Computing Multi-Relational Sufficient Statistics for Large Databases

ABSTRACT

Utilizing the information in a relational database for statistical modelling and pattern mining requires fast access to multi-relational sufficient statistics, that combine information across database tables. Previously established techniques are available to compute multi-relational sufficient statistics for conjunctive queries with positive relationships only. We present an efficient dynamic program that computes sufficient statistics for any combination of positive *and* negative relationships, starting with a set of statistics for positive relationships only. Our dynamic program performs a virtual join operation, that counts the number of statistics in a table join without actually constructing the join. We show that the run time of the algorithm is $O(r \log r)$, where r is the number of sufficient statistics to be computed. The computed statistics are stored in contingency tables. We introduce contingency table algebra, an extension of relational algebra, to elegantly describe and efficiently implement the dynamic program. Empirical evaluation on seven benchmark databases demonstrates the scalability of our algorithm; we compute sufficient statistics with positive and negative relationships in databases with over 1 million data records. Our experiments illustrate how access to sufficient statistics for both positive and negative relationships enhances feature selection, rule mining, and generative modelling.

1. INTRODUCTION

Relational databases contain information about attributes of entities, and which relationships do and do not hold among entities. To make this information accessible for knowledge discovery requires requires computing *sufficient statistics*. For relational statistical analysis to discover cross-table correlations, these sufficient statistics must be instantiation counts for conjunctive queries that combine information from different database tables, and that may contain any number of *positive and negative* relationships. Negative relationships concern the nonexistence of a relationship. Such statistics are important for learning correlations between different relationship types (e.g., if user u performs a web search for item i, is it likely that u watches a video about i ?).

Whereas sufficient statistics with positive relationships only can be efficiently computed by SQL joins of existing database tables, a table join approach is not feasible for negative relationships. This is because we would have to enumerate all tuples of entities that are *not* related (consider the number of user pairs who are *not* friends on Facebook). The cost of the enumeration approach is close to materializing the Cartesian product of entity sets, which grows exponentially with the number of entity sets involved. It may therefore seem that sufficient statistics with negative relationships can be computed only for small databases. Indeed such sufficient statistics have not been previously used in multi-relational data analysis, to our knowledge. We show that on the contrary, assuming that sufficient statistics with positive relationships are available, extending them to negative relationships can be achieved in a highly scalable manner, which does not depend on the size of the database.

Virtual Join Approach. Our approach to this problem introduces a new virtual join operation. A virtual join algorithm computes sufficient statistics *without* materializing a cross product [14]. Sufficient statistics can be represented in contingency tables [4]. Our virtual join operation is a dynamic programming algorithm that successively builds up a large contingency table from smaller ones, *without a need to access the original data tables.*

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

Evaluation. We evaluate our Virtual Join algorithm by com-

puting contingency tables for seven real-world databases. The computation times exhibit the near-linear growth predicted by our theoretical analysis. They range from two seconds on the simpler database schemas to just over two hours for the most complex schema with over 1 million tuples from the IMDB database.

Given that computing sufficient statistics for negative relationships is *feasible*, the remainder of our experiments evaluate their *usefulness*. These sufficient statistics allow statistical analysis to utilize the absence or presence of a relationship as a feature. Our benchmark datasets provide evidence that the relationship indicator features enhance different types of statistical analysis, as follows. (1) Feature selection: We learn two different feature sets for each of our databases and a given target class label. A standard feature selection method selects different features for classification when provided with statistics for negative and positive relationships, from those that it selects when given positive relationship statistics only. (2) Association Rule Mining: A standard association rule learning method includes many association rules with relationship conditions in its top 20 list. (3) Bayesian network learning. A Bayesian network provides a graphical summary of the probabilistic dependencies among relationships and attributes in a database. We learn two Bayesian network structures on each of our database, one whose input is a contingency table with only positive relationships, one whose input is a contingency table with both positive and negative relationships. Our largest database is an order of magnitude larger than the databases for which graphical models have been learned previously [9]. The two Bayes net structures learned are different for all but one database, and make different trade-offs between model likelihood and model complexity. On the two datasets with the most complex schemas, enhanced sufficient statistics lead to a clearly superior model (better data fit with fewer parameters).

Contributions. Our main contributions are as follows.

1. A dynamic program to compute a joint contingency table for sufficient statistics that combine several tables, and that may involve any number of *positive and negative* relationships.
2. An extension of relational algebra for contingency tables that supports the dynamic program conceptually and computationally.

Paper Organization. We review background for relational databases and statistical concepts. The main part of the paper describes the dynamic programming algorithm for computing a joint contingency table for all random variables. We describe the contingency table algebra for dealing with negative relationships. A complexity analysis establishes feasible upper bounds on the number of contingency table operations required by the Virtual Join algorithm. We also investigate the scalability of the algorithm empirically. The final set of experiments examines how the cached sufficient statistics support the analysis of cross-table dependencies for different learning and data mining tasks.

