Springer

Dear Author,

Here are the proofs of your article.

- You can submit your corrections **online**, via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and **email** the annotated PDF.
- For fax submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the **journal title**, **article number**, and **your name** when sending your response via e-mail or fax.
- **Check** the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- **Check** the questions that may have arisen during copy editing and insert your answers/ corrections.
- **Check** that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style.
  Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections **within 48 hours**, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The **printed version** will follow in a forthcoming issue.

Please note

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL: http://dx.doi.org/[DOI].
If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information go to: http://www.link.springer.com.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us if you would like to have these documents returned.

Metadata of the article that will be visualized in OnlineFirst

Please note: Images will appear in color online but will be printed in black and white.

ArticleTitle	FACTORBASE: multi-relational structure learning with SQL all the way	
Article Sub-Title		
Article CopyRight	Springer International Publishing AG, part of Springer Nature (This will be the copyright line in the final PDF)	
Journal Name	International Journal of Data Science and Analytics	

Corresponding Author	Family Name	**Qian**
	Particle	
	Given Name	**Zhensong**
	Suffix	
	Division	
	Organization	Simon Fraser University
	Address	Burnaby, Canada
	Phone	+1-778-782-7008
	Fax	
	Email	zqian@sfu.ca
	URL	
	ORCID	http://orcid.org/0000-0002-5315-5802
Author	Family Name	**Schulte**
	Particle	
	Given Name	**Oliver**
	Suffix	
	Division	
	Organization	Simon Fraser University
	Address	Burnaby, Canada
	Phone	+1-778-782-3390
	Fax	
	Email	oschulte@sfu.ca
	URL	
	ORCID	

Schedule	Received	30 April 2017
	Revised	
	Accepted	19 May 2018

Abstract — FACTORBASE is a new SQL-based framework that leverages a relational database management system to support multi-relational model discovery. A multi-relational statistical model provides an integrated analysis of the heterogeneous and interdependent data resources in the database. We adopt the BayesStore design philosophy: Statistical models are stored and managed as first-class citizens inside a database (Wang et al., in: PVLDB, pp 340–351, 2008). Whereas previous systems like BayesStore support multi-relational inference, FACTORBASE supports multi-relational learning. A case study on six benchmark databases evaluates how our system supports a challenging machine learning application, namely learning a first-order Bayesian network model for an entire database. Model learning in this setting has to examine a large number of potential statistical associations across data tables. Our implementation shows how the

	SQL constructs in FACTORBASE facilitate the fast, modular, and reliable development of scalable model learning systems.
Keywords (separated by '-')	Relational learning - Bayesian networks - Model selection - Relational database management systems
Footnote Information	

International Journal of Data Science and Analytics
https://doi.org/10.1007/s41060-018-0130-1

REGULAR PAPER

FACTORBASE: multi-relational structure learning with SQL all the way

Oliver Schulte[1] · Zhensong Qian[1]

Received: 30 April 2017 / Accepted: 19 May 2018

Abstract
FACTORBASE is a new SQL-based framework that leverages a relational database management system to support multi-relational model discovery. A multi-relational statistical model provides an integrated analysis of the heterogeneous and interdependent data resources in the database. We adopt the BayesStore design philosophy: Statistical models are stored and managed as first-class citizens inside a database (Wang et al., in: PVLDB, pp 340–351, 2008). Whereas previous systems like BayesStore support multi-relational inference, FACTORBASE supports multi-relational learning. A case study on six benchmark databases evaluates how our system supports a challenging machine learning application, namely learning a first-order Bayesian network model for an entire database. Model learning in this setting has to examine a large number of potential statistical associations across data tables. Our implementation shows how the SQL constructs in FACTORBASE facilitate the fast, modular, and reliable development of scalable model learning systems.

Keywords Relational learning · Bayesian networks · Model selection · Relational database management systems

1 Introduction

Data science brings together ideas from different fields for extracting value from large complex datasets. The system described in this paper combines advanced analytics from multi-relational or *statistical–relational* machine learning (SRL) with database systems. The power of combining machine learning with database systems has been demonstrated in several systems [3,14,19]. The novel contribution of FACTORBASE is supporting machine learning for *multi-relational* data, rather than for traditional learning where the data are represented in a *single* table or data matrix. We discuss new challenges raised by multi-relational model learning compared to single-table learning, and how FACTORBASE solves them using the resources of SQL (Structured Query Language). The name FACTORBASE indicates that our system supports learning factors that define a log-linear multi-relational model [18]. Supported new database services include constructing, storing, and transforming complex statistical objects, such as factor tables, cross-table sufficient statistics, parameter estimates, and model selection scores.

Multi-relational data have a complex structure that integrate heterogeneous information about different types of entities (customers, products, factories, etc.) and different types of relationships among these entities. A statistical–relational model provides an integrated statistical analysis of the heterogeneous, interdependent, and complex data resources maintained by the database system. 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. [4,11,25]. Database researchers have noted the usefulness of statistical–relational models for knowledge discovery and for representing uncertainty in databases [39,42]. They have developed a system architecture where statistical models are stored as first-class citizens *inside a database*. The goal is to seamlessly integrate query processing and statistical–relational inference. These systems focus on inference *given* a statistical–relational model, not on *learning* the model from the data stored in the RDBMS. The FACTORBASE system complements the in-database probabilistic inference systems by providing an in-database probabilistic model learning system.

✉ Zhensong Qian
zqian@sfu.ca

Oliver Schulte
oschulte@sfu.ca

[1] Simon Fraser University, Burnaby, Canada

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

1.1 Evaluation

We evaluate our approach on six benchmark databases. For each benchmark database, the system applies a state-of-the-art SRL algorithm to construct a statistical–relational model. Our experiments show that FACTORBASE pushes the scalability boundary: Learning scales to databases with over 10^6 records, compared to less than 10^5 for previous systems. At the same time, it is able to discover more complex cross-table correlations than previous SRL systems. We report experiments that focus on the key services for an SRL client:

- Computing and caching sufficient statistics (event counts).
- Computing model predictions on test instances.
- Constructing, storing, and evaluating a graphical model and the factors that define it.

For the largest benchmark database, our system handles 15M sufficient statistics. SQL facilitates block prediction for a set of test instances, which leads to a ten- to 100-fold speedup compared to a simple loop over test instances.

1.2 Contributions

FACTORBASE is the first system that leverages relational query processing for learning a multi-relational log-linear graphical model. Whereas the in-database design philosophy has been previously used for multi-relational inference, we are the first to adapt it for multi-relational model structure learning. Pushing the graphical model inside the database allows us to *use SQL as a high-level scripting language for SRL*, with the following advantages.

1. Extensibility and modularity, which support rapid prototyping. SRL algorithm development can focus on statistical issues and rely on a RDBMS for data access and model management.
2. Increased scalability, in terms of both the size and the complexity of the statistical objects that can be handled.
3. Generality and portability: Standardized database operations support "out-of-the-box" learning with a minimal need for user configuration.

1.3 Paper organization

We provide an overview of the system components and flow. For each component, we describe how the component is constructed and managed inside an RDBMS using SQL scripts and the SQL view mechanism. We show how the system manages sufficient statistics and test instance predictions. The evaluation section demonstrates the scalability advantages of in-database processing. The intersection of machine learning and database management has become a densely researched area, so we end with an extensive discussion of related work.

2 Background on statistical–relational learning

We review background from statistical–relational models and structure learning to motivate our system design. The extensive survey by Kimmig et al. [18] provides further details. The survey shows that SRL models can be viewed as log-linear models based on par-factors as follows. This includes well-known models such as Markov Logic Networks [4] and parametrized Bayesian networks [28,34]. This section gives the general mathematical definitions for the concepts, equations, and computations required for log-linear models. The main body of the paper describes how our FACTORBASE system uses RDBMS capabilities and SQL to implement the mathematical definitions.

2.1 Log-linear template models for relational data

Par-factor stands for "parametrized factor." A par factor represents an interaction among parametrized random variables, or par-RVs for short. We employ the following notation for par-RVs [18, 2.2.5]. Constants are expressed in lower case, for example *joe*, and are used to represent entities. A type is associated with each entity; for example, *joe* is a person. A first-order variable is also typed; for example, *Person* denotes some member of the class of persons. A functor maps a tuples of entities to a value. We assume that the range of possible values is finite. An *atom* is an expression of the form $r(\tau_1, \ldots, \tau_a)$, where each τ_i is either a constant or a first-order variable. If all of $\tau_1, \ldots, \tau_a$ are constants, $r(\tau_1, \ldots, \tau_a)$ is a *ground atom* or random variable (RV), otherwise a *first-order atom* or a **par-RV**. A par-RV is instantiated to an RV by grounding, i.e., substituting a constant of the appropriate domain for each first-order variable.

A **par-factor** is a pair $\Phi = (\mathsf{A}, \phi)$, where A is a set of par-RVs, and ϕ is a **potential function** from the values of the par-RVs to the nonnegative real numbers.[1] A **factor assignment** $\mathbf{x}_\mathsf{A}$ assigns a value to each par-RV in a parfactor. The factor potential function maps each factor assignment $\mathbf{x}_\mathsf{A}$ to a real number $\phi(\mathbf{x}_\mathsf{A})$.

A **factor grounding** assigns a constant to each first-order variable in the par-RV set A; the result is a ground par-factor $\mathbf{A}$. Intuitively, a ground par-factor represents a set of random variables that interact with each other locally. SRL models use *parameter tying*, meaning that if two groundings of the same par-factor are assigned the same values, they return the

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

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

same factor value. Let $\mathscr{I}(\Phi)$ denote the set of all ground par-RVs in par-factor Φ.

A relational log-linear model is defined by a set of par-factors $\mathscr{F}$. A set of par-factors $\mathscr{F}$ defines a joint probability distribution over the ground par-RVs as follows. Let $\mathbf{x}$ be a joint assignment of values to all ground random variables. Notice that this assignment determines the values of all ground atoms. An assignment $\mathbf{X} = \mathbf{x}$ is therefore *equivalent to a single database instance*. The probability of a database instance is given by the log-linear equation [18, Eq.7]:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\Phi=(\mathsf{A},\phi)\in\mathscr{F}} \prod_{\mathbf{A}\in\mathscr{I}(\Phi)} \phi(\mathbf{x}_\mathbf{A}) \tag{1}$$

where $\mathbf{x}_\mathbf{A}$ represents the values of those ground variables in $\mathbf{A}$ that are necessary to compute ϕ and Z is a normalization constant. Equation 1 can be evaluated without enumerating the ground par-factors as follows.

1. For each par-factor $\Phi = (\mathsf{A}, \phi)$, for each possible factor assignment $\mathbf{x}_\mathsf{A}$, find the number of ground factors with that assignment of values. This number of ground factors with the same assignment of values is known as a **sufficient statistic**; we denote it by $n(\mathbf{x}_\mathsf{A})$.
2. For each factor assignment, raise the potential function value $\phi(\mathbf{x}_\mathsf{A})$ to the number $n(\mathbf{x}_\mathsf{A})$ of ground factors that instantiate it in the database $\mathbf{X} = \mathbf{x}$.
3. Multiply the exponentiated potential function values.

