs into a single graphical model for the entire database. This algorithm treats the single-table BN learner as a module within the relational structure learning system. This means that only minimal work is required to build a statistical-relational JBN learner from a single-table BN learner.

The main algorithmic problem in parameter estimation (CP-table estimation) for Join Bayes nets is counting the number of satisfying instantiations (groundings) of a first-order formula in a database. We show how the recent virtual join algorithm of [15] can be applied to solve this problem efficiently. The virtual join algorithm is an efficient algorithm designed to compute database instantiation frequencies that involve non-existent links.

Our experiments provide evidence that the learn and join algorithm leverages the efficiency, scalability and reliability of single-table BN learning into efficiency, scalability and reliability for statistical-relational learning. We benchmark the computational performance of our algorithm against structure learning for Markov Logic Networks (MLNs), one of the most prominent statistical-relational formalisms [5]. In our experiments on small datasets, the run-time of the learn-and-join algorithm is about 20 times faster than the state-of-the art Alchemy program [5] for learning MLN structure. On medium-size datasets, such as the Financial database from the PKDD 1999 cup, Alchemy does not return a result given our system resources, whereas the learn-and-join algorithm produces a Join Bayes net model within less than 10 min. To evaluate the learned structures, we apply BN inference algorithms to predict relational frequencies and compare them with gold-standard frequencies computed via SQL queries. The learned Join Bayes nets predict the database frequencies very well. Our experiments on prediction take advantage of the fact that because JBNs use the standard Bayes net format, class-level queries can be answered by standard BN inference algorithms that are used "as is". Other SRL formalisms focus on instance-level inference, and do not support class-level inference, at least in their current implementations. Our datasets and code are available for ftp download from ftp://ftp.fas.sfu.ca/pub/cs/oschulte.

Paper Organization. As statistical-relational learning is a complex subject and a variety of approaches have been proposed, we review related work in some detail. A preliminary section introduces Bayes nets and predicate logic. We formally define Join Bayes nets and the instantiation frequency distribution that they model. The main part of the paper describes structure and parameter learning algorithms for Join Bayes nets. We evaluate the algorithms on one synthetic dataset and two public domain datasets (MovieLens and Financial), examining the learning runtimes and the predictive performance of the learned models.

Contributions. The main contributions of our work are as follows.

(1) A new type of first-order Bayes net for modelling the database distribution over attributes and relationships, which is defined by applying classic AI work on probability and logic.

(2) New efficient algorithms for learning the structure and parameters for the first-order Bayes net models.

(3) An evaluation of the algorithms on three relational databases, and an evaluation of the predictive performance of the learned structures.

2 Related Work

A preliminary version of our results appeared in the proceedings of the STRUCK and GKR workshops at IJCAI 2009. There is much AI research on the theoretical foundations of the database distribution, and on representing and reasoning with instantiation frequencies in an axiomatic framework based on theorem proving [11, 1]. Our approach utilizes graphical models for learning and inference with the database distribution rather than a logical calculus. For inference, our approach appears to be the first that utilizes standard BN inference algorithm to carry out probabilistic reasoning about database frequencies. For learning, our work appears to be the first to use Bayes nets to learn a statistical model of the database distribution.

Other SRL formalisms. Various formalisms have been proposed for combining logic with graphical models, such as Bayes Logic Networks (BLNs) [14], parametrized belief networks [21], object-oriented Bayes nets [16], Probabilistic Relational Model (PRMs) [7], and Markov Logic Networks [5]. We summarize the main points of comparison. General SRL overviews are provided in [14, 8, 3]. The main difference between JBNs and other SRL formalisms is their semantics. The semantics of JBNs is specified at the class level without reference to an instance-level model; a JBN models the database distribution defined by the instantiation frequencies.

Syntactically, Join Bayes Nets are very similar to parametrized belief networks and to BLNs. BLNs require the specification of combining rules, a standard concept in Bayes net design, to solve the combining problem. For instance, the class-level model may specify the probability that a student is highly intelligent given that she obtained an A in a difficult course. A set of grades thus translates into a multiset of probabilities, which the combining rule translates into a single probability. Essentially, a JBN is a BLN without combining rules.

The key difference between PRMs and BLNs is that PRMs use aggregation functions (like average), not combining rules [3]. [14, Sec.10.4.3] shows how aggregate functions can be added to BLNs; the same construction works for adding them to JBNs. Object-oriented Bayes nets use aggregation like PRMs, and have special constructs for capturing class hierarchies; otherwise their expressive power is similar to BLNs and JBNs.

Markov Logic Networks combine ideas from *undirected* graphical models with a logical representation. An MLN is a set of formulas, with a weight assigned to each. MLNs are like JBNs, and unlike BLNs and PRMs, in that an MLN is complete without specifying a combining rule or aggregation function. They are like BLNs and PRMs, and unlike JBNs, in that their semantics is specified in terms of an instance-level network whose nodes contain ground formulas (e.g., $age(jack) = 40$). Instance-level prediction is defined by using the log-linear form of Markov random fields [5, Sec.12.4].

Inference and Learning Because JBNs use the Bayes net format, class-level probabilistic queries that involve first-order variables and no constants can be answered by standard efficient Bayes nets inference algorithms, which are used "as is". This can be seen as an instance of lifted or first-order probabilistic inference, a topic that has received a good deal of attention recently [21, 4]. Other SRL formalisms focus on instance-level queries that involve constants only, and do not support class-level inference, at least in their current implementations.

The most common approach to SRL structure learning is to use the instance-level structure G_I to assign a likelihood to a given database $\mathcal{D}$. This likelihood is the counterpart to the likelihood of a sample given a statistical model in single table learning. Learning searches for a parametrized model G_M whose instance-level graph G_I assigns a maximum likelihood to the given database $\mathcal{D}$, where typically the likelihood is balanced with a complexity penalty to prevent overfitting [7, 14]. This approach is a conceptually elegant way to learn class-level dependencies and instance-level predictions at the same time. JBNs separate the task of finding generic class-level dependencies from the task of predicting the attributes of individuals. Two key advantages of learning directed graphical models such as Bayes nets at the class-level only include the following.

(1) Class-level learning avoids the problem that the instantiated model may contain cycles, even if the class-level model does not. For example, suppose that the class-level model indicates that if a student S_1 is friends with another student S_2, then their smoking habits are likely to be similar, so $Smokes(S_1)$ predicts $Smokes(S_2)$. Now if in the database we have a situation where a is friends with b, and b with c, and c with a, the instance level model would contain a cycle $Smokes(a) \rightarrow Smokes(b) \rightarrow Smokes(c) \rightarrow Smokes(a)$ [7, 19]. If learning is defined in terms of the instance model, cycles cause difficulties because concepts and algorithms for directed acyclic models no longer apply. In particular, the likelihood of a database that measures data fit is no longer defined. These difficulties have led researchers to conclude that "the acyclicity constraints of directed models severely limit their applicability to relational data" [19] (see also [5, 24]).

(2) Other directed SRL formalisms require extra structure to solve the combining problem for instance-level predictions, such as aggregation functions or combining rules. While the extra structure increases the representational power of the model, it also leads to considerable complications in learning.

3 Preliminaries

We combine the concepts of graphical models from machine learning, relational schemas from database theory, and conjunctions of literals from first-order logic. This section introduces notations and definitions for these background subjects.