2. BACKGROUND AND NOTATION

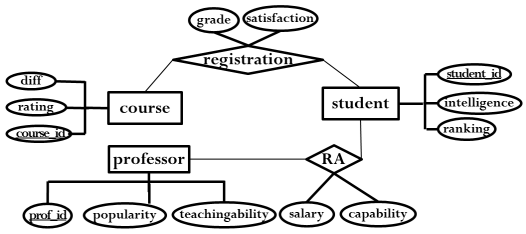

Figure 1: A relational ER Design.

We assume a standard **relational schema** containing a set of tables, each with key fields, descriptive attributes, and possibly foreign key pointers. A **database instance** specifies the tuples contained in the tables of a given database schema. We assume that tables in the relational schema can be divided into *entity tables* and *relationship tables*. This is the case whenever a relational schema is derived from an entity-relationship model (ER model) [12, Ch.2.2]. A **table join** of two or more tables contains the rows in the Cartesian products of the tables whose values match on common fields.

2.1 Relational Random Variables

We adopt function-based notation from logic for combining statistical and relational concepts [7]. A domain or **population** is a set of individuals. Individuals are denoted by lower case expressions (e.g., *bob*). A **functor** represents a mapping $f : \mathcal{P}_1, \ldots, \mathcal{P}_a \rightarrow V_f$ where f is the name of the functor, each $\mathcal{P}_i$ is a population, and V_f is the output type or **range** of the functor. In this paper we consider only functors with a finite range, disjoint from all populations. If $V_f = \{\mathrm{T}, \mathrm{F}\}$, the functor f is a (Boolean) **predicate**. A predicate with more than one argument is called a **relationship**; other functors are called **attributes**. We use uppercase for predicates and lowercase for other functors. Throughout this paper we assume that all relationships are binary, though this is not essential for our algorithm.

A **(Parametrized) random variable** (PRV) is of the form $f(\mathbb{X}_1, \ldots, \mathbb{X}_a)$, where each $\mathbb{X}_i$ is a first-order variable [6]. Each first-order variable is associated with a population/type.

Student

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

(a)

Course

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

(b)

Professor

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

(c)

RA

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

(d)

Registration

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

(e)

Figure 2: Database Instance based on Figure 1.

The functor formalism is rich enough to represent the constraints of an entity-relationship schema via the following translation: Entity sets correspond to populations, descriptive attributes to functions, relationship tables to relation-

ER Diagram	Type	Functor	Random Variable
Relation Tables	RVars	RA	$RA(\mathbb{P},\mathbb{S})$
Entity Attributes	*1Atts*	intelligence, ranking	$\{intelligence(\mathbb{S}), ranking(\mathbb{S})\}$ $= 1Atts(\mathbb{S})$
Relationship Attributes	*2Atts*	teaching-ability, salary	$\{teaching-ability(\mathbb{P},\mathbb{S}), salary(\mathbb{P},\mathbb{S})\}$ $= 2Atts(RA(\mathbb{P},\mathbb{S}))$

Table 1: Translation from ER Diagram to Random Variables.

ships, and foreign key constraints to type constraints on the arguments of relationship predicates. Table 1 illustrates this translation, distinguishing attributes of entities (*1Atts*) and attributes of relationships (*2Atts*).

2.2 Contingency Tables

Sufficient statistics can be represented in *contingency tables* as follows [4]. Consider a fixed list of random variables. A **query** is a set of ($variable = value$) pairs where each value is of a valid type for the random variable. The **result set** of a query in a database $\mathcal{D}$ is the set of instantiations of the first-order variables such that the query evaluates as true in $\mathcal{D}$. For example, in the database of Figure 2 the result set for the query ($intelligence(\mathbb{S}) = 2$, $rank(\mathbb{S}) = 1$, $popularity(\mathbb{P}) = 3$, $teaching-ability(\mathbb{P}) = 1$, $RA(\mathbb{P},\mathbb{S}) = T$) is the singleton $\{\langle kim, oliver\rangle\}$. The **count** of a query is the cardinality of its result set.

For every set of variables $\mathbf{V} = \{V_1,\ldots,V_n\}$ there is a **contingency table** $ct(\mathbf{V})$. This is a table with a row for each of the possible assignments of values to the variables in $\mathbf{V}$, and a special integer column called *count*. The value of the *count* column in a row corresponding to $V_1 = v_1,\ldots,V_n = v_n$ records the count of the corresponding query. Figure 3 shows the contingency table for the university database. The value of a relationship attribute is undefined for entities that are not related. Following [7], we indicate this by writing $capability(\mathbb{P},\mathbb{S}) = n/a$ for a reserved constant n/a. The assertion $capability(\mathbb{P},\mathbb{S})$ = n/a is therefore equivalent to the assertion that $RA(\mathbb{P},\mathbb{S})$ = F. A **conditional contingency table**, written

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

Figure 3: Excerpt from the joint contingency table for the university database of Figure 2.

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

is the contingency table whose column headers are $V_1,\ldots,V_k$ and whose counts are defined by subset of instantiations that match the condition to the right of the | symbol. We assume that contingency tables omit rows with count 0.