This procedure corresponds to the equation

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\Phi=(\mathsf{A},\phi)\in\mathscr{F}} \prod_{\mathbf{x}_\mathsf{A}} \phi(\mathbf{x}_\mathsf{A})^{n(\mathbf{x}_\mathsf{A})} \tag{2}$$

where the inner product runs over all possible factor assignments $\mathbf{x}_\mathsf{A}$ for the par-factor Φ. Notice that the set of possible factor assignments does not depend on the data and is typically much smaller than the number of ground par-factors $\mathscr{I}(\Phi)$ indexed in the inner product of Eq. 1. Given the sufficient statistics $n(\mathbf{x}_\mathsf{A})$, Eq. 2 therefore defines a computationally much more efficient method for computing the same result as Eq. 1. With finite floating-point precision, large products give rise to 0 multiplication issues. Therefore, it is common to use the logarithmic version of Eq. 2, which defines the **log-likelihood** of a database instance given a set of par-factors:

$$\ln P(\mathbf{X} = \mathbf{x}) = \sum_{(\mathsf{A},\phi)\in\mathscr{F}} \sum_{\mathbf{x}_\mathsf{A}} n(\mathbf{x}_\mathsf{A}) \ln \phi(\mathbf{x}_\mathsf{A}) - \ln(Z) \tag{3}$$

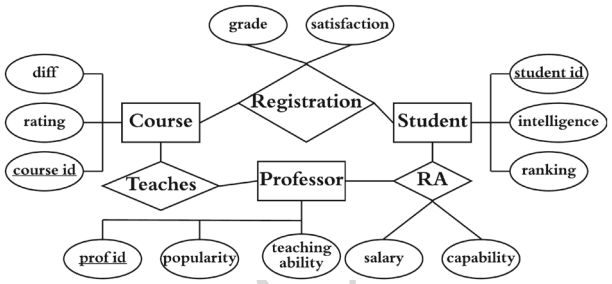

Fig. 1 A relational ER design for a university domain

Table 1 Tables in an example database instance

s_id	Intelligence	Ranking
Student		
Jack	3	1
Kim	2	1
Paul	1	2

p_id	Popularity	Teachingability
Professor		
Jim	2	1
Oliver	3	1
David	2	2

s_id	p_id	Salary	Capability
RA			
Jack	Oliver	High	3
Kim	Oliver	Low	1
Paul	Jim	Med	2
Kim	David	High	2

2.2 Examples

SRL has developed a number of formalisms for describing par-factors [18]. First-order probabilistic graphical models are popular within both SRL and the database community [18,42]. The model structure is defined by edges connecting par-RVs. For instance, a **parametrized Bayesian network structure** is a directed acyclic graph whose nodes are par-RVs. Figure 2 shows a Bayesian network for a university domain.

We use the university example as a toy running example throughout the paper. The schema for the university domain is given in Fig. 1. This schema features only one relationship for simplicity; FACTORBASE learns a model for any number of relationships. While we describe FACTORBASE abstractly in terms of par-factors, for concreteness we illustrate it using Bayesian networks. The system takes as input a database instance like that shown in Table 1 and produces as output a graphical model like that shown in Fig. 2.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

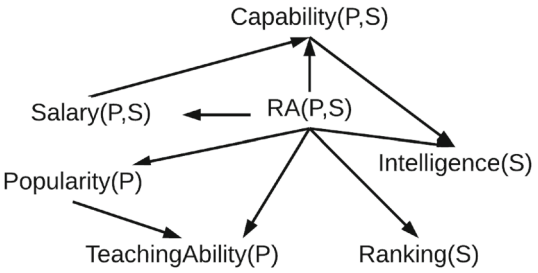

Fig. 2 Bayesian network for the university domain. We omit the *Registered* relationship for simplicity. The network was learned from the university dataset [30]

Table 2 Conditional probability table $Capability(P, S)\_CPT$, for the node $Capability(P, S)$

Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1
4	T	High	0.45
5	T	High	0.36
3	T	High	0.18
3	T	Low	0.2
2	T	Low	0.4
1	T	Low	0.4
2	T	Med	0.22
3	T	Med	0.44
1	T	Med	0.33

Only value combinations that occur in the data are shown. CP tables are stored in an RDBMS. This is an example of a factor table. The table represents a par-factor $\Phi = (\mathsf{A}, \phi)$, where $\mathsf{A} = \{Capability(P, S), RA(P, S), Salary(P, S)\}$. Each row specifies the value of the factor potential function for one factor assignment. For example, the first row shows that the potential function value for the factor assignment $\mathbf{x}_\mathsf{A} = \langle n/a, F, n/a\rangle$ is $\phi(\mathbf{x}_\mathsf{A}) = 1$

Table 3 Contingency table $Capability(P, S)\_CT$ for the node $Capability(P, S)$ and its parents, which define a par-factor set $\mathsf{A} = \{Capability(P, S), RA(P, S), Salary(P, S)\}$

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	n/a	F	n/a
5	4	T	High
4	5	T	High
2	3	T	High
1	3	T	Low
2	2	T	Low
2	1	T	Low
2	2	T	Med
4	3	T	Med
3	1	T	Med

CT tables are stored in an RDBMS. A CT table represents counts in of ground factors (sufficient statistics) in a database. For example, for the database of Table 1, the first row shows that the number of groundings for the factor assignment $\mathbf{x}_\mathsf{A} = \langle n/a, F, n/a\rangle$ is $n(\mathbf{x}_\mathsf{A}) = 203$

A par-factor in a Bayesian network is associated with a **family** of nodes [18, Sec.2.2.1]. A family of nodes comprises a child node and all of its parents. For example, in the BN of Fig. 2, one of the par-factors is associated with the par-RV set $\mathsf{A} = \{Capability(P, S), Salary(P, S), RA(P, S)\}$. For the database instance of Table 1, there are $3 \times 3 = 9$ possible ground factors associated with this par-RV, corresponding to the Cartesian product of three professors and three students. The potential function ϕ is a function from an assignment of family node values to a nonnegative real number. *In a Bayesian network, the potential function value represents the conditional probability of the child node value given its parent node values.* These conditional probabilities are typically stored in a table as shown in Table 2. This table represents therefore the potential function ϕ associated with the family par-factor. Assuming that all par-RVs have finite domains, a factor can always be represented by a **factor table** of the form Table 2: There is a column for each par-RV in the factor, each row specifies a joint assignment of values to a par-RV, and the factor column gives the value of the potential function for that assignment (cf. [18, Sec.2.2.1]).

To evaluate a joint probability $P(\mathbf{X} = \mathbf{x})$ over all ground par-RVs using Eq. 1, we must obtain the sufficient statistics: Count the number of times that each row in the CP table is instantiated in the joint assignment $\mathbf{X} = \mathbf{x}$. The sufficient statistics for the $Capability(P, S)$ family can be represented in a contingency table as shown in Table 3. For example, the first row of the contingency table indicates that the conjunction

$$Capability(P, S) = n/a, Salary(P, S) = n/a, RA(P, S) = \mathrm{F}$$

is instantiated 203 times in the university database (publicly available at [30]). This means that for 203 professor–student pairs, the professor did not employ the student as an RA (and therefore the salary and capability of this RA relationship are undefined or n/a).

Given a factor table for a par-factor, and a contingency table that represents the number of groundings for each factor assignment, it is straightforward to carry out the summation of Eq. 3. Table 4 illustrates the summation for our running example.

2.3 SRL structure learning

Algorithm 1 shows the generic format of a statistical–relational structure learning algorithm (adapted from [18]). The instantiation of procedures in lines 2, 3, 5, and 7 determines the exact behavior of a specific learning algorithm. The structure algorithm carries out a local search in the hypothesis space of graphical relational models. A set of candidates is generated based on the current model (line 3), typically using a search heuristic. For each candidate model,

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 4 To illustrate computing the log-likelihood of a database instance according to Eq. 3

CP	Count	ln(CP)* Count	Capa(P,S)	RA(P,S)	Salary(P,S)
1	203	0.00	n/a	F	n/a
0.45	5	−3.99	4	T	High
0.36	4	−4.09	5	T	High
0.18	2	−3.43	3	T	High
0.2	1	−1.61	3	T	Low
0.4	2	−1.83	2	T	Low
0.4	2	−1.83	1	T	Low
0.22	2	−3.03	2	T	Med
0.44	4	−3.28	3	T	Med
0.33	3	−3.33	1	T	Med
		$\sum -26.42$			

Each row corresponds to a factor assignment $\mathbf{x}_A$. For each factor assignment, the equation requires multiplying the par-factor instantiation count, by the logarithm of the conditional probability. For example, for the assignment $\mathbf{x}_A = \langle n/a, F, n/a \rangle$, we multiply the count $n(\mathbf{x}_A) = 203$, obtained from Table 3, by the logarithm of the conditional probability 1, obtained from Table 2. The value of this product is shown in the column ln(CP)*Count (with two digits of precision). Equation 3 sums the values of these products for each factor assignment; in this example, the resulting sum $\sum_{\mathbf{x}_A}$ equals −26.42

parameter values are estimated that maximize a model selection score function chosen by the user (line 5). A model selection score is computed for each model given the parameter values, and the best scoring candidate model is selected (line 7).

This concludes our overview of the mathematical definitions and algorithms required for learning a log-linear model. Our FACTORBASE system uses RDBMS capabilities and SQL to implement the mathematical definitions. FACTORBASE *supports model discovery for any log-linear model based on parametrized factors,* which covers the common log-linear template models used in statistical–relational learning. The remainder of the paper discusses our system design and how it supports model discovery algorithms that follow the outline of Algorithm 1.

2.4 Overview of system design

Figure 3 outlines the system components and dependencies among them. These components correspond to the mathematical definitions of Sects. 2.1 and 2.3 as follows.

1. The *variable manager* constructs the basic set of parametrized random variables, along with metadata required by the learning algorithm (e.g., the possible values that each par-RV can take). The parametrized random variables and the metadata are extracted from the schema of the input database. The definition of the par-RVs and the metadata are stored in tables in the random variable database VDB. The construction of the random variable database is described in Sect. 3 and in "Appendix."
2. The set of par-RVs and their metadata represented in the random variable database (implicitly) define the space of possible subsets of par-RVs. The *count manager* provides sufficient statistics for a list of par-RVs, counting the number of times that an assignment of values for the par-RVs is instantiated in the input database. These sufficient statistics are stored in contingency tables (cf. Table 3) in the count database CDB. The contingency tables are used to compute probabilistic predictions by evaluating Eq. 3 and to support structure learning.
3. The *model manager* stores a log-linear model using tables in the models database MDB. A graph structure specifies the sets of par-RVs that define a par-RV. In the Bayes net example of Sect. 2.2, each Bayes net family comprises a factor. The model manager stores the graph structure in MDB tables. The potential functions that specify the factor values are stored in factor tables (cf. Table 2). The model manager supports general structure learning following the blueprint of Algorithm 1. The computationally most demanding step is scoring candidate models (line 5), which requires learning parameter values. The model manager provides parameter estimates in the form of factor tables, utilizing the count manager. The model manager also provides other quantities for computing a model score, such as the number of parameters in a model and the sample size.