3.1 Bayes nets A random variable is a pair $X = \langle dom(X), P_X rangle$ where $dom(X)$ is a set of possible values for X called the **domain** of X and $P_X : dom(X) \rightarrow [0, 1]$ is a probability distribution over these values. For simplicity we assume in this paper that all random variables have finite domains (i.e., discrete or categorical variables). Generic values in the domain are denoted by lowercase letters like x and a. An **atomic assignment** assigns a value $X = x$ to random variable x, where $x \in dom(X)$. A **joint distribution** P assigns a probability to each conjunction of atomic assignments; we write $P(X_1 = x_1, ..., X_n = x_n) = p$, sometimes abbreviated as $P(x_1, ..., x_n) = p$. To compactly refer to a set of variables like $\{X_1, ..X_n\}$ and an assignment of values $x_1, .., x_n$, we use boldface $\mathbf{X}$ and $\mathbf{x}$. If $\mathbf{X}$ and $\mathbf{Y}$ are sets of variables with $P(\mathbf{Y} = \mathbf{y}) > 0$, the **conditional probability** $P(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y})$ is defined as $P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y})/P(\mathbf{Y} = \mathbf{y})$.

We employ notation and terminology from [20, 23] for a Bayesian Network. A **Bayes net structure** is a directed acyclic graph (DAG) G, whose nodes comprise a set of random variables denoted by V. When discussing a Bayes net, we refer interchangeably to its nodes or its variables. The parents of a node X in graph G are denoted by $\mathbf{PA}_X^G$, and an assignment of values to the parents is denoted as $\mathbf{pa}_X^G$. When there is no risk of confusion, we often simply write $\mathbf{pa}_X$. A Bayes net is a pair $\langle G, \theta_G rangle$ where θ_G is a set of parameter values that specify the probability distributions of children conditional on instantiations of their parents, i.e. all conditional probabilities of the form $P(X = x|\mathbf{pa}_X^G)$. These conditional probabilities are specified in a **conditional probability table** (CP-table) for variable X. A BN $\langle G, \theta_G rangle$ defines a joint probability distribution over $V = \{v_1, .., v_n\}$ according to the formula

$$P(\mathbf{v} = \mathbf{a}) = \prod_{i=1}^{n} P(v_i = a_i|\mathbf{pa}_i = a_{\mathbf{pa}_i})$$

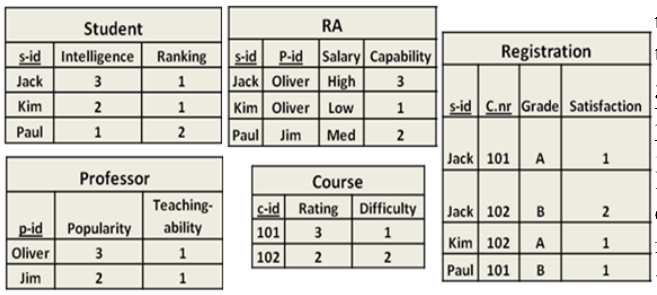
where a_i is the value of node v_i specified in assignment a, the term $a_{\mathbf{pa}_i}$ denotes the assignment of a value to each parent of v_i specified in assignment a, and $P(v_i = a_i|\mathbf{pa}_i = a_{\mathbf{pa}_i})$ is the corresponding CP-table entry. Thus the joint probability of an assignment is obtained by multiplying the conditional probabilities of each node value assignment given its parent value assignments.

3.2 Relational Schemas and First-Order Formulas We begin with a standard **relational schema** containing a set of tables, each with key fields, descriptive attributes, and foreign key pointers. A **database instance** specifies the tuples contained in the tables of a given database schema. Table reftable:university-schema shows a relational schema

$Student(\underline{student\_id}, intelligence, ranking)$
$Course(\underline{course\_id}, difficulty, rating)$
$Professor\ (\underline{professor\_id}, teaching\_ability, popularity)$
$reg\ (\underline{student\_id, course\_id}, grade, satisfaction)$
$ra\ (\underline{student\_id, prof\_id}, salary, capability)$

Table 1: A relational schema for a university domain. Key fields are underlined.

for a database related to a university (cf. [7]), and Figure reffig:university-tables displays a small database instance for this schema. A **table join** of two or more tables contains the rows in the Cartesian products of the tables whose values match on common fields.



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

RA			
s-id	P-id	Salary	Capability
Jack	Oliver	High	3
Kim	Oliver	Low	1
Paul	Jim	Med	2

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

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

Course		
c-id	Rating	Difficulty
101	3	1
102	2	2

Figure 1: A database instance for the schema in Table reftable:university-schema.

If the schema is derived from an entity-relationship model (ER model) [26, Ch.2.2], the tables in the relational schema can be divided into *entity tables* and *relationship tables.* Our algorithms generalize to any data model that can be translated into logical vocabulary based on first-order logic, which is the case for an ER model. The entity types of Schema reftable:university-schema are students, courses and professors. There are two relationship tables: $Registration$ records courses taken by each student and the grade and satisfaction achieved, while ra records research assistantship contracts between students and professors.

In our university example, there are two entity tables: a $Student$ table and a C table. There is one relationship table reg with foreign key pointers to the $Student$ and C tables whose tuples indicate which students have registered in which courses. Intuitively, an entity table corresponds to a type of entity, and a relationship table represents a relation between entity types.

It is well known that a relational schema can be translated into [26]. We follow logic-based approaches to SRL that use logic as a rigorous and expressive formalism for representing regularities found in the data. Specifically, we use first-order logic with typed variables and function symbols, as in [5, 21]. This formalism is rich enough to represent the constraints of an ER schema via the following translation: Entity sets correspond to types, descriptive attributes to functions, relationship tables to predicates, and foreign key constraints to type constraints on the arguments of relationship predicates.

Formulas in our syntax are constructed using three types of symbols: constants, variables, and functions. Sometimes we refer to logical variables as first-order variables, to distinguish them from random variables. A **type** or domain is a set of constants. In the case of an entity type, the constants correspond to primary keys in an entity table. Each constant and variable is assigned to a type, and so are the arguments (inputs) and the values (outputs) of functions. A predicate is a function whose values are the special truth values constants T, F. [5] discusses the appropriate use of first-order logic for SRL. While our logical syntax is standard, we give the definition in detail, to clarify the space of statistical patterns that our learning algorithms can be applied to, and to give a rigorous definition of the database frequency, which is key to our learning method. Table reftable:vocab defines the logical vocabulary. and illustrates the correspondence to an ER schema. A **term** θ is any expression that denotes a single object; the notation $\boldsymbol{\theta}$ denotes a vector or list of terms. If P is a predicate, we sometimes write $P(\boldsymbol{\theta})$ for $P(\boldsymbol{\theta}) = T$ and $\neg P(\boldsymbol{\theta})$ for $P(\boldsymbol{\theta}) = F$. Terms are recursively constructed as follows.

1. A constant or variable is a term.
2. Let f be a function term with argument type $\tau(f) = (\tau_1, \ldots, \tau_n)$. A list of terms $\theta_1, \ldots, \theta_n$ matches $\tau(f)$ if each θ_i is of type τ_i. If $\boldsymbol{\theta}$ matches the argument type of f, then the expression $f(\boldsymbol{\theta})$ is a **function term** whose type is $dom(f)$, the value type of f.

