Computing Multi-Relational Sufficient Statistics for Large Databases

Zhensong Qian  
School of Computing Science  
Simon Fraser University, CA  
zqian@sfu.ca

Oliver Schulte  
School of Computing Science  
Simon Fraser University, CA  
oschulte@sfu.ca

Yan Sun  
School of Computing Science  
Simon Fraser University, CA  
sunyans@sfu.ca

ABSTRACT

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. Such statistics 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 join tables. Contingency table algebra is a new extension of relational algebra, that facilitates the efficient implementation of this Möbius virtual join operation. The Möbius Join scales 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.

Categories and Subject Descriptors

H.2.8 [Database Applications]: Data mining; H.2.4 [Systems]: Relational databases

Keywords

sufficient statistics; multi-relational databases; virtual join; relational algebra

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 discrete data, these sufficient statistics are instantiation counts for conjunctive queries. For relational statistical analysis to discover cross-table correlations, sufficient statistics must combine information from different database tables. This paper describes a new dynamic programming algorithm for computing cross-table sufficient statistics that may contain any number of positive and negative relationships. Negative relationships concern the nonexistence of a relationship. Our algorithm makes the joint presence/absence of relationships available as features for the statistical analysis of databases. For instance, 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 cross product of entity sets, which grows exponentially with the number of entity sets involved. It may therefore seem that sufficient statistics with negative relationships can be computed only for small databases. 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 [16]. Sufficient statistics can be represented in contingency tables [5]. 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 refer to it as the Möbius Join since it is based on the Möbius extension theorem [12].

We introduce algebraic 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. The Möbius Join 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. A theoretical upper bound for the number of contingency table operations required by the algorithm is \( O(\ell \log \ell) \), where \( \ell \) is the number of sufficient statistics involving negative re-
relationships. In other words, the number of table operations is nearly linear in the size of the required output.

Evaluation. We evaluate the Möbius Join algorithm by computing contingency tables for seven real-world databases. The observed 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 positive and negative relationship features enhance different types of statistical analysis, as follows. (1) Feature selection: When provided with sufficient statistics for negative and positive relationships, a standard feature selection method selects relationship features for classification, (2) Association Rule Mining: A standard association rule learning method includes many association rules with relationship conditions in its top 20 list. (3) Bayesian network learning. A Bayesian network provides a graphical summary of the probabilistic dependencies among relationships and attributes in a database. On the two databases with the most complex schemas, enhanced sufficient statistics lead to a clearly superior model (better data fit with fewer parameters). This includes a database that is an order of magnitude larger than the databases for which graphical models have been learned previously [11].

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.

We contribute open-source code that implements the Möbius Join. All code and datasets are available on-line [3]. Our implementation makes extensive use of RDBMS capabilities. Like the BayesStore system [15], our system treats statistical concepts. A domain or population is a set of individuals. Individuals are denoted by lower case expressions (e.g., bob). A functor represents a mapping \( f : \mathcal{P}_1, \ldots, \mathcal{P}_n \rightarrow V_f \) 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 \( V_f = \{T,F\} \), 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 \( f(X_1, \ldots, X_n) \), where each \( X_i \) is a first-order variable [7]. Each first-order variable is associated with a population/type.

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-
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 Möbius Join algorithm computes contingency tables by reusing the results for smaller relationships for larger relationship chains.

A relationship variable set is a chain if it can be ordered as a list \([R_1(\tau_1), \ldots, R_k(\tau_k)]\) such that each relationship variable \(R_{i+1}(\tau_{i+1})\) shares at least one first-order variable with the preceding terms \(R_1(\tau_1), \ldots, R_i(\tau_i)\). 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

\[\text{Registration}(S, C), \ RA(P, S)\]

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

\[\text{Registration}(S, C), \ RA(P, S), \ Teaches(P, C)\]

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 each relationship chain \(R\) (Res for short) is associated a \(ct\)-table \(ct_R\). The variables in the \(ct\)-table \(ct_R\) comprise the relationship variables in \(R\), and the unary/binary descriptive attributes associated with each of the relationships. To define these, we introduce the following notation (cf. Table 1).