The following section describes the components and their implementation in detail.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

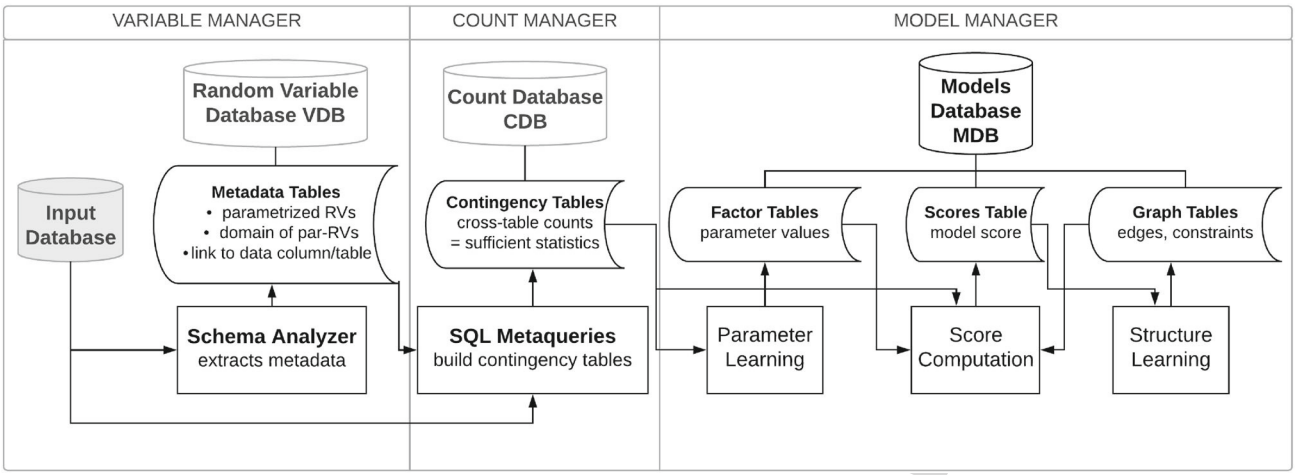

Fig. 3 FactorBase process diagram. All statistical objects are stored as first-class citizens in an RDBMS. Statistical objects are constructed and managed by three different modules. (1) The variable manager constructs the random variable database, using an SQL script that analyzes the input database schema. (2) The count manager constructs contingency tables using the metadata in the VDB. The contingency tables are constructed by count(*) queries, which themselves are constructed by SQL metaqueries. (3) The model manger constructs a relational graphical model with parameter estimates. It applies a graphical model search algorithm (specified by the user), which is supported by the metadata in the VDB, and by SQL queries against the contingency tables in the CDB

Algorithm 1: Structure learning algorithm template

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

3 The random variable database VDB

Figure 3 gives a brief summary of the random variable manager. Statistical–relational learning requires various metadata about the par-RVs in the model. These include the following.

Domain: the set of possible values of the par-RV.
Types: Pointers to the first-order variables in the par-RV.
Data Link: Pointers to the table and/or column in the input database associated with the par-RV.

The metadata must be machine readable. Following the in-database design philosophy, we store the metadata in tables so that an SRL algorithm can query it using SQL. The schema analyzer uses an SQL script that queries key constraints in the system catalog database and *automatically* converts them into metadata stored in the random variable database VDB. In contrast, existing SRL systems require users to specify information about par-RVs and associated types. Thus, FACTORBASE utilizes the data description resources of SQL to facilitate the "setup task" for relational learning [41]. We illustrate the general principles with the entity-relationship (ER) diagram of the university domain (Fig. 1).

The translation of an ER diagram into a set of functors converts each element of the diagram into a functor, except for entity sets and key fields [13]. Table 5 illustrates this translation. In terms of database tables, attribute par-RVs correspond to *columns*. Relationship par-RVs correspond to *tables*, not columns. Including a relationship par-RV in a statistical model allows the model to represent uncertainty about whether or not a relationship exists between two entities [18]. The values of descriptive attributes of relationships are undefined for entities that are not related. We represent this by introducing a new constant n/a in the domain of a relationship attribute [22]; see Table 6. Table 7 shows the schema for some of the tables that store metadata for each relationship par-RV as follows. par-RV and FO-Var are custom types.

Relationship:	The associated input data table.
Relationship_Attributes:	Descriptive attributes associated with the relationship and with the entities involved.
Relationship_FOVariables:	The first-order variables contained in each relationship par-RV.[2]

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

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 5 Translation from ER diagram into par-RVs

ER Diagram	Example	Par-RV equivalent
Entity set	Student, course	S, C
Relationship set	RA	RA(P,S)
Entity attributes	Intelligence, ranking	Intelligence(S), Ranking(S)
Relationship attributes	Capability, salary	Capability(P, S), Salary(P, S)

Table 6 Metadata about attributes represented in *VDB* database tables

Domain		AttributeColumns	
Column_Name	Value	Table_Name	Column_Name
Capability	1	Course	Diff
Capability	2	Course	Rating
Capability	3	Prof	Popularity
Capability	n/a	Prof	Teachingability
Diff	1	RA	Capability
Diff	2	RA	Salary
Grade	1	Registration	Grade
Grade	2	Registration	Sat
Grade	3	Student	Intelligence
Grade	n/a	Student	Ranking

The table *Domain* lists the domain for each functor. The table *AttributeColumns* specifies which tables and columns contain the functor values observed in the data. The column name is also the functor ID

While we have described constructing the variable database for an ER model, different structured data models can be represented by an appropriate first-order logic vocabulary [18], that is, an appropriate choice of functors. For example, in a star schema, facts can be represented in the form $f(D_1, \ldots, D_k)$, where the first-order variable D_i ranges over the primary key of dimension table i. Attributes of dimension i can be represented by a unary functor $a(D_i)$. FACTORBASE can perform structure learning for different data models after the corresponding data format has been translated into the *VDB* format.

Computational Complexity. While extracting the metadata involves a number of steps, computationally it is essentially just reformatting the primary and foreign key information, requiring only a linear pass over the database schema catalog. Finding the number of possible values for a par-RV needs only a linear pass over each database input table.

4 The count manager CDB

Figure 3 above gives a brief summary of the count manager and its connection to the other system components. The **count database** *CDB* stores a set of *contingency tables*. Contingency tables represent sufficient statistics as follows [23].

Consider a fixed list of par-RVs. A **query** is an assignment comprising $(variable = value)$ pairs, where each value is of a valid type for the variable. The **result set** of a query in a database $\mathscr{D}$ is the set of instantiations of the logical variables such that the query evaluates as true in $\mathscr{D}$. For example, in the database of Table 1, the result set for the query $RA(P, S) = \mathrm{T}$, $Capability(P, S) = 3$, $Salary(P, S) = high$ is the singleton $\{\langle jack, oliver\rangle\}$. The **count** of a query is the cardinality of its result set.

Every set of par-RVs $\mathsf{A} \equiv \{A_1, \ldots, A_n\}$ has an associated **contingency table** (CT) denoted by $CT(\mathsf{A})$. This is a table with a row for each of the possible assignments of values to the variables in $\mathbf{V}$, and a special integer column called *count*. The value of the *count* column in a row corresponding to $A_1 = x_1, \ldots, A_n = x_n$ records the count of the corresponding query. Table 3 shows a contingency table for the par-RVs $RA(P, S)$, $Capability(P, S)$, $Salary(P, S)$. The **contingency table problem** is to compute a contingency table for par-RVs A and an input database $\mathscr{D}$.

SQL Implementation With Metaqueries. We describe how the contingency table problem can be solved using SQL. This is relatively easy for a *fixed* set of par-RVs; the challenge is a general construction that works for different sets of par-RVs. For a fixed set, a contingency table can be computed by an SQL count(*) query of the form

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

FACTORBASE uses SQL itself to construct the count-conjunction query. We refer to this construction as an SQL **metaquery**. We represent a count(*) query in four kinds of tables: the Select, From, Where, and Group By tables. Each of these tables lists the entries in the corresponding count(*) query part. Given the four metaquery tables, the corresponding SQL count(*) query can be easily constructed and executed in an application to construct the contingency table. Given a list of par-RVs as input, the metaquery tables are constructed as follows from the metadata in the database *VDB*.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 7 Selected tables in the variable database schema

Table name	Column names
Relationship	RVarID: par-RV, TABLE_NAME: string
Relationship_Attributes	RVarID: par-RV, AVarID: par-RV, FO-ID: FO-Var
Relationship_FOvariables	RVarID: par-RV, FO-ID1: FO-Var, FO-ID2: FO-Var

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

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

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

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

Table 8 shows an example of a metaquery for the university database. This metaquery defines a view that in turn defines a contingency table for the random variable list associated with the relationship table *RA*. This list includes the entity attributes of professors and of students, as well as the relationship attributes of the *RA* relationship. The Bayesian network of Fig. 2 was learned from this contingency table. The contingency table defined by the metaquery of Table 8 contains only rows where the value of *RA* is true. The Möbius Virtual Join [31] can be used to extend this contingency table to include counts for when *RA* is false (as illustrated in Table 3).

Computational Complexity. Executing the views created by the metaqueries requires only a linear pass over the metadata in the random variable database, which are stored in small tables. The metaqueries do not require access to the original data. The computationally demanding part is executing the count(*) queries to build the contingency tables. The complexity of count(*) queries is well analyzed and depends on the algorithm and file organization used; for a textbook discussion, see [33, Ch.12]. We will give a highly simplified analysis, that is however sufficient to assess the cost of this FACTORBASE component relative to others, and to highlight the scalability bottlenecks. Conceptually, the count(*) operation needs to join the tables listed in the FROM clause and then compute aggregate counts for each group. In our setting, a group is defined by a list of values for a set of par-RVs. Our analysis uses the following parameters.

1. t is the number of tables to be joined.
2. r is the maximum number of rows in each table.
3. n is the number of parametrized random variables for which the contingency table is to be built.
4. d is the maximum number of values for each par-RV in the GROUP BY clause (i.e., the maximum domain size).