3. RELATIONAL CONTINGENCY TABLES

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

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

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

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

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

The subset ordering defines a lattice on relationship sets/chains. Figure 4 illustrates the lattice for the relationship variables in the university schema. For reasons that we explain below, entity tables are also included in the lattice and linked to relationships that involve the entity in question. With

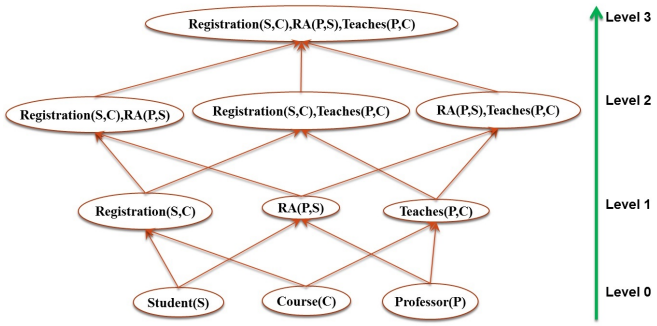

Figure 4: A lattice of relationship sets for the university schema of Figure 1.

each relationship chain $\mathbf{R}$ (Rchain for short) is associated a *ct*-table $ct_{\mathbf{R}}$. The variables in the *ct*-table $ct_{\mathbf{R}}$ comprise the relationship variables in $\mathbf{R}$, and the unary/binary descriptive attributes associated with each of the relationships. To describe these, we introduce the following notation (cf. Table 1).

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

In this notation, the variables in the *ct*-table $ct_{\mathbf{R}}$ are denoted as $\mathbf{R} \cup Atts(\mathbf{R})$. The goal of the Virtual Join algorithm is to compute a contingency table for each chain $\mathbf{R}$. In the example of Figure 4, the algorithm computes 10 contingency tables. The *ct*-table for the top element of the lattice is the **joint** *ct*-**table** for the entire database.

If a conjunctive query involves only positive relationships, then it can be computed using SQL's count aggregate function applied to a table join. To illustrate, we show the SQL for computing the positive relationship part of the *ct*-table for the $RA(\mathbb{P}, \mathbb{S})$ chain.

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

Even more efficient than SQL count queries is the Tuple ID propagation method, a Virtual Join method for computing query counts with positive relationships only [14]. In the next section we assume that contingency tables for positive relationships only have been computed already, and consider how such tables can be extended to full contingency tables with both positive and negative relationships.

4. COMPUTING CONTINGENCY TABLES FOR NEGATIVE RELATIONSHIPS

We describe a Virtual Join algorithm that computes the required sufficient statistics without the materializing a Cartesian product of entity sets. Our experiments below compare the virtual joins with materializing table joins. First, we introduce an extension of relational algebra that we term **contingency table algebra**. The purpose of this extension is to show that query counts using $k+1$ negative relationships can be computed from two query counts that each involve at most k relationships. Second, a dynamic programming algorithm applies the algebraic identify repeatedly to build up a complete contingency table from partial tables.

4.1 Contingency Table Algebra

We introduce relational algebra style operations defined on contingency tables.

4.1.1 Unary Operators

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

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

```
SELECT SUM(count) AS count, V_1, ..., V_k
FROM ct
GROUP BY V_1, ..., V_k
```

Conditioning $\chi_\phi ct$ returns a conditional contingency table. Ordering the columns as $(V_1, \ldots, V_k, \ldots, V_{k+j})$, suppose that the selection condition is a conjunction of values of the form $C = (V_{k+1} = v_{k+1}, \ldots, V_{k+j} = v_{k+j})$. Conditioning can be defined in terms of selection and projection by the equation:

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

4.1.2 Binary Operators

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

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

```
SELECT
(ct_1.count * ct_2.count) AS count, U, V
FROM ct_1, ct_2
```

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

```
SELECT ct_1.count+ct_2.count AS count, V
FROM ct_1, ct_2
WHERE ct_1.V = ct_2.V
```

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

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

4.1.3 Implementation

The selection operator can be implemented using SQL as with standard relational algebra. Projection with *ct*-tables requires use of the GROUP BY construct as shown in Section 4.1.1.

For addition/subtraction, simply executing the SQL query shown above may lead to a quadratic cost if a nested-loops join is used, and therefore does not scale to large datasets. A more efficient algorithm is a sort-merge join [12]. Given two union-compatible *ct*-tables, each row in one matches at most one other on the non-count columns. As is well known, with unique matches, the cost of a sort-merge join is *size*(*table1*) + *size*(*table2*)+ the cost of sorting both tables. Our experiments below are based on a sort-merge join (our own implementation in Java).

The cross product is easily implemented in SQL as shown in Section 4.1.2. The cross product size is quadratic in the size of the input tables, so a quadratic cost is unavoidable.

4.2 Lattice Computation of Contingency Tables

This section describes a method for computing the contingency tables level-wise in the relationship chain lattice. We start with a contingency table algebra equivalence that allows us to compute counts for rows with negative relationships from rows with positive relations. Following [4], we use a "don't care" value $*$ to indicate that a query does not specify the value of a node. For instance, the query $R_1 = \mathrm{T}, R_2 = *$ is equivalent to the query $R_1 = \mathrm{T}$.