An **atom** is an equation of the form $\theta = \theta'$ where the types of θ and θ' match. A **negative literal** is an atom of the form $P(\boldsymbol{\theta}) = F$; all other atoms are **positive literals**. The formulas we consider are **conjunctions of literals**, or for short just conjunctions. We use the Prolog-style notation $Ł_1, \ldots, Ł_n$ for $Ł_1 \wedge \cdots \wedge Ł_n$, and vector notation $Ł, \mathbf{C}$ for conjunctions of literals. A term (literal) is **ground** if it contains no variables; otherwise the term (literal) is **open**. If $F = \{f_1, \ldots, f_n\}$ is a finite set of open function terms F, an **assignment** to F is a conjunction of the form $A = (f_1 = a_1, \ldots, f_n = a_n)$, where each a_i is a constant. A **relationship literal** is a literal with two or more variables.

A database instance $\mathcal{D}$ (possible world, Herbrand interpretation) assigns a denotation constant to each ground function term $f(\mathbf{a})$ which we denote by

$$[f(\mathbf{a})]_{\mathcal{D}}.$$

The value of descriptive relationship attributes is not defined for tuples that are not linked by the relationship. (For example, $grade(kim, 101)$ is not well defined in the

Description	Notation	Comment/Definition	Example
Constants	$T, F, \bot, a_1, a_2, \ldots$	$\bot$ for "undefined"	$\mathit{jack}, \mathit{101}, A, B, \mathit{1}, \mathit{2}, \mathit{3}$
Types	$\tau_1, \tau_2, \ldots$	list of types	$\mathit{Student}, A, B, C, D$
Functions	$f_1, f_2, \ldots$	number of arguments = arity of f	$\mathit{intelligence}, \mathit{difficulty}, \mathit{grade}$ $\mathit{Student}, \mathit{Registration}$
Function Domain	$\mathit{dom}(f) = \tau_i$	The value domain of f	$\mathit{dom}(\mathit{intelligence}) = \{\mathit{1}, \mathit{2}, \mathit{3}\}$ $\mathit{dom}(\mathit{Student}) = \{T, F\}$
Predicate	P, E, R, etc.	A function with domain $\{T, F\}$	$\mathit{Student}, \mathit{Course}, \mathit{Registration}$
Relationship	R, R', etc.	A predicate of arity greater than 1	$\mathit{Registration}$
(Entity) Types	$E_1, E_2, \ldots$	A set of unary predicates that are also types	$\mathit{Student}, \mathit{Course}$
(Typed) Variables	$X_1^\tau, X_2^\tau, \ldots$	A list of variables of type τ	$S_1^{Student}, S_2^{Student}, C^{Course}$ or simply S_1, S_2, C
Argument Type	$\tau(f) = (E_1, \ldots, E_n)$	A list of argument types for f.	$\tau(\mathit{Registration}) =$ $(\mathit{Student}, \mathit{Course})$
Descriptive Attribute	$f_1^P, f_2^P, \ldots$	Functions associated with a predicate P $\tau(f^P) = \tau(P)$	$\mathit{intelligence}, \mathit{difficulty}, \mathit{grade}$

Table 2: Definition of Logical Vocabulary with typed variables and function symbols. The examples refer to the database instance of Figure reffig:university-tables.

database instance of Figure reffig:university-tables since $\mathit{Registered}((\mathit{kim}, \mathit{101})$ is false in $\mathcal{D}$.) In this case, we assign the descriptive attribute the special value $\bot$ for "undefined". The general constraint is that for any descriptive attribute f^P of a predicate P,

$$[f^P(\mathbf{a})]_\mathcal{D} = \bot \iff [P(\mathbf{a})]_\mathcal{D} = F.$$

For a ground literal Ł, we write $\mathcal{D} \models$ Ł if Ł evaluates as true in $\mathcal{D}$, and $\mathcal{D} \not\models$ Ł otherwise. If P is a predicate, we sometimes write $\mathcal{D} \models P(\mathbf{a})$ for $\mathcal{D} \models (P(\mathbf{a}) = T)$ and $\mathcal{D} \models \neg P(\mathbf{a})$ for $\mathcal{D} \models (P(\mathbf{a}) = F)$. A ground conjunction $\mathbf{C}$ evaluates to true just in case each of its literals evaluate to true. We make the *unique names assumption* that distinct constants denote different objects, so

$$\mathcal{D} \models (\theta = \theta') \iff [\theta]_\mathcal{D} = [\theta']_\mathcal{D}.$$

If E is an entity type, the domain of E is the set of constants satisfying E (primary keys in the E table). The **domain of a variable** X^T of type T is the same as the domain of the type, so $\mathit{dom}_\mathcal{D}(X^T) = \mathit{dom}_\mathcal{D}(T)$. A **grounding** γ for a set of variables $X_1, \ldots, X_k$ assigns a constant of the right type to each variable X_i (i.e., $\gamma(X_i) \in \mathit{dom}_\mathcal{D}(X_i)$). If γ is a grounding for all variables that occur in a conjunction $\mathbf{C}$, we write $\gamma\mathbf{C}$ for the result of replacing each occurrence of a variable X in $\mathbf{C}$ by the constant $\gamma(X)$. The number of groundings that satisfy a conjunction $\mathbf{C}$ in $\mathcal{D}$ is defined as

$$\#_\mathcal{D} = |\{\gamma : \mathcal{D} \models \gamma\mathbf{C}\}|$$

where γ is any grounding for the variables in $\mathbf{C}$, and $|S|$ denotes the cardinality of a set S.

The ratio of the number of groundings that satisfy $\mathbf{C}$, over the number of possible groundings is called the **instantiation frequency** or the **database frequency** of $\mathbf{C}$. Formally we define

$$(3.1) \qquad P_\mathcal{D}(\mathbf{C}) = \frac{\#_\mathcal{D}(\mathbf{C})}{|\mathit{dom}_\mathcal{D}(X_1)| \times \cdots \times |\mathit{dom}_\mathcal{D}(X_k)|}$$

where $X_1, \ldots, X_k, k > 0$, is the set of variables that occur in $\mathbf{C}$.

Discussion. When all function terms contain the same single variable X, Equation (3.1) reduces to the standard definition of the single-table frequency of events. For example, $P_\mathcal{D}(\mathit{intelligence}(S) = \mathit{3})$ is the ratio of students with an intelligence level of 3, over the number of all students. Halpern gave a definition of the frequency with which a first-order formula holds in a given database [11, Sec.2], which assumes a distribution μ over the domain of each type. The instantiation frequency (3.1) is a special case of his with a uniform distribution μ over the elements of each domain [11, fn.1]. As Halpern explains, an intuitive interpretation of this definition is that it corresponds to generic regularities or random individuals, such as "the probability that a randomly chosen bird will fly is greater than .9". The conditional instantiation frequency plays an important role for discriminative learning in Inductive Logic Programming (ILP). For instance, the classic FOIL system generalizes entropy-based decision tree learning to first-order rules, where the goal is to predict a class label l. FOIL uses the conditional instantiation frequency $P_\mathcal{D}(l|\mathbf{C})$ to define the entropy of the empirical class distribution conditional on the body $\mathbf{C}$ of a rule.

Examples. The examples refer to the database instance $\mathcal{D}$ of Figure reffig:university-tables. The entity types are $\mathit{Student}, \mathit{Course}, \mathit{Professor}$. For each entity type we introduce one variable S, C, P. The constants of type *Student* are $\{jack, kim, paul\}$. True ground literals include $intelligence(jack) = 3$ and $\neg Registered(kim, 101)$. Table reftable:literals shows various open literals and their frequency in database $\mathcal{D}$ as derived from the number of true groundings.