An efficient general algorithm for evaluating aggregate operator queries is the sort-merge join, with the following analysis [33, Ch.12]. First, sort the join tables on the join condition (WHERE clause), and then merge them with a single pass over each table. Assuming that enough memory buffer pages are available, sorting can be done in

$$O(t \times r \times \log(r))$$

time steps. Once the join tables are sorted on the join fields, finding tuples matching the join condition can be done through scanning the tables, finding matching tuples, and incrementing the count for the corresponding group. The number of scans required depends on how many matching tuples there are. A simple lower bound can be obtained by considering the output size, since at a minimum we need one time step to write out an aggregate count. If all possible combinations of par-RV values appear in the data at least once, the contingency table will contain $O(d^n)$ tuples, the size of the cross-product of par-RV domains. So the overall worst-case complexity can be estimated as

$$O(t \times r \times \log(r) + d^n).$$

The key point from this analysis is that *contingency table computation is scalable as long as the number of columns n in the contingency table is small.*

5 The model manager MDB: parameter learning

Figure 3 above gives a brief summary of the model manager and its connection to the other system components. The

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 8 Example of metaqueries and their results based on university database and the par-RV metadata (Table 7)

Metaqueries	Entries
CREATE VIEW Select_List AS	COUNT(*) AS "count"
SELE RVarID, CONCAT('COUNT(*)', 'AS "'COUNT"') AS Entries	'Popularity(P)'
FROM VDB.Relationship UNION DISTINCT	'Teachingability(P)'
SELECT RVarID, AVarID as Entries	'Intelligence(S)'
FROM VDB.Relationship_Attributes;	'Ranking(S)'
CREATE VIEW From_List AS	@database@.prof AS P
SELECT RVarID, CONCAT ('@database@.',TABLE_NAME) AS Entries	@database@.student AS S
FROM VDB.Relationship_FOvariables UNION DISTINCT	@database@.RA AS 'RA'
SELECT RVarID, CONCAT('@database@.',TABLE_NAME)	
AS Entries FROM VDB.Relationship;	
CREATE VIEW Where_List AS	'RA'.p_id = P.p_id
SELECT RVarID, CONCAT(RVarID,'.', COLUMN_NAME, '=',	'RA'.s_id = S.s_id
FO_ID, '.', REFERENCED_COLUMN_NAME) AS Entries	
FROM VDB.Relationship_FOvariables;	

model manager provides three key services for statistical–relational structure learning. In terms of Algorithm 1:

1. Estimating and storing parameter values (line 5).
2. Computing one or more model selection scores (line 7).
3. Generating, scoring, and storing candidate model structures (line 3).

FACTORBASE uses a *store+score* design for these services, which is illustrated in Fig. 3 above. A **model structure table** represents a candidate model. When a candidate model structure is inserted, a view uses the sufficient statistics from a contingency table to compute a table of parameter values. Another view uses the parameter values and sufficient statistics together to compute the score for the candidate model.

5.1 *MDB* tables for parameter learning

The relational schema for the tables that support parameter learning is shown in Table 9. The @par-RVID@ parameter refers to the ID of a par-RV, for instance $Capability(P, S)$. The model manager stores a set of factor tables (cf. Sect. 2.2). In a graphical model, each factor is defined by the local topology of the model template graph. For concreteness, we illustrate how factor tables can be represented for Bayesian networks. The graph structure can be stored straightforwardly in a database table *BayesNet* whose columns are *child* and *parent*. The table entries are the IDs of par-RVs. For each node, the *MDB* manages a conditional probability table. This is a factor table that represents the factor associated with the node's family (as is shown in Table 2). In a Bayesian network, model selection scores are decomposable. This means that there is a local score associated with each family, such that the total score for the BN model is the sum of the local scores. For each family, the local score is stored in the *Scores* table indexed by the family's child node.

5.2 Parameter learning

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

SQL Implementation With Natural Join. Given the sufficient statistics in a contingency table, a conditional probability table containing the maximum likelihood estimates can be computed by aggregation using SQL as in the example below.

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

5.3 Model score computation

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

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

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 9 Main tables in the models database *MDB*

BayesNet(child:par-RV,parent:par-RV)
@par-RVID@_CPT(@par-RVID@:par-RV, parent_1:par-RV,...,parent_k:par-RV,cp:real)
Scores(child:par-RV,loglikelihood:real,#par:int,aic:real)

For a Bayesian network, the *MDB* stores its structure, parameter estimates, and model selection scores

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

5.3.1 Parameter number computation

To determine the number of parameters of the child node @parVar-ID@, the number of possible child and parent values can be found from the *VDB.Domain* table in the random variable database.

5.3.2 Likelihood computation

As explained in Sect. 2.1, the log-likelihood can be computed by multiplying the instantiation counts of a factor by its value. Assuming that instantiation counts are represented in a contingency table and factor values in a factor table, this multiplication can be elegantly performed using the natural join operator. For instance, the log-likelihood score associated with the $Capability(P, S)$ family is given by the SQL query below.

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

The aggregate computation in this short query illustrates how well SQL constructs support complex computations with structured interrelated statistical objects.

Computational Complexity. As the SQL query shows, computing conditional probability parameter values essentially involves a natural join of a contingency table with (an aggregated version of) itself. This can be done in time

$$O(r_{ct} \times \log(r_{ct}))$$

using sort-merge (as explained in Sect. 4), where r_{ct} is the number of rows in the contingency table. Finding the number of parameter values is simply a product over the domain sizes, which are stored in the VDB metadata. As the SQL query shows, the likelihood computation involves a similar join of a contingency table with a conditional probability table. The key point is that *once contingency tables have been constructed, computing parameter values and model scores has a nearly linear cost in the size of the contingency tables.*

6 The model manager MDB: structure learning

For learning the structure of a parametrized Bayesian network, we used FACTORBASE to implement the previously existing learn-and-join algorithm (LAJ) [16,35]. The LAJ algorithm follows the general structure learning schema of Algorithm 1. It uses a sophisticated strategy for generating candidate graphs (Line 3 of Algorithm 1) that exploits the lattice nesting of relational paths. The model search strategy of the LAJ algorithm is an iterative deepening search for correlations among attributes along longer and longer chains of relationships. A similar strategy was proposed by Friedman et al. [7]. We use the LAJ algorithm as our main example because it is one of the most complex, and also one of the most effective, relational structure learning algorithms. Our discussion shows how the FACTORBASE components construct and manage the statistical objects required by the LAJ algorithm, by leveraging SQL capabilities. The previous implementation of the LAJ algorithm posted at [30] limits the par-factors so they contain *at most two* relationship par-RVs; FACTORBASE overcomes this limitation.

6.1 The learn-and-join algorithm

The algorithm takes as input a database and a lattice of relationship chains. A chain represents a path template of connected entities (a metapath in the terminology of [40]). The algorithm learns a Bayesian network for each chain in the lattice. The presence or absence of edges learned for shorter chains is propagated to longer chains. The final output is the Bayesian network associated with the longest relationship chain. Figure 4 illustrates the learning strategy for our running example. Algorithm 2 presents pseudocode; the

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

following sections discuss the different components of the algorithm in detail.

Algorithm 2: Learn-and-Join Structure Learning

Input: Database $\mathscr{D}$; parametrized random variables F; relationship chain lattice $\mathscr{L}$ with maximum chain length m.
Output: A Bayes multi-net $\mathbb{B}_{\mathbf{R}}$ for relationship chains in $\mathscr{L}$.
Calls BNL: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.

Notation: $BNL(T, Econstraints)$ is the output DAG of the Bayes net learner given data table T and edge constraints.

1: **for each** entity type E **do** {compute BN for each entity type}
2: $T_E :=$ the contingency table for the attribute nodes of E
3: $\mathbb{B}_E := BNL(T_E, \emptyset)$
4: **end for**
5: **for each** relationship node R **do** {compute BN for each relationship node}
6: Find $constraints_R$ propagated from entity BNs {Constraint 1}
7: $T_R := CT(Vars(R)$ {the contingency table for the nodes associated with R}
8: $\mathbb{B}_R := BNL(T_R, constraints_R)$
9: **end for**
10: **for** chain length $\ell \leftarrow 2, 3, \ldots, m$ **do**
11: **for** each chain $\mathbf{R}$ of length ℓ **do**
12: Find $constraints_{\mathbf{R}}$ propagated from shorter chains $\mathbf{R}^* \subset \mathbf{R}$ {Constraint 2}
13: $T_{\mathbf{R}} := CT(Vars(\mathbf{R})$ {the contingency table for the nodes associated with $\mathbf{R}$}
14: $\mathbb{B}_{\mathbf{R}} := BNL(T_{\mathbf{R}}, constraints_{\mathbf{R}})$
15: **end for**
16: **end for**

6.2 The lattice of relationship chains

We represent sets of relationship par-RVs by lists without repeating elements. Assuming an ordering of relationship par-RVs, a **relationship set** $\mathbf{R} = \{R_1(\tau_1), \ldots, R_k(\tau_k)\}$ translates into a **relationship list** $[\mathbf{R}] = [R_1(\tau_1), \ldots, R_k(\tau_k)]$. For order-independent concepts, we refer to sets rather than to lists. A relationship list $[R_1(\tau_1), \ldots, R_k(\tau_k)]$ is a **chain** if each functor $R_{i+1}(\tau_{i+1})$ shares at least one population variable with the preceding terms $R_1(\tau_1), \ldots, R_i(\tau_i)$.[3] In the following, we use the set notation $\mathbf{R}$ for both chains and the associated relationship set. For instance, in the university schema of Fig. 1, a relationship chain of length 2 is the list

$$[RA(P, S), Registered(S, C)]. \quad (4)$$

A three-element chain is

$$[RA(P, S), Registered(S, C), TA(C, S)]. \quad (5)$$

[3] Essentially, the same concept is called a slot chain in PRM modeling [9].

A relationship chain $\mathbf{R}$ is a **subchain** of another chain $\mathbf{R}'$, written $\mathbf{R} \sqsubseteq \mathbf{R}'$, if every relationship par-RV in $\mathbf{R}$ occurs also in $\mathbf{R}'$. For example, the chain (4) is a subchain of the chain (5). Two chains are equivalent in case they contain the same relationship variables. The **relationship lattice** contains a representative chain from each equivalence class. A representative chain for a set of relationship variables can be generated using any fixed order on relationship variables. In the following, we do not distinguish between a relationship chain and its equivalence class unless there is risk of confusion. The subchain relation $\sqsubseteq$ defines a lattice on (equivalence classes of) relationship chains. Figure 5 illustrates the lattice for the relationship nodes in the university schema of Fig. 2. For reasons that we explain below, entity tables are also included in the lattice and linked to relationships that involve the entity in question.

6.3 The Bayesian multinet

Each chain in the lattice corresponds to a subset $\mathbf{R}$ of relationship variables. Associated with the chain is a set of par-RVs $Vars(\mathbf{R})$, comprising the following:

- All relationship par-RVs in $\mathbf{R}$.
- Each attribute par-RV associated with a relationship par-RV in $\mathbf{R}$.

For each chain $\mathbf{R}$, the learn-and-join algorithm learns a Bayesian network $\mathbb{B}_{\mathbf{R}}$ whose nodes comprise the set $Vars(\mathbf{R})$. This network is learned from the contingency table $CT(Vars(\mathbf{R}))$.

The lattice structure defines a *multinet* rather than a single Bayes net. Multinets are a classic Bayes net formalism for modeling context-sensitive dependencies among variables. Geiger and Heckerman contributed a standard reference article for the multinet formalism [8]. In the learn-and-join algorithm, the context of a multinet is defined by a chain of relationship functor nodes. Distinguishing these different contexts allows us to represent that the existence of certain dependencies among attributes of entities depends on which kind of links exists between the entities. The final output of the learn-and-join algorithm is a single Bayes net derived from the multinet.