PROPOSITION 1. *Let R be a relationship variable and let $\mathbf{R}$ be a set of relationship variables. Let Vars be a set of variables that does not contain R nor any of the 2Atts of R. Let $\mathbb{X}_1, \ldots, \mathbb{X}_l$ be the first-order variables that appear in R but not in Vars, where l is possibly zero. Then we have*

$$ct(\mathit{Vars} \cup \mathit{1Atts}(R)|\mathbf{R} = \mathrm{T}, R = F) = \quad (1)$$
$$ct(\mathit{Vars}|\mathbf{R} = \mathrm{T}, R = *) \times ct(\mathbb{X}_1) \times \cdots \times ct(\mathbb{X}_l)$$
$$- ct(\mathit{Vars} \cup \mathit{1Atts}(R)|\mathbf{R} = \mathrm{T}, R = T).$$

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

We omit the proof due to space constraints. The core of the argument is that Equation 1 can be treated as a relational algebra analog of the probabilistic equality $P(\phi, R = \mathrm{F}) = P(\phi) - P(\phi, R = \mathrm{T})$ that holds for any condition ϕ.

Algorithm 1: The Pivot function returns a conditional contingency table for a set of attribute variables and all possible values of the relationship R_{pivot}, including $R_{pivot} = \mathrm{F}$. The set of conditional relationships $\mathbf{R} = (R_{pivot}, \ldots, R_\ell)$ may be empty in which case the Pivot computes an unconditional ct-table.

Input: Two conditional contingency tables
$ct_{\mathrm{T}} := ct(\mathit{Vars}, \mathit{2Atts}(R_{pivot})|R_{pivot} = \mathrm{T}, \mathbf{R} = \mathrm{T})$
and $ct_* := ct(\mathit{Vars}|R_{pivot} = *, \mathbf{R} = \mathrm{T})$.
Precondition: The set *Vars* does not contain the relationship variable R_{pivot} nor any of its descriptive attributes $\mathit{2Atts}(R_{pivot})$.;
Output: The conditional contingency table
$ct(\mathit{Vars}, \mathit{2Atts}(R_{pivot}), R_{pivot}|\mathbf{R} = \mathrm{T})$.

1: $ct_{\mathrm{F}} := ct_* - \pi_{\mathit{Vars}} ct_{\mathrm{T}}$.
{Implements the algebra Equation 1 in proposition 1.}
2: ct^+_{F} := extend ct_{F} with columns R_{pivot} everywhere false and $\mathit{2Atts}(R_{pivot})$ everywhere n/a.
3: ct^+_{T} := extend ct_{T} with columns R_{pivot} everywhere true.
4: **return** $ct^+_{\mathrm{F}} \cup ct^+_{\mathrm{T}}$

The construction of the ct_{F} table in Figure 5 illustrates Equation (1). Algorithm 1 provides pseudo-code for applying Equation (1) to compute a complete *ct*-table given a partial table where a specified relationship variable R is true and another partial table that does not contain the relationship variable. We refer to R as the **pivot** variable. For extra generality, we apply Equation (1) with a condition that lists a set of relationship variables fixed to be true. Algorithm 2 shows how the Pivot operation can be applied repeatedly to find all contingency tables in the relationship lattice. Figure 5 illustrates the computation for the case of only one relationship. The computation proceeds as follows.

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

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

4.3 Complexity Analysis

The key point about the Virtual Join algorithm is that it avoids the materializing the cross product of entity tuples. *The algorithm accesses only **existing** tuples, never constructs nonexisting tuples.* The number of *ct*-table operation is therefore independent of the number of data records in the original database. We bound the total number of *ct*-algebra operations performed by the Virtual Join algorithm in terms of the size of its output, the number of sufficient statistics that involve negative relationships.

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

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

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

Derivation Outline. Let m be the number of relationship variables. By counting *ct*-table operations in the lattice, we can show that

$$\#ct\_ops = O(m \cdot 2^{m-1}). \quad (2)$$

Now let s be the number of sufficient statistics where all relationship variables are true. For instance, if there are n binary attributes, then $s = 2^n$. Then we have

$$r = s \cdot (2^m - 1) \quad (3)$$

since there are $2^m - 1$ combinations of relationship values with at least one negative relationship. Solving for m and