Literal(s) L	$\#_{\mathcal{D}}(\text{Ł})$	$P_{\mathcal{D}}(\text{Ł})$
$intelligence(S) = 1$	1	1/3
$\neg intelligence(S) = 1$	2	2/3
$difficulty(C) = 2$	1	1/2
$Registered(S, C)$	4	4/6
$\neg Registered(S, C)$	2	2/6
$grade(S, C) = B$	2	2/6
$grade(S, C) = B$, $salary(S, P) = hi$	1	1/12

Table 3: Examples of open literals and their frequency in the database instance of Figure reffig:university-tables. The bottom four lines show relationship literals.

4 Structure Learning for Join Bayes Nets

We define the class of Bayes net models that we use to model the database distribution and present a structure learning algorithm.

DEFINITION 1. *Let $\mathcal{D}$ be a database with associated logical vocabulary vocab. A Join Bayes Net (JBN) structure for $\mathcal{D}$ is a directed acyclic graph whose nodes are a finite set $\{f_1(\boldsymbol{\theta_1}), \ldots, f_n(\boldsymbol{\theta_n})\}$ of open function terms. The domain of a node $v_i = f_i(\boldsymbol{\theta_i})$ is the range of f_i (the set of possible output values for f_i).*

Figure reffig:university-JBN shows an example of a Join Bayes net. We also refer to relationship terms that appear in a JBN as relationship indicator variables or simply relationship variables. A JBN assigns probabilities to conjunctions of literals of the form $f_1(\boldsymbol{\theta_1}) = a_1, \ldots, f_n(\boldsymbol{\theta_n}) = a_n$. We use the term "Join" because such conjunctions correspond to table joins in databases. A key step in model learning is to define an *empirical distribution* over the random variables in the model. Since the random variables in a JBN are function terms, this amounts to associating a probability with a conjunction $\mathbf{C}$ of literals given a database. We can use the instantiation frequency $P_{\mathcal{D}}(\mathbf{C})$ as the empirical distribution over the nodes in a JBN. The goal of JBN structure learning is then to construct a model of $P_{\mathcal{D}}$ given a database $\mathcal{D}$ as input.

4.1 The Learn-and-Join Algorithm We describe our structure learning algorithm, then discuss its scope and limitations. The algorithm is based on a general schema for

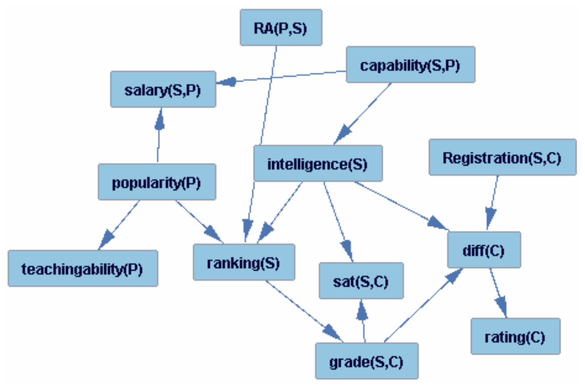


Figure 2: A Join Bayes Net for the relational schema shown in Table reftable:university-schema.

upgrading a propositional BN learner to a statistical relational learner. By "upgrading" we mean that the propositional learner is used as a function call or module in the body of our algorithm. We require that the propositional learner takes as input, in addition to a single table of cases, also a set of *edge constraints* that specify required and forbidden directed edges. The output of the algorithm is a DAG G for a database $\mathcal{D}$ with variables as specified in Definition refdef:jbn. Our approach is to "learn and join": we apply the BN learner to single tables and combine the results successively into larger graphs corresponding to larger table joins.

In principle, a JBN may contain any set of open function terms, depending on the attributes and relationships of interest. To keep the description of the structure learning algorithm simple, we assume that a JBN contains a default set of nodes as follows: (1) one node for each descriptive attribute, of both entities and relationships, (2) one Boolean indicator node for each relationship, (3) the nodes contain no constants. For each type of entity, we introduce one first-order variable. The algorithm has four phases (pseudocode shown as Algorithm refalg:structure).

(1) *Analyze single tables.* Learn a BN structure for the descriptive attributes of each entity table E of the database separately (with primary keys removed). The aim of this phase is to find within-table dependencies among descriptive attributes (e.g., $intelligence(S)$ and $ranking(S)$).

(2) *Analyze single join tables.* Each relationship table R is considered. The input table for each relationship R is the join of that table with the entity tables linked by a foreign key constraint (with primary keys removed). Edges between attributes from the same entity table E are constrained to agree with the structure learned for E in phase (1). Additional edges from variables corresponding to attributes from different tables may be added. The aim of this phase is to find dependencies between descriptive attributes *conditional on* the existence of a relationship. This phase

also finds dependencies between descriptive attributes of the relationship table R.

(3) *Analyze double join tables.* The extended input relationship tables from the second phase (joined with entity tables) are joined in pairs to form the input tables for the BN learner. Edges between variables considered in phases (1) and (2) are constrained to agree with the structures previously learned. The graphs learned for each join pair are merged to produce a DAG G. The aim of this phase is to find dependencies between descriptive attributes *conditional on* the existence of a relationship chain of length 2.

(4) *Satisfy slot chain constraints.* For each link $A \rightarrow B$ in G, where A and B are attributes from different tables, arrows from Boolean relationship variables into B are added if required to satisfy the following constraints: (1) A and B share variable among their arguments, or (2) the parents of B contain a chain of foreign key links connecting A and B. Figure reffig:structure-learn illustrates the increasingly large joins built up in successive phases. Algorithm refalg:structure gives pseudocode.

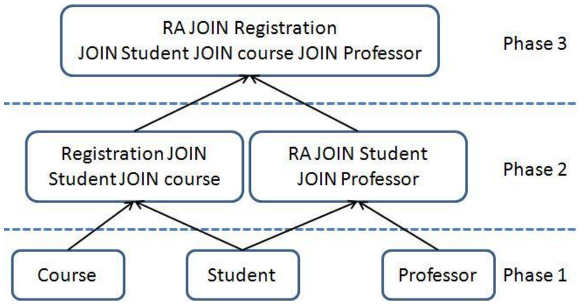


Figure 3: To illustrate the learn-and-join algorithm. A given BN learner is applied to each table and join tables (nodes in the graph shown). The presence or absence of edges at lower levels is inherited at higher levels.

4.2 Discussion

We discuss the patterns that can be represented in our current JBN implementation and the dependencies that can be found by the learn-and-join algorithm.

Number of Variables for each Entity Type. There are data patterns that require more than one variable per type to express. For a simple example, suppose we have a social network of friends represented by a $Friend(X,Y)$ relationship both of whose arguments are of type $Person$. There may be a correlation between the smoking of a person and that of her friends, so a JBN may place a link between nodes $smokes(X)$ and $smokes(Y)$, which requires two variables of type $Person$. In principle, the case of several variables for a given entity type can be translated to the case of one variable per entity type as follows [22]. For each variable of a given type, make a copy of the entity table. In our example, we would obtain two $Person$ tables, $Person_X$ and $Person_Y$. Each copy can be treated as its own type with just one associated variable. We leave for future work an efficient implementation of the learn-and-join algorithm with several variables for a given entity type.