6.4 Edge inheritance in the relationship lattice

These constraints state that the presence or absence of edges in graphs associated with join tables lower in the lattice is inherited by graphs associated with join tables higher in the lattice. The motivation for these constraints is that dependencies should be assessed in the most specific context possible. The first constraint states that edges from an entity table are

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

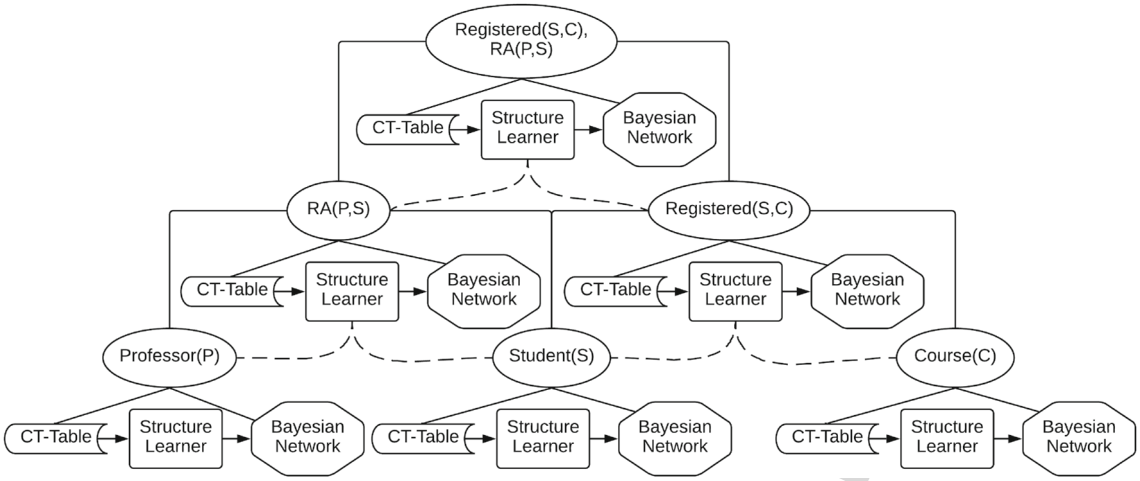

Fig. 4 Overview of the learn-and-join hierarchical structure learning method. The hierarchy is shown for two relationships, *Registered* and *RA*. For each relationship chain, SQL metaqueries compute a contingency table. Solid block lines: The inclusion relations among points in the relationship chain lattice. Solid arrows: for each relationship chain, a single-table Bayesian network learner constructs a Bayesian network, given the contingency table for each relationship chain. Dashed lines: The absence and presence of Bayesian network edges learned for a shorter relationship chain are propagated as constraints for learning for a longer relationship chain

inherited by relationship tables that involve the entity in question.

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

The second constraint states that edges learned on smaller relationship sets are inherited by larger relationship sets. If the smaller sets are ambiguous with regard to the direction of an adjacency, the larger relationship set must contain the adjacency; the direction is then resolved by applying Bayes net learning to the larger relationship set. We write an expression such as $f(\tau)$ to denote a parametrized random variable with functor f and first-order variables τ.

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

1. *If $f(\tau)$ and $g(\tau')$ are not adjacent in any DAG $\mathbb{B}_{\mathbf{R}^*}$ associated with a relationship subset $\mathbf{R}^* \subset \mathbf{R}$, then $f(\tau)$ and $g(\tau')$ are not adjacent in the graph associated with the relationship set $\mathbf{R}$.*
2. *Else if all subset graphs agree on the orientation of the adjacency $f(\tau) - g(\tau')$, the graph associated with the relationship set $\mathbf{R}$ inherits this orientation.*
3. *Else the graph associated with the relationship set $\mathbf{R}$ must contain the edge $f(\tau) \rightarrow g(\tau')$ or the edge $f(\tau) \leftarrow g(\tau')$.*

Examples Figure 5 presents a trace of the LAJ algorithm for part of our running example.

Constraint 1. The Bayes net for the entity type *Professor* contains the edge $Popularity(P) \rightarrow Teaching\_ability(P)$. Therefore, the length 1 chain $RA(P, S)$ is *required* to contain this edge as well. The Bayes net graph for the entity type *Course* does not contain the edge $Difficulty(C) \rightarrow Level(C)$, so the Bayes net for the length 1 chain $Registered(S, C)$ must *not* contain the edge $Difficulty(C) \rightarrow Level(C)$.

Constraint 2. The Bayes net for the length 1 chain $RA(P, S)$ contains an edge $Salary(P, S) \rightarrow Capability(P, S)$ and does not contain $Popularity(P) \rightarrow Salary(P, S)$. So for the length 2 chain $Registered(S, C), RA(P, S)$, the edge $Salary(P, S) \rightarrow Capability(P, S)$ is required. The edge $Difficulty(C) \rightarrow Level(C)$ is forbidden. We next discuss how FACTORBASE leverages SQL to implement the LAJ algorithm.

6.5 SQL implementation

Table 10 shows the main tables that support Bayesian network structure learning. These comprise two groups: Lattice tables related to relationship chains and graph tables related to edges among par-RVs.

6.5.1 Lattice tables

The lattice tables support SQL access to the internal structure of a relationship chain: The *LatticeMember* table lists, for each valid relationship chain, the relationship nodes that

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

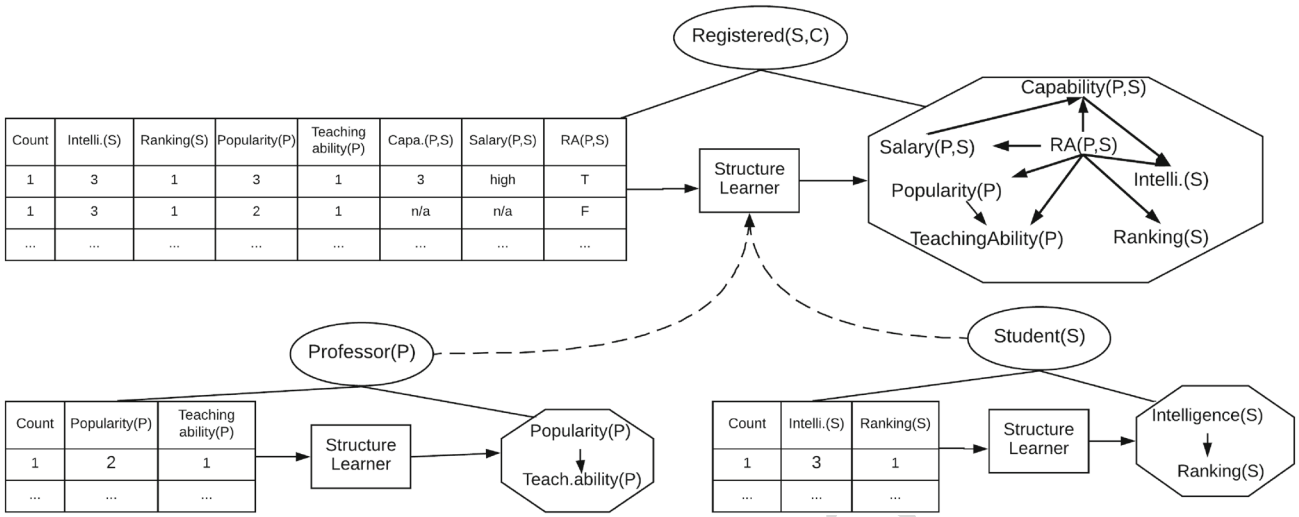

Fig. 5 Trace of the learn-and-join hierarchical structure learning method of Fig. 4 for the university domain. The trace is shown for the *RA* relationship only. For each relationship chain, the figure shows the complete Bayesian network structures learned and excerpts from the contingency tables. Notice that the edges learned for the Professor table and for the Student table are propagated to the Bayesian network for the Registered table

Table 10 Tables supporting structure learning

LatticeMember(Member:par-RV, Chain: rchain)
LatticeOrder(Subchain:rchain,SuperChain:rchain)
ChainBayesNets(Chain:rchain,child:par-RV,parent:par-RV)
RequiredEdges(Chain:rchain,child:par-RV,parent:par-RV)
ForbiddenEdges(Chain:rchain,child:par-RV,parent:par-RV)

Table 11 Lattice tables

Member	Chain
LatticeMember	
$RA(P, S)$	$[RA(P, S), Registered(S, C)]$
$Registered(S, C)$	$[RA(P, S), Registered(S, C)]$
Subchain	**SuperChain**
LatticeOrder	
$RA(P, S)$	$[RA(P, S), Registered(S, C)]$
$Registered(S, C)$	$[RA(P, S), Registered(S, C)]$

Table 12 Tabular representation of the Bayesian multi-net from Fig. 4

Rchain	Child	Parent
ChainBayesNets		
RA(P,S)	Capability(P,S)	*RA(P,S)*
RA(P,S)	Intelligence(S)	*RA(P,S)*
RA(P,S)	Popularity(P)	*RA(P,S)*
RA(P,S)	Popularity(P)	Teachingability(P)
RA(P,S)	Ranking(S)	Intelligence(S)
RA(P,S)	Salary(P,S)	*RA(P,S)*
RA(P,S)	Salary(P,S)	Capability(P,S)
FOVariable	**Child**	**Parent**
EntityBayesNets		
Prof	Popularity(P)	Teachingability(P)
Student	Ranking(S)	Intelligence(S)

Top: edges learned for the relationship chain RA(P,S). Bottom: edges learned for first-order variables Prof and Student. Bayes net learning found no edges (correlations) for attributes of courses. *P* ranges over professors, *S* over students

are members of this chain. The *LatticeOrder* table lists for each relationship chain, its immediate subchain. Table 11 shows the lattice tables for our running example. Concatenating the IDs for a relationship par-RV defines an ID for a relationship chain. The relationship par-RVs are listed in the variable database.

We generate IDs for valid relationship chains using an application language outside of SQL. (Java in our system). The space of possible relationship chains can be constructed to reflect domain knowledge. In our experiments, we follow the suggested default for the LAJ algorithm [35]: include all relationship chains, of any length, that contain at most three first-order variables.