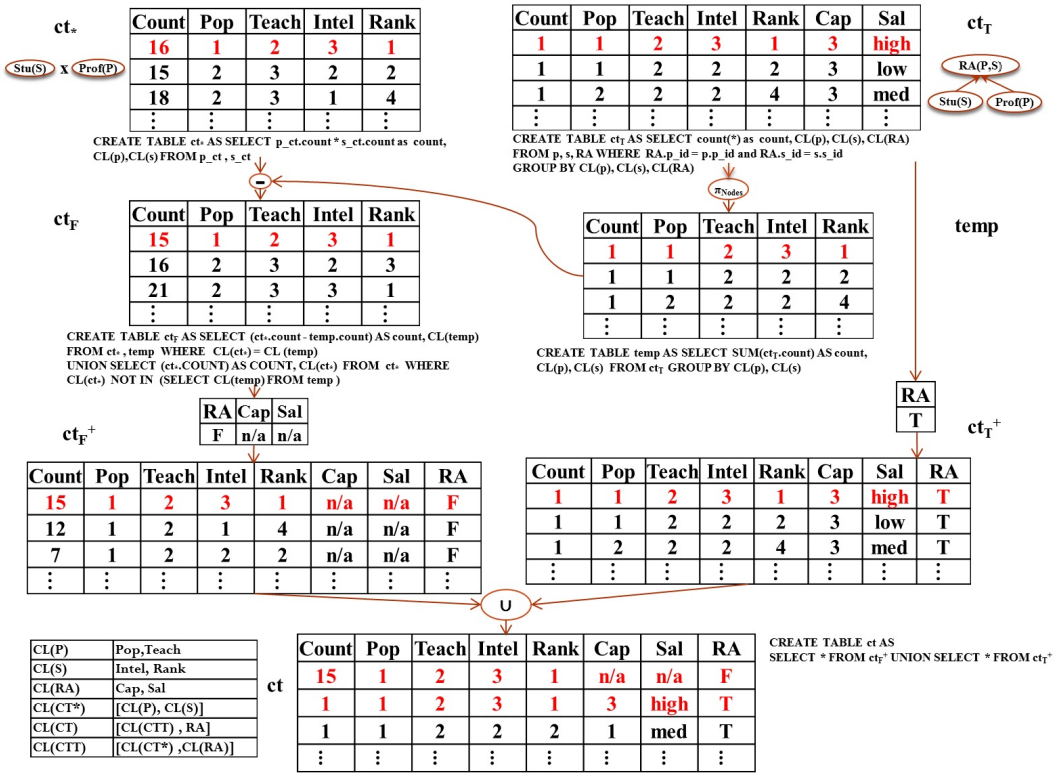

Figure 5: Computation of the ct-table for the relationship $RA(\mathbb{P}, \mathbb{S})$ using Algorithm 1. The operations are implemented using dynamic SQL queries as shown. Lists of column names are abbreviated as shown and as follows. $CL(ct_*) = CL(temp) = CL(ct_F)$, $CL(ct) = CL(ct_F^+) = CL(ct_T^+)$.

substituting the result into the expression (2) leads to the desired bound.

Since the time cost of any algorithm must be at least as great as the time for writing the output, which is as least as great as r, the Virtual Join algorithm adds at most a logarithmic factor to this lower bound. This means that if the number r of sufficient statistics is a reasonable bound on computational time and space, then computing the sufficient statistics is feasible. In our benchmark datasets, the number of sufficient statistics was feasible, as we report below. In Section 8 below we discuss options in case the number of sufficient statistics grows too large.

5. EVALUATION OF CONTINGENCY TABLE COMPUTATION

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

5.1 Datasets

We used seven benchmark real-world databases, described by Schulte and Khosravi [9]. See that article for more details and the sources of the databases. Table 2 summarizes basic information about the benchmark datasets. A self-relationship relates two entities of the same type (e.g. *Borders* relates two countries in Mondial). Random variables for each database were defined as described in Section 2.1 (see also [9]). IMDB is the largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. Our dataset combines the MovieLens database with data from the Internet Movie Database (IMDB)[1] following [5].

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

Table 2: Datasets characteristics. #Tuples = total number of tuples over all tables.

5.2 Contingency Tables With Negative Relationships

Table 3 reports measurements about our Virtual Join (VJ) algorithm for constructing the joint contingency tables for

[1]www.imdb.com, July 2013

Algorithm 2: Virtual Join algorithm for Computing the Contingency Table for Input Database

Input: A relational database $\mathcal{D}$; a set of variables
Output: A contingency table that lists the count in the database D for each possible assignment of values to each variable.