Algorithm 1 Pseudocode for structure learning

```
Input: Database $\mathcal{D}$ with $E_1, ..E_e$ entity tables, $R_1, ...R_r$ Relationship tables,
Output: A JBN graph for $\mathcal{D}$.
Calls: PBN: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.
Notation: PBN($T$, Econstraints) denotes the output DAG of PBN. Get-Constraints($G$) specifies a new set of edge constraints, namely that all edges in $G$ are required, and edges missing between variables in $G$ are forbidden.
 1: Add descriptive attributes of all entity and relationship tables as variables to $G$. Add a boolean indicator for each relationship table to $G$.
 2: Econstraints = $\emptyset$ [Required and Forbidden edges]
 3: for m=1 to e do
 4:     Econstraints += Get-Constraints(PBN($E_m$ , $\emptyset$))
 5: end for
 6: for m=1 to r do
 7:     $N_m$ := join of $R_m$ and entity tables linked to $R_m$
 8:     Econstraints += Get-Constraints(PBN($N_m$, Econstraints))
 9: end for
10: for all $N_i$ and $N_j$ with an entity table foreign key in common do
11:     $K_{ij}$ := join of $N_i$ and $N_j$
12:     Econstraints += Get-Constraints(PBN($K_{ij}$, Econstraints))
13: end for
14: Construct DAG $G$ from Econstraints
15: Add edges from Boolean relationship variables to satisfy slot chain constraints
16: Return $G$
```

Correlation Coverage. The join-and-learn algorithm finds correlations between descriptive attributes, within a single table and between attributes from different linked tables. It does not, however, find dependencies between relationship variables (e.g., $Married(X,Y)$ predicts $Friend(X,Y)$). A JBN search for such dependencies could use local search methods such as those described in [14, 7]. In sum, the learn-and-join algorithm is suitable when the goal is to find correlations between descriptive attributes conditional on a given link structure.

5 Parameter Learning in Join Bayes Nets

This section treats the problem of computing conditional frequencies in the database distribution, which corresponds to computing sample frequencies in the single table case. The main problem is computing probabilities conditional on the *absence* of a relationship. This problem arises because a JBN includes relationship indicator variables such as $reg(S,C)$, and building a JBN therefore requires modelling the case where a relationship does not hold. We apply the recent virtual join (VJ) algorithm of [15] to address this computational bottleneck. The key constraint that the VJ algorithm seeks to satisfy is to *avoid enumerating the number of tuples that satisfy a negative relationship literal.* A numerical example illustrates why this is necessary. Consider a university database with 20,000 Students, 1,000 Courses and 2,000 TAs. If each student is registered in 10 courses, the size of a *Registered* table is 200,000, or in our notation $\#_{\mathcal{D}}(Registered(S,C))_{\mathcal{D}}) = 2 \times 10^5$. So the number of complementary student-course pairs is $\#_{\mathcal{D}}(\neg Registered(S,C))) = 2 \times 10^7 - 2 \times 10^5$, which is a much larger number that is too big for most database systems. If we consider joins, complemented tables are even more difficult to deal with: suppose that each course has at most 3 TAs. Then $\#_{\mathcal{D}}(Registered(S,C), TA(T,C)) < 6 \times 10^5$, whereas $\#_{\mathcal{D}}(\neg Registered(S,C), \neg TA(T,C))$ is on the order of 4×10^{10}. After explaining the basic idea behind the VJ algorithm, we give pseudocode for utilizing it in to estimate CP-table entries via joint probabilities. We conclude with a run-time analysis.

The Virtual Join algorithm for CP-tables. Instead of computing conditional frequencies, the VJ algorithm computes joint probabilities of the form $P_{\mathcal{D}}(child, parent\_values)$, which correspond to conjunctions of literals. Conditional probabilities are easily obtained from the joint probabilities by summation. While generally it is easier to find conditional rather than joint probabilities, there is an efficient dynamic programming scheme (informally a "1 minus" trick) that relies on the simpler structure of conjunctions. The enumeration of groundings for negative literals can be avoided by recursively applying a probabilistic principle (a "1-minus" scheme). Consider the equation $P(\mathbf{C}) = P(\mathbf{C}, \text{Ł}) + P(\mathbf{C}, \neg\text{Ł})$, which entails

$$P(\mathbf{C}, \neg\text{Ł}) = P(\mathbf{C}) - P(\mathbf{C}, \text{Ł}). \tag{5.2}$$

where $\mathbf{C}$ is a conjunction of literals and Ł a literal. This equation shows that the computation of a probability involving a negative relationship literal $\neg$Ł can be recursively reduced to two computations, one with the positive literal Ł and one with neither Ł nor $\neg$Ł, each of which contains one less negative relationship literal. In the base case, all literals are positive, so the problem is to find $P_{\mathcal{D}}(\mathbf{C})$ for database instance $\mathcal{D}$ where $\mathbf{C}$ contains positive relationship literals only. This can be done with a standard database table join. Figure reffig:example illustrates the recursion.

The VJ algorithm computes the database frequency for just a single input conjunction of literals. We adapt it for CP-table estimation with two changes: (1) We compute all frequencies that are defined by the same join table at once when the join has been built, and (2) we use the CP-table itself as our data structure for storing the results of intermediate computations from which other database frequencies are derived. The algorithm can be visualized as a dynamic program that successively fills in rows in a **joint probability table**, or **JP-table**, where we first fill in rows with 0 nonexistent relationships, then rows with 1 nonexistent relationship, etc. A JP-table is just like a CP-table whose rows correspond to joint probabilities rather than conditional probabilities. Algorithm refalg:adapted shows the pseudocode for the VJ algorithm.

Implementation and Complexity Analysis. The intermediate results of the computation are stored in an extended JP-table structure that features a third value $*$ for relationship predicates (in addition to T, F). This is used to represent the frequencies where a relationship predicate and its attributes are unspecified.

The algorithm satisfies the key constraint that it never enumerates the groundings that satisfy a negative relationship literal. A detailed complexity analysis of the VJ algorithm is given in [15]; we summarize the main points that are relevant for CP-table estimation. Essentially, each computation step fills in one entry in the extended JP-table. Compared to the CP-tables for a given Join Bayes net structure, our algorithm adds an extra auxilliary value * to the domain of each relationship indicator variable. Thus the increase in the size of the data structure is manageable compared to the CP-table in the original JBN. An important point is that the recursive update in Line refline:update does not require a database access. Data access occurs only in Lines refline:start-join– refline:end-join. Therefore the cost in terms of database accesses is essentially the cost of counting frequencies in a join table comprising all m relationships that occur in the child or parent nodes (Line refline:join with $i = m$). This computation is necessary for any algorithm that estimates the CP-table entries. It can be optimized using the tuple ID propagation method of [28, 29]. The crucial parameter for the complexity of this join is the number m of relationship predicates. In our learn-and-join algorithm, this parameter is bounded at $m = 2$. [15] provide an analysis whose upshot is that for typical SRL applications this parameter can be treated as a small constant. A brief summary of the reasons is as follows. (1) The space of models or rules that need to be searched becomes infeasible if too many relationships are considered at once. (2) Patterns that involve many relationships, for instance relationship chains of length 3 or more, are hard to understand for users. (3) Objects related by long relationship chains tend to carry less statistical information about a target object. We next examine empirically the performance of our our structure and parameter learning algorithms.