6.5.2 Graph tables

For each relationship chain, a Bayesian network is stored using the tabular representation described in Sect. 5. Table 12 illustrates this representation in our running example.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 13 Required and forbidden edges for the relationship chain RA(P,S)

Rchain	Child	Parent
RequiredEdges		
RA(P,S)	Popularity(P)	Teachingability(P)
RA(P,S)	Ranking(S)	Intelligence(S)
Reg.(S,C), RA(P,S)	Capability(P,S)	Salary(P,S)
Reg.(S,C), RA(P,S)	Popularity(P)	Teachingability(P)
Reg.(S,C), RA(P,S)	Popularity(P)	Teachingability(P)
ForbiddenEdges		
Reg.(S,C)	Level(C)	Difficulty(C)
Reg.(S,C)	Difficulty(C)	Level(C)
Reg.(S,C), RA(P,S)	Capability(P,S)	Salary(P,S)
Reg.(S,C), RA(P,S)	Level(C)	Difficulty(C)
Reg.(S,C), RA(P,S)	Difficulty(C)	Level(C)

Horizontal lines separate constraints for different relationship chains. *P* ranges over professors, *S* over students, *C* over courses

```
CREATE VIEW RequiredEdges AS
SELECT DISTINCT
        LatticeOrder.Superchain AS Rchain,
        ChainBayesNets.child AS child,
        ChainBayesNets.parent AS parent
FROM
        ChainBayesNets,
        LatticeOrder
WHERE
       LatticeOrder.Subchain =
       ChainBayesNets.Rchain
UNION
       SELECT DISTINCT
        RNodes_pvars.rnid AS Rchain,
        Entity_BayesNets.child AS child,
        Entity_BayesNets.parent AS parent
        FROM
        RNodes_pvars, Entity_BayesNets
        WHERE
        RNodes_pvars.pvid = Entity_BayesNets.pvid
```

Fig. 6 SQL for a view that computes required edges from learned edges stored in *ChainBayesNets*

For each relationship node (e.g., RA(P,S)), the *Required Edges* are the edges learned for the associated FOVariables (contained in the *EntityBayesNets* table). For each relationship chain, the required edges are the union of learned edges for shorter chains. Similarly for *Forbidden Edges*. Required and forbidden edges are exported as constraints for Bayesian network learning. Table 13 illustrates this representation.

The RequiredEdges table is implemented as a view shown in Fig. 6. The view mechanism automatically adds new required edges when new learned edges are added to the ChainBayesNets table. There is a similar view (not shown) that adds forbidden edges, based on which edges are *not* added to the ChainBayesNets table.

Computational Complexity. We have already discussed the complexity of scoring a candidate model in Sect. 5.3.2. We now consider how many candidate models are typically generated during a local search. This topic is extensively analyzed in the literature on graphical model learning [1]. As in previous sections, we present a simplified analysis that focuses on comparing model generation to other aspects of FACTORBASE. Typical local search algorithms consider only local operations such as adding or deleting an edge at a time from the current Bayesian network. The GES algorithm that we employ in our experiments starts with the empty graph. In its forward stage, it considers only adding edges. In its second backward stage, it considers only deleting edges. Given n nodes (par-RVs), this means that at each step, the search considers $\binom{n}{2}$ edge additions/deletions. The maximum number of steps in each phase is $\binom{n}{2}$, so the overall number of candidates generated is $O(\binom{n}{2}^2)$. So a loose worst-case bound is $O(n^4)$. In practice, the number of steps in each phase is much smaller, on the order of the number of nodes [35], and the observed complexity is closer to

$$O\left(n \times \binom{n}{2}\right).$$

The key point is that *model structure algorithm scales well in the number of parametrized random variables.* Even for non-relational data represented in a single data table, the cost of computing sufficient statistics dominates model search [23]. This is even more the case for relational data where sufficient statistics require table to join with their potentially exponential cost.

This completes our description of how the modules of FACTORBASE are implemented using SQL. We next show how these modules support a key learning task: computing the predictions of an SRL model on a test instance.

7 Test set predictions

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

- Classifying the test instance, which is one of the main applications of a machine learning system for end users.
- Comparing the class labels predicted against true class labels is a key step in several approaches to model scoring [18].
- Evaluating the accuracy of a machine learning algorithm by the train-and-test paradigm, where the system is provided a training set for learning and then we test its predictions on unseen test cases.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 14 SQL queries for computing target contingency tables supporting test set prediction

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

<Attribute-List> and <Key-Equality-List> are as in Table 8

Table 15 Target contingency tables for target = jack and for target = jill

Sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
jack_Capability_(P,S)_CT				
Jack	5	N/A	N/A	F
Jack	5	4	High	T
…	…	…	…	…
jill_Capability_(P,S)_CT				
Jill	3	N/A	N/A	F
Jill	7	4	High	T
…	…	…	…	…

We first discuss how to compute a prediction for a single test case and then how to compute an overall prediction score for a set of test cases. Class probabilities can be derived from Eq. 1 as follows [18, Sec.2.2.2]. Let Y denote a ground par-RV to be classified, which we refer to as the **target variable**. For example, a ground atom may be $Intelligence(jack)$. In this example, we refer to jack as the **target entity**. Write $\mathbf{x}_{-Y}$ for a database instance that specifies the values of all ground par-RVs, except for the target, which are used to predict the target node. Let $[\mathbf{x}_{-Y}, y]$ denote the completed database instance where the target node is assigned value y. The log-linear model uses the likelihood $P([\mathbf{x}_{-Y}, y])$ as the joint score of the label and the predictive features. The conditional probability is proportional to this score:

$$P(y|\mathbf{x}_{-\mathbf{Y}}) \propto P([\mathbf{x}_{-Y}, y]) \quad (6)$$

where the joint distribution on the right-hand side is defined by Eq. 1, and the scores of the possible class labels need to be normalized to define conditional probabilities.

SQL Implementation. The obvious approach to computing the log-linear score would be to use the likelihood computation of Sect. 5.3 for the entire database. This is inefficient because only instance counts that involve the target entity change the classification probability. This means that we need only consider query instantiations that match the appropriate logical variable with the target entity (e.g., $S = jack$).

Assuming that for each node with ID @parRVID@, a target contingency table named $CDB.target\_@parRVID@\_CT$ has been built in the count database CDB, the log-likelihood SQL is as in Sect. 5.3. For instance, the contribution of the $Capability(P, S)$ family is computed by the SQL query shown, but with the contingency table jack_Capability(P,S)_CT in place of Capability(P,S)_CT. The new problem is finding the target contingency table. SQL allows us to solve this easily by restricting counts to the target entity in the WHERE clause. To illustrate, suppose we want to modify the contingency table query of Table 8 to compute the contingency table for $S = jack$. We add the student id to the SELECT clause and the join condition $S.s\_id = jack$ to the WHERE clause; see Table 14. The FROM clause is the same as in Table 8. The metaquery of Table 8 is easily changed to produce these SELECT and WHERE clauses.

Next, consider a setting where a model is to be scored against an entire test set. For concreteness, suppose the problem is to predict the intelligence of a set of students $Intelligence(jack)$, $Intelligence(jill)$, $Intelligence(student_3)$, $\ldots, Intelligence(student_m)$. An obvious approach is to loop through the set of test instances, repeating the likelihood query above for each single instance. Instead, SQL supports *block access* where we process the test instances as a block. Intuitively, instead of building a contingency table for each test instance, we build a single contingency table that stacks together the individual contingency tables (Table 15). Blocked access can be implemented in a beautifully simple manner in SQL: We simply add the primary key id field for the target entity to the GROUP BY list; see Table 14.

8 Evaluation

Our experimental study describes how FACTORBASE can be used to implement a challenging machine learning application: constructing a Bayesian network model for a relational database. Bayesian networks are a good illustration of typical challenges and how RDBMS capabilities can address them because: (1) Bayesian networks are widely regarded as a very useful model in machine learning and AI, which supports decision making and reasoning under uncertainty. At the same time, they are considered challenging to learn from data. (2) Database researchers have proposed Bayesian networks for combining databases with uncertainty [42]. (3) A Bayesian network with par-RVs can be easily converted into other first-order relational models, such as a Markov Logic Network; see [4,36].

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 16 Datasets characteristics

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

#Tuples = total number of tuples over all tables in the dataset

We describe the system and the datasets we used. Code was written in MySQL Script and Java, JRE 1.7.0. and executed with 8GB of RAM and a single Intel Core 2 QUAD Processor Q6700 with a clock speed of 2.66GHz (no hyper-threading). The operating system was Linux Centos 2.6.32. The MySQL Server version 5.5.34 was run with 8GB of RAM and a single core processor of 2.2GHz. All code and datasets are available online [30].

8.1 Datasets

We used six benchmark real-world databases. For detailed descriptions and the sources of the databases, please see [30] and the references therein. Table 16 summarizes basic information about the benchmark datasets. IMDb is the largest dataset in terms of number of total tuples (more than 1.3 M tuples) and schema complexity. It combines the MovieLens database[4] with data from the Internet Movie Database (IMDb)[5] following [27].

Table 16 provides information about the number of par-RVs generated for each database. More complex schemas generate more random variables.

8.2 Bayesian network learning

We applied to each dataset our new SQL-based implementation of the LAJ algorithm described in Sect. 6.

A major design decision is how to make sufficient statistics available to the LAJ algorithm. In our experiments, we followed a *pre-counting* approach where the count manager constructs a **joint contingency table** for *all* par-RVs in the random variable database. An alternative would be *on-demand* counting, which computes many contingency tables, but only for factors that are constructed during the model search [21]. Pre-counting is a form of data preprocessing: Once the joint contingency table is constructed, local contingency tables can be built quickly by summing (Group By). Different structure learning algorithms can therefore be run quickly on the same joint contingency table. For our evaluation, pre-counting has several advantages.

- Constructing the joint contingency table presents a maximally challenging task for the count manager.
- Separating counting/data access from model search allows us to assess separately the resources required for each task.

Limitations of pre-counting. Although a pre-counting approach has advantages and is suitable for evaluating FACTORBASE, it presents a scalability bottleneck. As our analysis showed above, the worst-case complexity of computing a contingency table scales as $O(d^n)$, the possible number of sufficient statistics. Therefore, pre-counting does not scale to larger numbers of par-RVs n. In database terms, it does not scale to databases with many columns. On-demand counting may help; a rough complexity analysis will illustrate the payoffs. The motivation for on-demand counting is the observation that Bayes net learning typically introduces a small number of parents only that can be treated as a constant k (Lv et al. [21] suggest that k is typically 4). In the worst case, on-demand counting needs to construct $n \times \binom{n}{k}$ different contingency tables, one for each node and for each possible parent set of size k. Each of these contingency tables contains at most d^{k+1} sufficient statistics (rows), so the overall number of sufficient statistics that need to be managed is bounded by

$$O\left(n \times \binom{n}{k} \times d^{k+1}\right),$$

compared to $O(d^n)$ for the pre-counting approach. Thus, on-demand counting computes a polynomial number of small contingency tables, whereas pre-counting computes a single exponential-size contingency table.

8.3 Results

Table 17 reports the number of sufficient statistics for constructing the joint contingency table. This number depends mainly on the number of par-RVs. The number of sufficient statistics can be quite large, over 15M for the largest dataset IMDb. Even with such large numbers, constructing contingency tables using the SQL metaqueries is feasible, taking just over 2 hours for the very large IMDb set. The number of Bayesian network parameters is much smaller than the number of sufficient statistics. The difference between the number of parameters and the number of sufficient statistics measures how compactly the BN summarizes the statistical information in the data. Table 17 shows that Bayesian

[4] www.grouplens.org, 1M version.

[5] www.imdb.com, July 2013.