for all first-order variables $\mathbb{X}$ **do**
  compute $ct(\mathit{1Atts}(\mathbb{X}))$ using SQL queries.
end for
for all relationship variable R **do**
  $ct_* := ct(\mathbb{X}) \times ct(\mathbb{Y})$ where $\mathbb{X}$,$\mathbb{Y}$ are the first-order variables in R.
  $ct_{\mathrm{T}} := ct(\mathit{1Atts}(R)|R = \mathrm{T})$ using SQL joins.
  Call $\mathit{Pivot}(ct_{\mathrm{T}}, ct_*)$ to compute $ct(\mathit{1Atts}(R), \mathit{2Atts}(R), R)$.
end for
for Rchain length $\ell = 2$ to m **do**
  for all Rchain $R_1, \ldots, R_\ell$ **do**
    $\mathit{Current\_ct} := ct(\mathit{1Atts}(R_1, \ldots, R_\ell), \mathit{2Atts}(R_1, \ldots, R_\ell)|R_1 = \mathrm{T}, \ldots, R_\ell = \mathrm{T})$ using SQL joins.
    for $i = 1$ to ℓ **do**
      if i equals 1 **then**
        $ct_* := ct(\mathit{1Atts}(R_2, \ldots, R_\ell), \mathit{2Atts}(R_2, \ldots, R_\ell)|R_1 = *, R_2 = \mathrm{T}, \ldots, R_\ell = \mathrm{T}) \times ct(\mathbb{X})$ where $\mathbb{X}$ is the first-order variable in R_1, if any, that does not appear in $R_2, \ldots, R_\ell$ {ct_* can be computed from a ct-table for a Rchain of length $\ell - 1$.}
      else
        $\mathit{1Atts}_{\bar{i}} := \mathit{1Atts}(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_\ell)$.
        $\mathit{2Atts}_{\bar{i}} := \mathit{2Atts}(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_\ell)$.
        $ct_* := ct(\mathit{1Atts}_{\bar{i}}, \mathit{2Atts}_{\bar{i}}, R_1, \ldots, R_{i-1})|R_i = *, R_{i+1} = \mathrm{T}, \ldots, R_\ell = \mathrm{T}) \times ct(\mathbb{Y})$ where $\mathbb{Y}$ is the first-order variable in R_i, if any, that does not appear in $\mathbf{R}$.
      end if
      $\mathit{Current\_ct} := \mathit{Pivot}(\mathit{Current\_ct}, ct_*)$.
    end for{Loop Invariant: After iteration i, the table $\mathit{Current\_ct}$ equals $ct(\mathit{1Atts}(R_1, \ldots, R_\ell), \mathit{2Atts}(R_1, \ldots, R_\ell), R_1, \ldots, R_i|R_{i+1} = \mathrm{T}, \ldots, R_\ell = \mathrm{T})$}
  end for{Loop Invariant: The ct-tables for all Rchains of length ℓ have been computed.}
end for
return the ct-table for the Rchain involves all the relationship variables.

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

Table 3: Constructing the contingency table for each dataset. M = million. N.T. = non-termination.

all variables together, for each database. We compare the VJ algorithm with using an SQL query to construct the contingency table with negative relationships. Cross-checking the VJ contingency tables with the SQL contingency tables confirmed the correctness of our implementation. The SQL query materializes the cross product of the entity tables for each first-order variable (primary keys). The ratio of the cross product size to the number of statistics in the ct-table measures how much compression the ct-table provides compared to enumerating the cross product. The ct-table provides a substantial compression of the statistical information in the database, by a factor of over 4,500 for the largest database IMDB.

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

The SQL query did not always terminate, crashing after around 4, 5, and 10 hours on Financial, IMDB and Hepatitis respectively. When the SQL query did terminate, it took orders of magnitude longer than the VJ method except for the Mondial dataset. Generally the higher the compression ratio, the higher the time savings. On Mondial the compression ratio is especially low, so materializing the cross-product was faster. We also tried a more complex SQL query that combines the cross-product (in the FROM clause) with the Count aggregate function and Group By on the variables. This ran even more slowly because it requires counting in addition to processing the cross-product.

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

Table 4: Number of Sufficient Statistics for Link Analysis On and Off.

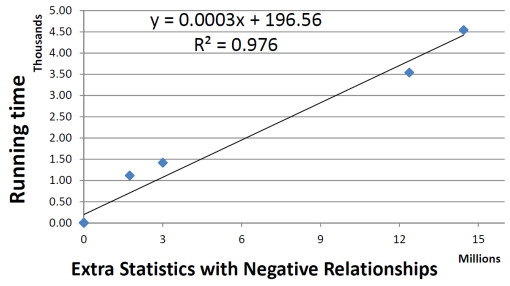

Figure 6: The VJ Algorithm Running Time (s)

5.3 Contingency Tables with Negative Relationships vs. Positive Relationships Only

Here and below we use the following terminology. **Link Analysis On** refers to using a contingency table with sufficient statistics for both positive and negative relationships. An example is table ct in Figure 5. **Link Analysis Off** refers to using a contingency table with sufficient statistics for positive relationships only. An example is table ct_{T}^{+} in Figure 5. Table 4 shows the number of sufficient statistics required for link analysis on vs. off. The difference between the link analysis on statistics and the link analysis statistics is the number of Extra Statistics. The Extra Time column shows how much time the VJ algorithm requires to compute the Extra Statistics *after* the contingency tables for positive relationships are constructed using SQL joins. As Figure 6 illustrates, the Extra Time stands in a nearly linear relationship to the number of Extra Statistics, which confirms the analysis of Section 4.3. Figure 7 shows that most of the VJ run time is spent on the Pivot component (Algorithm 1) rather than the main loop (Algorithm 2). In terms of ct-table operations, most time is spent on subtraction/union rather than cross product.