6 Empirical Evaluation of JBN Learning Algorithms

The learn-and-join algorithm trades off learning complexity with the coverage of data correlations. In this section

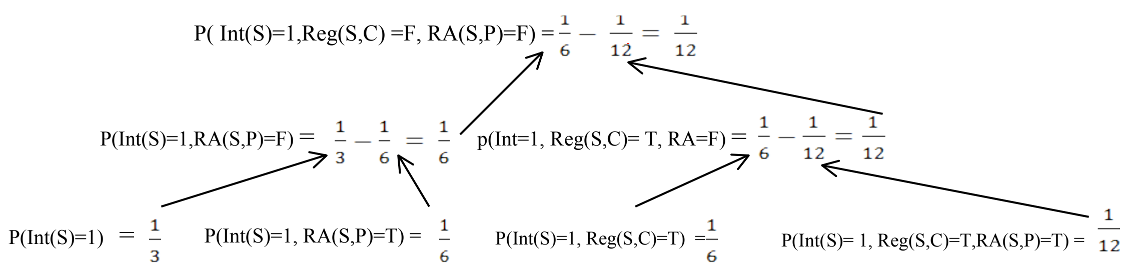

P(Int(S)=1,Reg(S,C) =F, RA(S,P)=F) = $\frac{1}{6} - \frac{1}{12} = \frac{1}{12}$
P(Int(S)=1,RA(S,P)=F) = $\frac{1}{3} - \frac{1}{6} = \frac{1}{6}$
p(Int=1, Reg(S,C)= T, RA=F) = $\frac{1}{6} - \frac{1}{12} = \frac{1}{12}$
P(Int(S)=1) = $\frac{1}{3}$
P(Int(S)=1, RA(S,P)=T) = $\frac{1}{6}$
P(Int(S)=1, Reg(S,C)=T) = $\frac{1}{6}$
P(Int(S)= 1, Reg(S,C)=T,RA(S,P)=T) = $\frac{1}{12}$

Figure 4: A frequency with negated relationships (nonexistent links) can be computed from frequencies that involve positive relationships (existing links) only. The leaves in the computation tree involves existing database tables only. The subtractions involve looking up results of previous computations. To reduce clutter, we abbreviated some of the predicates.

we evaluate both sides of this trade-off on three datasets: the run-time of the algorithm, especially as the dataset grows larger, and its performance in predicting database probabilities at the class level, which is the main task that motivates our algorithm. The more important dependencies the algorithm misses, the worse its predictive performance, so evaluating the predictions provides information about the quality of the JBN structure learned.

6.1 Implementation and Datasets All experiments were done on a QUAD CPU Q6700 with a 2.66GHz CPU and 8GB of RAM. The implementation used many of the procedures in version 4.3.9-0 of CMU's Tetrad package [25]. For single table BN search we used the Tetrad implementation of GES search [2] with the BDeu score (structure prior uniform, ESS=8). Edge constrains were implemented using Tetrad's "knowledge" functionality. JBN inference was carried out with Tetrad's Rowsum Exact Updater algorithm. Our datasets and code are available for ftp download from ftp://ftp.fas.sfu.ca/pub/cs/oschulte.

Datasets *University Database.* In order to check the correctness of our algorithms directly, we manually created a small dataset, based on the schema given in reftable:university-schema. The entity tables contain 38 students, 10 courses, and 6 Professors. The reg table has 92 rows and the RA table has 25 rows.

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

Financial Database. The third dataset is a modified version of the financial dataset from the PKDD 1999 cup. There are two entity tables, $Client$ with 5,369 tuples and $Account$ with 4,500 tuples. Two relationship tables, $CreditCard$ with 2,676 tuples and $Disposition$ with 5,369 tuples relate a client with an account. The Client table has 10 descriptive attributes: the client's age, gender and 8 attributes on demographic data of the client. The Account table has 3 descriptive attributes: information on loan amount associated with an account, account opening date, and how frequently the account is used.

6.2 Learning: Experimental Design To benchmark the runtimes of our learning algorithm, we applied the structure learning routine `learnstruct` (default options) of the Alchemy package for MLNs [5]. We chose MLNs for the following reasons: (1) MLNs are currently one of the most active areas of SRL research (e.g., [5, 12]). Part of the reason for this is that undirected graphical models avoid the computational and representational problems caused by cycles in instance-level directed models (Section refsec:related). Discriminative MLNs can be viewed as logic-based templates for conditional Markov random fields, a prominent formalism for relational classification [24]. (2) Alchemy provides open-source, state-of-the-art learning software for MLNs. Structure learning software for alternative systems like BLNs and PRMs is not easily available.[1] (3) BLNs and PRMs require the specification of additional structure like aggregation functions or combining rules. This confounds our experiments with more parameters to specify. Also, incorporating the extra structure complicates learning for these formalisms, so arguably a direct comparison with JBN learning is not fair. In contrast, the MLN formalism, like the JBN, does not re-

[1]The webpage [13] lists different SRL systems and which software is available for them. The Balios BLN engine [14] supports only parameter learning, not structure learning; see [13]. We could not obtain source code for PRM structure learning.

Algorithm 2 A dynamic program for estimating JP-table entries in a Join Bayes Net.

```
Notation: A row $r$ corresponds to a partial or complete assignment for function terms. The value assigned to function term $f(\theta)$ in row $r$ is denoted by $f_r$. The probability for row $r$ stored in JP-table $\tau$ is denoted by $\tau(r)$.
Input: database $\mathcal{D}$; child variable and parent variables divided into a set $\mathbf{R_1}, \ldots, \mathbf{R_m}$ of relationship predicates and a set $\mathbf{C}$ of function terms that are not relationship predicates.
Calls: initialization function JOIN-FREQUENCIES($\mathbf{C}, T = R_1 = \cdots = R_k$). Computes join frequencies conditional on relationships $R_1, \ldots, R_k$ being true.
Output: Joint Probability table $\tau$ such that the entry $\tau(r)$ for row $r \equiv (\mathbf{C} = \mathbf{C}_r, \mathbf{R} = \mathbf{R}_r)$ is the frequencies $P_{\mathcal{D}}(\mathbf{C} = \mathbf{C}_r, \mathbf{R} = \mathbf{R}_r)$ in the database distribution $\mathcal{D}$.
1: {fill in rows with no false relationships using table joins}
2: for all valid rows $r$ with no assignments of $F$ to relationship predicates do
3:   for $i = 0$ to $m$ do
4:     if $r$ has exactly $i$ true relationship literals $R^1, .., R^i$ then {$r$ has $m-i$ unspecified relationship literals}
5:       find $P_{\mathcal{D}}(\mathbf{C} = \mathbf{C}_r, \mathbf{R} = \mathbf{R}_r)$ using JOIN-FREQUENCIES($\mathbf{C} = \mathbf{C}_r, \mathbf{R} = \mathbf{R}_r$). Store the result in $\tau(r)$.
6:     end if
7:   end for
8: end for
9: {Recursively extend the table to JP-table entries with false relationships.}
10: for all rows $r$ with at least one assignment of $F$ to a relationship predicate do
11:   for $i = 1$ to $m - 1$ do
12:     if $r$ has exactly $i$ false relationship literals $R^1, .., R^i$ then {find conditional probabilities when $R^1$ is true and when unspecified}
13:       Let $r_T$ be the row such that (1) $R^1_{r_T} = T$, (2) $f^{R^1}_{r_T}$ is unspecified for all attributes $f^{R^1}$ of $R^1$, and (3) $r_T$ matches $r$ on all other variables.
14:       Let $r_*$ be the row that matches $r$ on all variables $f$ that are not $R^1$ or an attribute of $R^1$ and has $R^1_{r_*}$ unspecified.
15:       {The rows $r_*$ and $r_T$ have one less false relationship variable than $r$.}
16:       Set $\tau(r) := \tau(r_*) - \tau(r_T)$.
17:     end if
18:   end for
19: end for
```

quire extra structure beyond the relational schema.