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 17 Count manager: sufficient statistics and parameters

Dataset	# Database tuples	# Sufficient statistics (SS)	SS computing time (s)	#BN parameters
Movielens	1,010,051	252	2.7	292
Mutagenesis	14,540	1631	1.67	721
UW-CSE	712	2828	3.84	241
Mondial	870	1,746,870	1112.84	339
Hepatitis	12,927	12,374,892	3536.76	569
IMDb	1,354,134	15,538,430	7467.85	60,059

Table 18 Model manager evaluation

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

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

Table 18 shows that the graph structure of a Bayesian network contains a small number of edges relative to the number of parameters. The model manager provides fast maximum likelihood estimates for a given structure. This is because computing a local contingency table for a BN family is fast given the joint contingency table.

Figure 7 compares computing predictions on a test set using an instance-by-instance loop, with a separate SQL query for each instance, vs. a single SQL query for all test instances as a block (illustrated in Table 14). Table 19 specifies the number of test instances for each dataset. We split each benchmark database into 80% training data, 20% test data. The test instances are the ground atoms of all descriptive attributes of entities. The blocked access method is 10–100 faster depending on the dataset. The single access method did not scale to the large IMDb dataset (time-out after 12 hours).

Table 20 reports result for the complete learning of a Bayesian network, structure, and parameters. It benchmarks FACTORBASE against functional gradient boosting, a state-of-the-art multi-relational learning approach [24]. Functional gradient boosting learns models with highly competitive predictive accuracy, and it is the only structure learning system that scales to the relatively large datasets we use in this study (besides the LAJ method). MLN_Boost learns a Markov Logic Network and RDN_Boost a Relational Dependency Network. We used the Boostr implementation [17]. To make

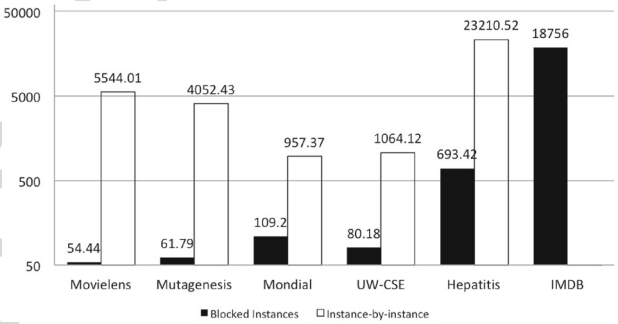

Fig. 7 Times (s) for computing predictions on test instances. The right white column shows the time for looping over single instances using the single access query of Table 14. The left black column shows the time for the blocked access query of Table 14

the results easier to compare across databases and systems, we divide the total running time by the number of par-RVs for the database (Table 16). Table 20 shows the total runtimes for the different methods. FB-Total is the runtime for our FACTORBASE implementation of the LAJ algorithm. We separately report the cost of pre-computing the large contingency table in the last FB-Count column. The table shows that structure learning with FACTORBASE is fast: Even the large complex database IMDb requires only around 8 minutes/par-RV. Compared to the boosting methods, FACTORBASE shows excellent scalability: Neither boosting method terminates on the IMDb database, and while RDN_Boost terminates on the MovieLens database, it is almost 5,000 times slower than FACTORBASE. Much of the speed of our implementation is due to quick access to sufficient statistics. As the last FB-Count column of Table 20 shows, on the larger datasets FACTORBASE spends about 80% of computation time on

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 19 # of test instances

Dataset	Movielens	Mutagenesis	UW-CSE	Mondial	Hepatitis	IMDb
#instance	4742	3119	576	505	2376	46,275

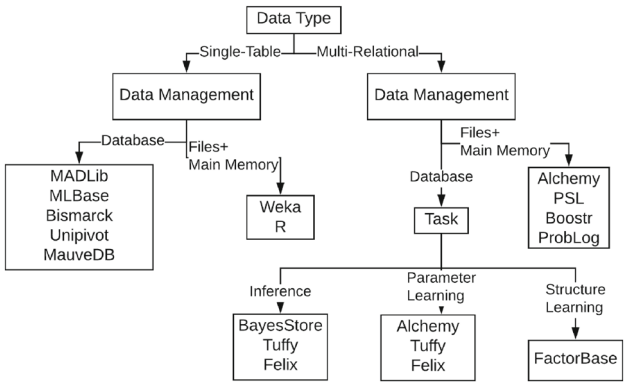

Fig. 8 A tree structure for related work in the design space of machine learning × data management

gathering sufficient statistics via the count manager. This suggests that a large speedup for the boosting algorithms could be achieved if they used the FACTORBASE in-database design for managing sufficient statistics.

We do not report detailed accuracy results because predictive accuracy is not the focus of this paper. On the standard conditional log-likelihood metric, as defined by Eq. 6, the model learned by FACTORBASE performs better than the boosting methods on all databases. This is consistent with the results of previous studies [35].

Conclusion. FACTORBASE leverages RDBMS capabilities for scalable management of statistical analysis objects. It efficiently constructs and stores large numbers of sufficient statistics and parameter estimates. The RDBMS support for statistical–relational learning translates into orders of magnitude improvements in speed and scalability.

9 Related work

The design space for combining machine learning with data management systems offers a number of possibilities, several of which have been explored in previous and ongoing research. We selectively review the work most relevant to our research. Figure 8 provides a tree structure for the research landscape.

9.1 Single-table machine learning

Most machine learning systems, such as Weka or R, support learning from a single table or data matrix only. The single-table representation is appropriate when the data points represent a homogeneous class of entities with similar attributes, where the attributes of one entity are independent of those of others [18]. The only way a single-table system can be applied to multi-relational data is after an extract-transform-load preprocessing step where multiple interrelated tables are converted into a single data table. When the learning task is classification, such preprocessing is often called propositionalization [18]. This "flattening" of the relational structure typically involves a loss of information. The ClowdFlows system [20] allows a user to specify a MySQL database as a data source and then convert the MySQL data to a single-table representation using propositionalization.

9.1.1 RDBMS learning

Leveraging RDBMS capabilities through SQL programming is the unifying idea of the recent MADLib framework [14]. An advantage of the MADLib approach that is shared by FACTORBASE is that in-database processing avoids exporting the data from the input database. The Apache Spark [2] framework includes MLBase and SparkSQL that provide support for distributed processing, SQL, and automatic refinement of machine learning algorithms and models [19]. Other RDBMS applications include gathering sufficient statistics [12] and convex optimization [6]. The MauveDB system [3] emphasizes the importance of several RDBMS features for combining statistical analysis with databases. As in FACTORBASE, this includes storing models and associated parameters as objects in their own right, and using the view mechanism to update statistical objects as the data change. A difference is that MauveDB presents model-based views of the *data* to the user, whereas FACTORBASE presents views of the *models* to machine learning applications.

9.1.2 RDBMS inference

Wong et al. [44] applied SQL operators such as the natural join to perform log-linear inference with a single-table graphical model stored in an RDBMS. Monte Carlo methods have also been implemented with an RDBMS to perform inference with uncertain data [15,43]. The MCDB system [15] stores parameters in database tables like FACTORBASE.

9.2 Multi-relational learning

For overviews of multi-relational learning, please see [4,10, 18]. Most implemented systems, such as Aleph and Alchemy,

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Table 20 Learning time comparison (s) with other statistical–relational learning systems

Dataset	RDN_Boost	MLN_Boost	FB-total	FB-count
MovieLens	5562	N/T	1.12	0.39
Mutagenesis	118	49	1	0.15
UW-CSE	15	19	1	0.27
Mondial	27	42	102	61.82
Hepatitis	251	230	286	186.15
IMDb	N/T	N/T	524.25	439.29

NT non-termination

use a logic-based representation of data derived from Prolog facts that originated in the inductive logic programming community [5].

9.2.1 RDBMS learning

Singh and Graepel [39] present an algorithm that analyzes the relational database system catalog to generate a set of nodes and a Bayesian network structure. This approach utilizes SQL constructs as a data description language in a way that is similar to our Schema Analyzer. Differences include the following. (1) The Bayesian network structure is fixed and based on latent variables, rather than learned for observable variables only, as in our case study. (2) The RDBMS is not used to support learning after random variables have been extracted from the schema.

Qian et al. [31] discuss work related to the contingency table problem and introduce contingency table algebra. Their paper focuses on a virtual join algorithm for computing sufficient statistics that involve negated relationships. They do not discuss integrating contingency tables with other structured objects for multi-relational learning.

Quakkelaar [32] builds on the FACTORBASE system by leveraging database technology such as dynamic views to increase the efficiency of computing contingency tables and parameter values.

9.2.2 RDBMS inference

Database researchers have developed powerful probabilistic inference algorithms for multi-relational models. The BayesStore system [42] introduced the principle of treating all statistical objects as first-class citizens in a relational database as FACTORBASE does. The Tuffy system [25] achieves highly reliable and scalable inference for Markov Logic Networks (MLNs) with an RDBMS. It leverages inference capabilities to perform MLN parameter learning. RDBMS support for local search parameter estimation procedures, rather than closed-form maximum likelihood estimation as in our case study, has also been explored [6,25,26].

10 Conclusion and future work

Compared to traditional learning with a single data table, learning for multi-relational data requires new system capabilities. In this paper, we described FACTORBASE, a system that leverages the existing capabilities of an SQL-based RDBMS to support statistical–relational learning. FACTORBASE supports model discovery for any log-linear model based on parametrized factors, which covers the common log-linear template models used in statistical–relational learning. Representational tasks include specifying metadata about structured parametrized random variables and storing the structure of a learned model. Computational tasks include storing and constructing sufficient statistics, and computing parameter estimates and model selection scores. We showed that SQL scripts can be used to implement these capabilities, with multiple advantages. These advantages include:

- Fast program development through high-level SQL constructs for complex table and count operations.
- Managing large and complex statistical objects that are too big to fit in main memory. For instance, some of our benchmark databases require storing and querying millions of sufficient statistics.

While FACTORBASE provides good solutions for each of these system capabilities in isolation, the ease with which large complex statistical–relational objects can be integrated via SQL queries is a key feature. Empirical evaluation on six benchmark databases showed significant scalability advantages from utilizing the RDBMS capabilities: Both structure and parameter learning scaled well to millions of data records, beyond what previous multi-relational learning systems can achieve.

Future Work. FACTORBASE opens a number of avenues for future work, such as exploring alternative designs with different trade-offs and developing challenging applications.

Alternative Designs. The scalability bottleneck in our current system is the number of parametrized random variables (columns in the input database). This limitation is not due to an inherent limitation of FACTORBASE but arises because we

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

used FACTORBASE with a pre-counting design that requires computing an exponential-size contingency table before model search. An alternative would be *on-demand* counting [21], which computes polynomially many small contingency tables. On-demand counting raises the issue of how to construct and manage many contingency tables and how to store and cache them to be reused as much as possible. Extending our in-database design to many contingency tables offers a promising approach to this challenge.