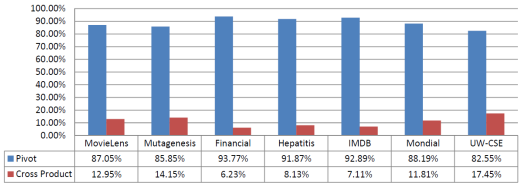

Figure 7: Breakdown of VJ Running Time

If the link analysis off contingency table contains no zero count statistics, the number of Extra Statistics is at most $2^m - 1$ times the number required for link analysis off, where m is the number of relationships (cf. Equation (3)). For example, the IMDB dataset contains three relationship variables (cf. Table 2), so the link analysis on statistics increase by at most $2^3 = 8$. However, the actual number is 15 times bigger. This shows that when all relationships are positive, many possible attribute combinations do not occur in IMDB. The reason for this is that not all entities participate in every relationship. Attribute information for these entities is lost conditional on the relationship being true. An extreme example is the Mondial dataset where all relationship variables are never simultaneously true. In such cases there is a trade-off between turning the link analysis on and off: On the one hand, the number of extra sufficient statistics and hence the space storage requirements is bigger than one would expect from the number of relationships alone. On the other hand, turning link analysis off loses information not only about the distribution of relationships, but also about the distribution of attributes in the database. We next examine how this loss of information affects statistical learning and mining tasks.

6. STATISTICAL APPLICATIONS

We evaluate using link analysis on three different types of cross-table statistical analysis: feature selection, association rule mining, and learning a Bayesian network.

6.1 Feature Selection

For each database, we selected a target for classification, then used Weka's CFS feature subset selection method (Version 3.6.7) to select features for classification [2]. The idea is that if the existence of relationships is relevant to classification, then there should be a difference between the set selected with link analysis on and that selected with link analysis off. We measure how different two feature sets are by 1-Jaccard's coefficient:

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

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

Table 5: Selected Features for Target variables for Link Analysis Off vs. Link Analysis On. Rvars denotes the number of relationship features selected.

Distinctness measures how different the selected feature subset is with link analysis on and off, on a scale from 0 to 1. Here 1 = maximum dissimilarity. Table 5 compares the feature sets selected. In 4/7 datasets, the feature sets are disjoint (coefficient = 0). For the Mutagenesis and MovieLens data sets, no new features are selected. In almost all datasets, sufficient statistics about negative relationships generate new relevant features for classification.

6.2 Association Rules

A widely studied task is finding interesting association rules in a database. We considered association rules of the form $body \rightarrow head$, where *body* and *head* are conjunctive queries. We searched for interesting rules using both the link analysis off and the link analysis on contingency tables for each database. The idea is that if a relationship indicator is relevant for other features, it should appear in an association rule. With link analysis off, all relationship indicators always have the value T, so they do not appear in any association rule. We used Weka's Apriori implementation to search for association rules in both modes. The interestingness metric was Lift. Parameters were set to their default values.

Table 6 shows the number of rules that utilize relationship indicators with link analysis on, out of the top 20 rules. In all cases, a majority of rules utilize relationship indicators, all of them in Mutagenesis and IMDB. An example of an association rule for Financial is

$$statement\_freq.(acc) = monthly \rightarrow HasLoan(acc, Loan) = \text{T}.$$

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

Table 6: Number of top 20 Association Rules that use relationship indicator features.

6.3 Learning Bayesian Networks

Our most challenging application is constructing a Bayesian network for a relational database. For single-table data, Bayesian network learning has been considered as a benchmark application for precomputing sufficient statistics [4, 3]. A Bayesian network structure is a directly acyclic graph whose nodes are random variables. Given an assignment of values to its parameters, a Bayesian network represents a joint distribution over both attributes and relationships in a relational database. Several researchers have noted the usefulness of constructing a graphical statistical model for a relational database [11, 13]. For data exploration, a Bayes net model provides a succinct graphical representation of complex statistical-relational correlations. The model also supports probabilistic reasoning for answering "what-if" queries about the probabilities of uncertain outcomes conditional on observed events.

We used the previously existing learn-and-join method (LAJ), which is the state of the art for Bayes net learning in relational databases [9]. The LAJ method takes as input a contingency table for the entire database, so we can apply it with both link analysis on and link analysis off to obtain two different Bayes net structures for each database. Our experiment is the first evaluation of the LAJ method with link analysis on. We used the LAJ implementation provided by its creators. We score all learned graph structures using the same full contingency table with link analysis on, so that the scores are comparable. The idea is that turning link analysis on should lead to a different structure that represents correlations, involving relationship variables, that exist in the data.

6.3.1 Structure Learning Times

Table 7 provides the model search time for structure learning with link analysis on and off. Both learning methods are fast, even for the largest contingency table IMDB (less than 10 minutes run-time). With link analysis on, structure learning takes more time as it processes more information. In both modes, the run-time for building the contingency tables (Table 3) dominates the structure learning cost.

6.3.2 Statistical Scores.

We report two model metrics, the log-likelihood score, and the model complexity as measured by the number of parameters. The **log-likelihood** is denoted as $L(\hat{G}, \mathbf{d})$, where $\hat{G}$ is the BN G with its parameters instantiated to be the maximum likelihood estimates given the dataset $\mathbf{d}$, and the quantity $L(\hat{G}, \mathbf{d})$ is the log-likelihood of $\hat{G}$ on $\mathbf{d}$. We use the relational log-likelihood score defined in [8], which differs from the standard single-table Bayes net likelihood only by replacing counts by frequencies so that scores are comparable across different nodes and databases. To provide information about the qualitative graph structure learned, we report edges learned that point to a relationship variable as a child. Such edges can be learned only with link analysis on. We distinguish edges that relationship variables—R2R—and that link attribute variables to relationships—A2R.