6.3 Learning: Results We present results of applying our learning algorithms to the three relational datasets. The resulting JBNs are shown in Figures reffig:university-JBN, reffig:structmovie, and reffig:structfinancial. In the MovieLens dataset, the algorithm finds a number of cross-entity table links involving the age of a user. Because genres have a high negative correlation, the algorithm produces a dense graph among the genre attributes. The richer relational structure of the Financial dataset is reflected in a more complex graph with several cross-table links. The birthday of a customer (translated into discrete age levels) has especially many links with other variables. The CP-tables for the learned graph structures were filled in using the dynamic programming algorithm refalg:adapted. Table reftable:runtime presents a summary of the run times for the datasets.

The databases translate into ground atoms for Alchemy input as follows: University 390, MovieLens 39,394, and Financial 16,129. On our system, Alchemy was able to process the University database, but did not have sufficient computational resources to return a result for the MovieLens and Financial data. We therefore subsampled the datasets to obtain small databases on which we can compare Alchemy's runtime with that of the join-and-learn algorithm. Because Alchemy returned no result on the complete datasets, we formed three subdatabases by randomly selecting entities for each dataset. We restricted the relationship tuples in each subdatabase to those that involve only the selected entities. The resulting subdatabase sizes are as follows. For MovieLens, (i) 100 users, 100 movies = 1,477 atoms (ii) 300 users, 300 movies = 9698 atoms (iii) 500 users, 400 movies = 19,053 atoms. For Financial, (i) 100 Clients, 100 Accounts = 3,228 atoms, (ii) 300 Clients, 300 Accounts = 9,466 atoms, (iii) 500 Clients, 500 Accounts = 15,592 atoms. For the Financial dataset, which contains numerous descriptive attributes, Alchemy returned a result only for the smallest subdatabase (i). Table reftable:runtime shows that the runtime of the JBN learning algorithm applied to the entire dataset is 600 times faster than Alchemy's learning time on a dataset about half the size. We emphasize that this is not a criticism of Alchemy structure learning, which aims to find a structure that is optimal for instance-level predictions (cf. Section refsec:related). Rather, it illustrates that the task of finding class-level dependencies is much less computationally complex taken as an independent task than when it is taken in conjunction with optimizing instance-level predictions. The next section compares the probabilities estimated by the JBN with the database frequencies computed directly from SQL queries.

6.4 Inference: Experimental Design A direct comparison of class-level inference with other SRL formalisms

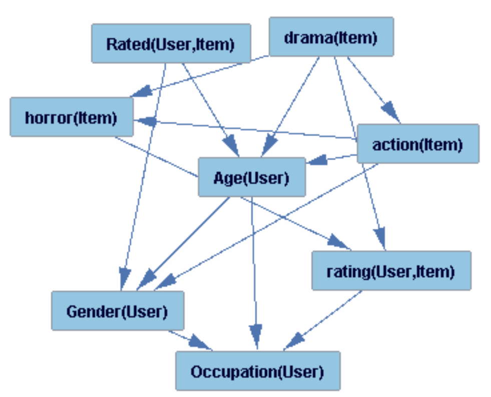


Figure 5: The JBN structure learned by our merge learning algorithm refalg:structure for the MovieLens Dataset.

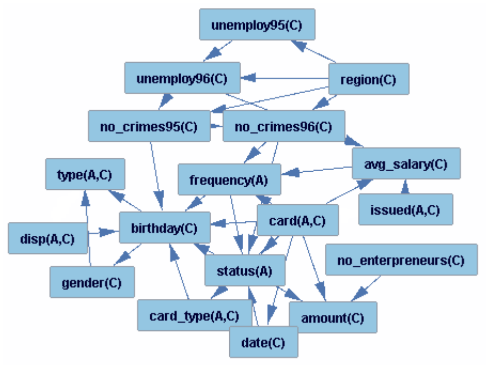


Figure 6: The JBN structure learned by our merge learning algorithm refalg:structure for the Financial Dataset.

is difficult as the available implementations support only instance-level queries.[2] This reflects the fact that current SRL systems are designed for the task of predicting attributes of individual entities, rather than class-level prediction. We therefore evaluated the class-level predictions of our system against gold standard frequency counts obtained directly from the data using SQL queries. We follow the approach of [9] in our experiments and use the sample frequencies for parameter estimation. This is appropriate when the goal is to evaluate whether the statistical model adequately summarizes the data distribution. If the data frequencies are smoothed to support generalizations beyond the data, the first step is to compute the data frequencies, so our experiments are relevant to this case too.

We randomly generated 10 queries for each data set according to the following procedure. First, choose randomly a target node V and a value a such that $P(V = a)$ is the probability to be predicted. Then choose randomly the number k of conditioning variables, ranging from 1 to 3. Make a random selection of k variables $V_1, \ldots, V_k$ and corresponding values $a_1, \ldots, a_k$. The query to be answered is then $P(V = a | V_1 = a_1, \ldots, V_k = a_k)$. Table reftable:inference shows representative test queries. It compares the probabilities predicted by the JBN with the frequencies in the database as computed by an SQL query, as well as the runtimes for computing the probability using the JBN vs. the SQL.

[2]For example, Alchemy and the Balios BLN engine [14] support only queries with ground atoms as evidence. We could not obtain source code for PRM inference.

Dataset	JBN		MLN-subdatasets		
	PL	SL	small	medium	large
University	0.495	0.64			90
Movie Lens	35	15	455	1,316	33,656
Financial	136	52	21,607	n/a	n/a

Table 4: The runtimes—in seconds—for structure learning (SL) and parameter learning (PL) on our three datasets.

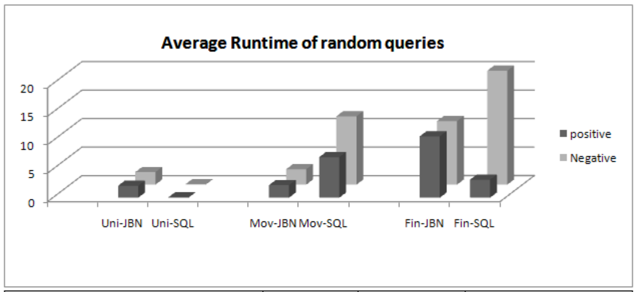
6.5 Inference: Results The averages reported are taken over 10 random queries for each dataset. We see in Table reftable:inference and Figure reffig:results that the predicted probabilities are very close to the data frequencies: The average difference is less than 3% for MovieLens, and less than 8% for Financial. The measurements for Financial are taken on queries with positive relationship literals only, because the SQL queries that involve negated relationships did not terminate with a result for the Financial dataset. The graph shows the nontermination as corresponding to a high time-out number.

Observations about processing speed that hold for all datasets include the following. (1) For queries involving *negated relationships*, JBN inference was much faster on the MovieLens dataset. SQL queries with negated relationships were infeasible on the Financial dataset, whereas the JBN returns an answer in around 10 seconds. (2) Both JBN and SQL queries speed up as the number of conditioning variables increases. (3) Higher probability queries are slower with SQL, because they correspond to larger joins, whereas the size of the probability does not affect JBN query processing.