Our current implementation leverages an RDBMS to store statistical objects on disk, which minimizes the amount of main memory required, but incurs latency through disk accesses. The Apache Spark [2] framework supports machine learning that leverages a main memory cluster (e.g., MLBase). Spark was designed to be SQL friendly (e.g., SparkSQL), which makes it very suitable for our SQL-based approach. For example, the SQL queries and scripts we have described can be used, with minimal modifications, to create suitable Spark dataframes that represent factor tables, graph structures, model scores, etc. The portability of SQL to different system environments like Spark is one of the advantages of FACTORBASE.

Applications. Further potential application areas for FACTORBASE include managing massive numbers of aggregate features for classification [29] and collective matrix factorization [38,39]. There are opportunities for optimizing RDBMS operations for the workloads required by statistical–relational structure learning. These include view materialization and the key scalability bottleneck of computing multi-relational sufficient statistics. NoSQL databases can exploit a flexible data representation for scaling to very large datasets. However, SRL requires count operations for random complex join queries, which is a challenge for less structured data representations. An important goal is a single RDBMS package for both learning and inference that integrates FACTORBASE with inference systems such as BayesStore and Tuffy.

Acknowledgements This research was supported by a Discovery Grant to Oliver Schulte by the Natural Sciences and Engineering Research Council of Canada. Zhensong Qian was supported by a grant from the China Scholarship Council. We are indebted to anonymous reviewers for the Journal of Data Science and Analytics for helpful comments that improved the paper presentation substantially.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

A Appendix: The random variable database layout

We provide details about the Schema Analyzer. A complete SQL script that implements the Schema Analyzer is available [37]. Table 21 shows the relational schema of the Random Variable Database. Figure 9 shows dependencies between the tables of this schema.

Table 21 Schema for random variable database

Table name	Column names
AttributeColumns	TABLE_NAME, COLUMN_NAME
Domain	COLUMN_NAME, VALUE
Pvariables	Pvid, TABLE_NAME
1Variables	1VarID, COLUMN_NAME, Pvid
2Variables	2VarID, COLUMN_NAME, Pvid1, Pvid2, TABLE_NAME
Relationship	RVarID, TABLE_NAME, Pvid1, Pvid2, COLUMN_NAME1, COLUMN_NAME2

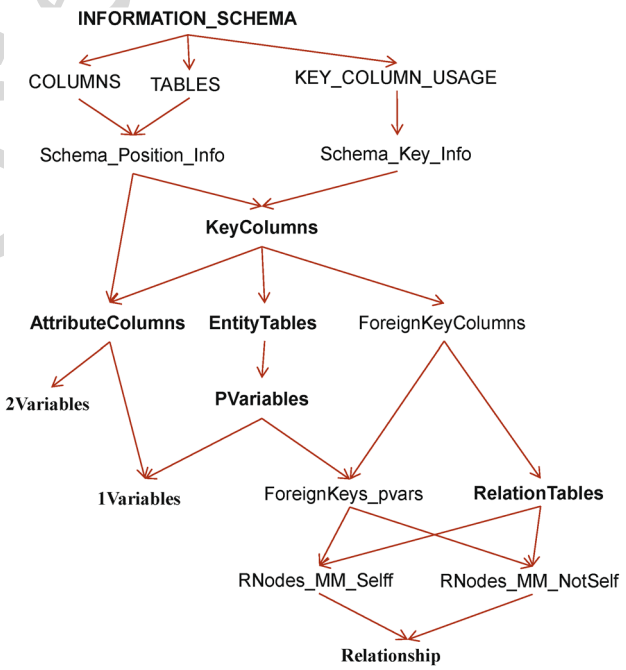

Fig. 9 Table dependencies in the random variable database *VDB*

References

1. Chickering, D.: Optimal structure identification with greedy search. J. Mach. Learn. Res. **3**, 507–554 (2003)
2. Contributors, A.S.P.: Apache Spark. http://spark.apache.org/
3. Deshpande, A., Madden, S.: MauveDB: supporting model-based user views in database systems. In: SIGMOD, pp. 73–84. ACM (2006)
4. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Morgan and Claypool Publishers, San Rafael (2009)
5. Dzeroski, S., Lavrac, N. (eds.): Relational Data Mining. Springer, Berlin (2001)
6. Feng, X., Kumar, A., Recht, B., Ré, C.: Towards a unified architecture for in-RDBMS analytics. In: SIGMOD Conference, pp. 325–336 (2012)
7. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: IJCAI, pp. 1300–1309. Springer (1999)
8. Geiger, D., Heckerman, D.: Knowledge representation and inference in similarity networks and Bayesian multinets. Artif. Intell. **82**(1–2), 45–74 (1996)
9. Getoor, L., Friedman, N., Koller, D., Pfeffer, A., Taskar, B.: Probabilistic relational models. In: Introduction to Statistical Relational Learning [10], chap. 5, pp. 129–173
10. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
11. Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. ACM SIGMOD Rec. **30**(2), 461–472 (2001)
12. Graefe, G., Fayyad, U.M., Chaudhuri, S.: On the efficient gathering of sufficient statistics for classification from large SQL databases. In: KDD, pp. 204–208 (1998)
13. Heckerman, D., Meek, C., Koller, D.: Probabilistic entity-relationship models, PRMs, and plate models. In: Getoor and Taskar [10]
14. Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., Kumar, A.: The MADlib analytics library: Or MAD skills, the SQL. PVLDB **5**(12), 1700–1711 (2012)
15. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a Monte Carlo approach to managing uncertain data. In: SIGMOD Conference, pp. 687–700 (2008)
16. Khosravi, H., Schulte, O., Man, T., Xu, X., Bina, B.: Structure learning for Markov logic networks with many descriptive attributes. In: AAAI, pp. 487–493 (2010)
17. Khot, T., Shavlik, J., Natarajan, S.: Boostr. http://pages.cs.wisc.edu/~tushar/Boostr/
18. Kimmig, A., Mihalkova, L., Getoor, L.: Lifted graphical models: a survey. Mach. Learn. **99**(1), 1–45 (2015). https://doi.org/10.1007/s10994-014-5443-2
19. Kraska, T., Talwalkar, A., Duchi, J.C., Griffith, R., Franklin, M.J., Jordan, M.I.: MLbase: a distributed machine-learning system. In: CIDR (2013)
20. Lavravc, N., Perovvsek, M., Vavpetivc, A.: Propositionalization online. In: ECML, pp. 456–459. Springer (2014)
21. Lv, Q., Xia, X., Qian, P.: A fast calculation of metric scores for learning Bayesian network. Int. J. Autom. Comput. **9**, 37–44 (2012)
22. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: probabilistic models with unknown objects. In: IJCAI, pp. 1352–1359 (2005)
23. Moore, A.W., Lee, M.S.: Cached sufficient statistics for efficient machine learning with large datasets. JAIR **8**, 67–91 (1998)
24. Natarajan, S., Khot, T., Kersting, K., Gutmann, B., Shavlik, J.W.: Gradient-based boosting for statistical relational learning: the relational dependency network case. Mach. Learn. **86**(1), 25–56 (2012)
25. Niu, F., Ré, C., Doan, A., Shavlik, J.W.: Tuffy: scaling up statistical inference in Markov logic networks using an RDBMS. PVLDB **4**(6), 373–384 (2011)
26. Niu, F., Zhang, C., Ré, C., Shavlik, J.: Felix: Scaling Inference for Markov Logic with an Operator-Based Approach. ArXiv e-prints (2011)
27. Peralta, V.: Extraction and integration of MovieLens and IMDb data. Technical report, Laboratoire PRiSM (2007)
28. Poole, D.: First-order probabilistic inference. In: Gottlob, G., Walsh, T. (eds.) IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003, pp. 985–991. Morgan Kaufmann (2003)
29. Popescul, A., Ungar, L.: Feature generation and selection in multi-relational learning. In: Introduction to Statistical Relational Learning [10], chap. 16, pp. 453–476
30. Qian, Z., Schulte, O.: The Bayes base system (2015). http://www.cs.sfu.ca/~oschulte/BayesBase/BayesBase.html
31. Qian, Z., Schulte, O., Sun, Y.: Computing multi-relational sufficient statistics for large databases. In: CIKM, pp. 1249–1258. ACM (2014)
32. Quakkelaar, R.: Exploiting relational database technology for statistical machine learning in factor base. Master thesis, Open Universiteit Nederland (2017)
33. Ramakrishnan, R., Gehrke, J.: Database Management Systems, 3rd edn. McGraw-Hill, New York (2003)
34. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River (2010)
35. Schulte, O., Khosravi, H.: Learning graphical models for relational data via lattice search. Mach. Learn. **88**(3), 331–368 (2012)
36. Schulte, O., Luo, W., Greiner, R.: Mind-change optimal learning of Bayes net structure from dependency and independency data. Inf. Comput. **208**, 63–82 (2010)
37. Schulte, O., Qian, Z.: Factorbase: SQL for learning a multi-relational graphical model. arXiv preprint (2015). arxiv:abs/1508.02428
38. Singh, A.P., Gordon, G.J.: Relational learning via collective matrix factorization. In: SIGKDD, pp. 650–658. ACM (2008)
39. Singh, S., Graepel, T.: Automated probabilistic modeling for relational data. In: CIKM, pp. 1497–1500. ACM (2013)
40. Sun, Y., Han, J.: Mining Heterogeneous Information Networks: Principles and Methodologies, vol. 3. Morgan & Claypool Publishers, San Rafael (2012)
41. Walker, T., O'Reilly, C., Kunapuli, G., Natarajan, S., Maclin, R., Page, D., Shavlik, J.W.: Automating the ILP setup task: converting user advice about specific examples into general background knowledge. In: ILP, pp. 253–268 (2010)
42. Wang, D.Z., Michelakis, E., Garofalakis, M., Hellerstein, J.M.: BayesStore: managing large, uncertain data repositories with probabilistic graphical models. In: PVLDB, pp. 340–351 (2008)
43. Wick, M.L., McCallum, A., Miklau, G.: Scalable probabilistic databases with factor graphs and MCMC. In: PVLDB, pp. 794–804 (2010)
44. Wong, S.M., Butz, C.J., Xiang, Y.: A method for implementing a probabilistic model as a relational database. In: UAI, pp. 556–564 (1995)

Journal: 41060 MS: 0130 TYPESET DISK LE CP Disp.:2018/5/26 Pages: 21 Layout: Large

Journal: 41060
Article: 130

Springer
the language of science

Author Query Form

Please ensure you fill out your response to the queries raised below and return this form along with your corrections

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query	Details required	Author's response
1.	Please confirm the inserted city name "Burnaby" is correct and amend if necessary.	
2.	The term 'statistical-relational' has been changed to 'statistical–relational' throughout the article. Please check.	
3.	Tables 6 and 7 are inter changed to ensure sequential ordering as per citation. Please check.	
4.	The sentence 'Similarly for Forbidden...' is incomplete. Please check.	
5.	Please check and confirm whether the edit made in the sentence 'This is even more the case...' conveys the intended meaning.	
6.	Please provide accessed date for the Refs. [2, 17, 30].	