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

Table 7: Model Structure Learning Time in seconds.

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

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

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

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

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

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

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

Table 8: Comparison of Statistical Performance of Bayesian Network Learning.

Structure learning can use the new type of dependencies to find a better, or at least different, trade-off between model complexity and model fit. On two datasets (IMDB and Financial), link analysis leads to a superior model that achieves better data fit with fewer parameters. These are also the datasets with the most complex relational schemas (see Table 2). On IMDB in particular, considering only positive links leads to a very poor structure with a huge number of parameters. On four datasets, extra sufficient statistics lead to different trade-offs: On MovieLens and Mutagenesis, link analysis leads to better data fit but higher model complexity,

and the reverse for Hepatitis and UW-CSE.

7. RELATED WORK

Sufficient Statistics for Single Data Tables. Several data structures have been proposed for storing sufficient statistics defined on a *single* data table. One of the most well-known are ADtrees [4]. The ADtree provides a memory-efficient data structure for *storing* and retrieving sufficient statistics once they have been computed. In this paper, we focus on the problem of *computing* the sufficient statistics, especially for the case where the relevant rows have not been materialized. Thus ADtrees and contingency tables are complementary representations for different purposes: contingency tables support a computationally efficient block access to sufficient statistics, whereas ADtrees provide a memory efficient compression of the sufficient statistics. An interesting direction for future work is to build an ADtree for the contingency table once it has been computed.

Relational Sufficient Statistics. Getoor et al. provided a subtraction method for the special case of estimating counts with only a single negative relationship [1, Sec.5.8.4.2]. They did not treat contingency tables with multiple negative relationships. Schulte *et al.* show that the fast Möbius transform can be used to extend the subtraction method to the case of multiple negative relationships [10]. They considered only Bayes net parameter learning, not structure learning. Parameter learning requires sufficient statistics only for a child node and its parents (at most 6 nodes in their experiments). Their evaluation involved statistics for only one relationship. In contrast, our method can be used to support Bayes net structure learning, which requires joint sufficient statistics over the entire database. Other novel aspects are the *ct*-table operations and using the relationship chain lattice to facilitate dynamic programming.

8. CONCLUSION

Databases contain information about which relationships do and do not hold among entities. To make this information accessible for statistical analysis requires computing sufficient statistics that combine information from different database tables, and may involve any number of *positive and negative* relationships. With a naive enumeration approach, computing sufficient statistics for negative relationships is feasible only for small databases. We solve this problem with a new dynamic programming algorithm that performs a virtual join, where the requisite counts are computed without materializing any join tables. A new extension of relational algebra, called *ct*-table algebra, facilitates the efficient implementation of this virtual join operation. The efficient computation of sufficient statistics allows the system to scale to large datasets (over 1M tuples) with complex schemas. Empirical evaluation with seven benchmark datasets showed that information about the presence and absence of links can be exploited in feature selection, association rule mining, and Bayesian network learning.

Limitations and Future Work. Our dynamic program scales well with the number of rows, but not with the number of columns and relationships in the database. This limitation stems from the fact that the contingency table size grows exponentially with the number of random variables in the table. In this paper, we applied the algorithm to construct a large table for *all* variables in the database. We emphasize that this is only one way to apply the algorithm. The Virtual Join algorithm efficiently finds cross-table statistics for any set of variables, not only for the complete set of all variables in the database. An alternative is to apply the virtual join only up to a pre-specified relatively small relationship chain length. Another possibility is to use postcounting [3]: Rather than precompute a large contingency table prior to learning, compute many small contingency tables for small subsets of variables on demand during learning.

In sum, our Virtual Join algorithm efficiently computes query counts which may involve any number of *positive and negative* relationships. These sufficient statistics support a scalable statistical analysis of associations among both relationships and attributes in a relational database.

9. REFERENCES

[1] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning*, chapter 5, pages 129–173. MIT Press, 2007.

[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[3] Q. Lv, X. Xia, and P. Qian. A fast calculation of metric scores for learning bayesian network. *International Journal of Automation and Computing*, 9:37–44, 2012.

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

[5] V. Peralta. Extraction and integration of movielens and IMDB datas. Technical report, Laboratoire PRiSM, Universite de Versailles, 2007.

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

[7] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010.

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

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

[10] O. Schulte, H. Khosravi, A. Kirkpatrick, T. Gao, and Y. Zhu. Modelling relational statistics with bayes nets. *Machine Learning*, 94:105–125, 2014.

[11] S. Singh and T. Graepel. Automated probabilistic modelling for relational data. In *CIKM*, pages 1497–1500, 2013.

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

[13] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. pages 340–351, 2008.

[14] X. Yin, J. Han, J. Yang, and P. S. Yu. Crossmine: Efficient classification across multiple database relations. In *ICDE*, pages 399–410. IEEE Computer Society, 2004.