Other observations depend on the difference between the datasets: MovieLens has relatively many tuples and few attributes, whereas Financial has relatively few tuples and many attributes. These factors differentially affected the performance of SQL vs. JBN inference as follows (for queries involving positive relationships only). (1) A larger number of attributes and/or a larger number of categories in the attributes in the schema decreases the speed of both JBN inference and SQL queries. But the slowdown is greater for JBN queries because the required marginalization steps are expensive (summing over possible values of nodes). (2) The number of tuples in the database table is a very significant

Dataset	**Query**	**P(Model)/time**	**P(SQL/time)**
University	p($ranking = 3 \| RA = F, popularity = 1, intelligence = 2$)	0.6048/ 2.12	0.6086/ 0.06
University	p($grade = 3 \| registration = T, popularity = 1, intelligence = 2$)	0.1924/2.03	0.2/0.064
MovieLens	p($Age = 2 \| Rated = T, rating = 4, horror = 1$)	0.0878/0.78	0.09 /12.23
MovieLens	p($Age = 1 \| Rated = F, rating = 4, horror = 1$)	0.5433/ 1.23	0.5166/14.34
MovieLens	p($Gender = m \| Rated = F, Age = 1, horror = 1, action = 1$)	0.6904/0.97	0.6857/9.66
MovieLens	p($horror = 1 \| Rated = T, Age = 1, Gender = F, action = 1$)	0.0433/1.20	0.0293/7.23
Financial	p($Gender = M \| date = 1, status = A$)	0.5211/11.32	0.5391/6.37
Financial	p($Gender = M \| date = 1, status = A, Disp = F$)	0.5187/10.65	No result

Table 5: The table shows representative randomly generated queries. We compare the probability estimated by our learned JBN model (P(Model)) with the database frequency computed from direct SQL queries (P(SQL)), and the execution times (in seconds) for each inference method. The SQL query in the last line exceeded our system resources. For simplicity we omitted first-order variables.



Dataset	University	MovieLens	Financial(positive)
Average probability difference	0.003	0.027	0.075

Figure 7: Comparing the probability estimates and times (in sec) for obtaining them from the learned JBN models versus SQL queries.

factor for the speed of SQL queries but does not affect JBN inference. This last point is an important observation about the *data scalability of JBN inference*: While the cost of the learning algorithms depends on the size of the database, once the learning is completed, query processing is independent of database size. So for applications like query optimization that involve many calls to the statistical inference procedure, the investment in learning a JBN model is quickly amortized in the fast inference time.

7 Conclusion

Class-level generic dependencies between attributes of linked objects and of links are important in themselves, and they support applications like policy making, strategic planning, and query optimization. The focus on class-level dependencies brings gains in tractability of learning and inference. The theoretical foundation of our approach is classic AI research which established a definition of the frequency of a first-order formula in a given relational database. We described efficient and scalable algorithms for structure learning and parameter estimation in Join Bayes nets, which model the database frequencies over attributes of linked objects and links. Our algorithms upgrade a single-table Bayes net learner as a self-contained module to perform relational learning. JBN inference can be carried out with standard algorithms "as is" to answer class-level probabilistic queries. An evaluation of our methods on three data sets shows that they are computationally feasible for realistic table sizes, and that the learned structures represented the statistical information in the databases well. Querying database statistics via the net is often faster than directly with SQL queries, and does not depend on the size of the database.

A fundamental limitation of our approach is that Join Bayes nets do not directly support instance-level queries (our model does not include a solution to the combining problem). Limitations that can be addressed in future research include restrictions on the types of correlation that our structure learning algorithm can discover, such as dependencies between relationships, and dependencies that require more than one first-order variable per entity type to represent.

Acknowledgments

This research was supported by Discovery Grants to the first and fifth author from the Natural Sciences and Engineering Research Council of Canada. A preliminary version of our results was presented at the STRUCK and GKR workshops at IJCAI 2009, and at the Computational Intelligence Forum at the University of British Columbia. We thank the audiences and Ke Wang for helpful comments.

References

[1] F. Bacchus. *Representing and reasoning with probabilistic knowledge: a logical approach to probabilities*. MIT Press, Cambridge, MA, USA, 1990.

[2] D. M. Chickering and C. Meek. Finding optimal bayesian networks. In *UAI*, pages 94–102, 2002.

[3] L. de Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[4] R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Introduction to Statistical Relational Learning* [10], chapter 15, pages 433–452.

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

[6] R. Frank, M. Ester, and A. Knobbe. A multi-relational approach to spatial classification. In J. F. E. IV, F. Fogelman-Soulié, P. Flach, and M. Zaki, editors, *KDD*, pages 309–318. ACM, 2009.

[7] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning* [10], chapter 5, pages 129–173.

[8] L. Getoor and B. Taskar. Introduction. In Getoor and Taskar [10], pages 1–8.

[9] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. *ACM SIGMOD Record*, 30(2):461–472, 2001.

[10] L. Getoor and B. Tasker. *Introduction to statistical relational learning*. MIT Press, 2007.

[11] J. Y. Halpern. An analysis of first-order logics of probability. *Artif. Intell.*, 46(3):311–350, 1990.

[12] T. N. Huynh and R. J. Mooney. Discriminative structure and parameter learning for markov logic networks. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 416–423, New York, NY, USA, 2008. ACM.

[13] K. Kersting. Probabilistic-logical model repository. URL = `http://www.informatik.uni-freiburg.de/~kersting/plmr/`.

[14] K. Kersting and L. de Raedt. Bayesian logic programming: Theory and tool. In *Introduction to Statistical Relational Learning* [10], chapter 10, pages 291–318.

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

[16] D. Koller and A. Pfeffer. Object-oriented bayesian networks. In D. Geiger and P. P. Shenoy, editors, *UAI*, pages 302–313. Morgan Kaufmann, 1997.

[17] B. J. McMahan, G. Pan, P. Porter, and M. Y. Vardi. Projection pushing revisited. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2004.

[18] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. Tom M. Mitchell. Includes bibliographical references and indexes. 1. Introduction – 2. Concept Learning and the General-to-Specific Ordering – 3. Decision Tree Learning – 4. Artificial Neural Networks – 5. Evaluating Hypotheses – 6. Bayesian Learning – 7. Computational Learning Theory – 8. Instance-Based Learning – 9. Genetic Algorithms – 10. Learning Sets of Rules – 11. Analytical Learning – 12. Combining Inductive and Analytical Learning – 13. Reinforcement Learning.

[19] J. Nevile and D. Jensen. Relational dependency networks. In *An Introduction to Statistical Relational Learning* [10].

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

[21] D. Poole. First-order probabilistic inference. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 985–991. Morgan Kaufmann, 2003.

[22] O. Schulte, H. Khosravi, F. Moser, and M. Ester. Join bayes nets: A new type of bayes net for relational data. Technical Report 2008-17, Simon Fraser University, 2008. also in CS-Learning Preprint Archive.

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

[24] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2002.

[25] C. M. U. The Tetrad Group, Department of Philosophy. The tetrad project: Causal models and statistical data, 2008. http://www.phil.cmu.edu/projects/tetrad/.

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

[27] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[28] X. Yin and J. Han. Exploring the power of heuristics and links in multi-relational data mining. In *Springer-Verlag Berlin Heidelberg 2008*, 2008.

[29] X. Yin, J. Han, J. Yang, and P. S. Yu. Crossmine: Efficient classification across multiple database relations. In *Constraint-Based Mining and Inductive Databases*, pages 172–195, 2004.