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

In this notation, the variables in the \( ct \)-table \( ct_R \) are denoted as \( RU_1\text{Atts}(R) \). The goal of the Möbius Join algorithm is to compute a contingency table for each chain \( R \). 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 \( RA(P, S) \) chain.

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

Even more efficient than SQL count queries is the Tuple ID propagation method, a Möbius Join method for computing query counts with positive relationships only [16]. 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.


We describe a Virtual Join algorithm that computes the required sufficient statistics without materializing a cross product of entity sets. 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 identity 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 \( \phi \). This is the standard relational algebra operation except that the selection condition \( \phi \) may not involve the count column.

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

```sql
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

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

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

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

4.1.2 Binary Operators

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

**Cross Product** The cross-product of \( ct_1(U), ct_2(V) \) 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:

```sql
SELECT (ct_1, count + ct_2, count) AS count, U, V
FROM ct_1, ct_2
```

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

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

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, assuming that a sort-merge join is used [14], a standard analysis shows that the cost of a sort-merge join is size(\text{table1}) + size(\text{table2})+ the cost of sorting both tables.

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.
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 [5], we use a “don’t care” value $\ast$ to indicate that a query does not specify the value of a node. For instance, the query $R_1 = T, R_2 = \ast$ is equivalent to the query $R_1 = T$.

**Proposition 1.** Let $R$ be a relationship variable and let $R$ 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 $X_1, \ldots, X_l$ be the first-order variables that appear in $R$ but not in $Vars$, where $l$ is possibly zero. Then we have

$$ct(Vars \cup 1Atts(R)|R = T, R = F) = \prod_{i=1}^{l} ct(X_i)$$

where $ct(X_i)$ is the count of $X_i$ in the $ct_T$ table.

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

The proof is available at [8]. Figure 5 illustrates Equation (1). The construction of the $ct_T$ table in Algorithm 1 provides pseudo-code for applying Equation (1) to compute a complete $ct_T$ table, given a partial table where a specified relationship variable $R_{pivot}$ is $T$. The $ct_T$-table operations are implemented using dynamic SQL queries as shown. Lists of column names are abbreviated as shown and also as follows.

Algorithm 1:

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

**Input:** Two conditional contingency tables $ct_T := ct(Vars, 2Atts(R_{pivot})|R_{pivot} = T, R = T)$ and $ct_* := ct(Vars|R_{pivot} = \ast, R = T)$.

**Precondition:** The set $Vars$ does not contain the relationship variable $R_{pivot}$ nor any of its descriptive attributes $2Atts(R_{pivot})$.

**Output:** The conditional contingency table $ct(Vars, 2Atts(R_{pivot})|R_{pivot} = \ast, R = T)$.

1. $ct_T := ct_* - \pi_{Vars} ct_T$.
   {Implements the algebra Equation 1 in proposition 1.}
2. $ct_T^F := extend ct_T$ with columns $R_{pivot}$ everywhere false and $2Atts(R_{pivot})$ everywhere n/a.
3. $ct_T^T := extend ct_T$ with columns $R_{pivot}$ everywhere true.
4. return $ct_T^F \cup ct_T^T$.
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, Algorithm 1 applies Equation (1) with a condition that lists a set of relationship variables fixed to be true. Figure 5 illustrates the Pivot computation for the case of only one relationship. Algorithm 2 shows how the Pivot operation can be applied repeatedly to find all contingency tables in the relationship lattice. Figures 4 and 6 provide illustration. The outline of this computation is as follows.

 Initialization. Compute $ct$-tables for entity tables. Compute $ct$-tables for each single relationship variable $R$, conditional on $R = T$. 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 Möbius Join (MJ) algorithm is that it avoids materializing the cross product of entity tuples. The algorithm accesses only existing tuples, never constructs nonexistent 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 Möbius Join algorithm in terms of the size of its output: the number of sufficient statistics that involve negative relationships.

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

\[ \#ct\text{-ops} = O(r \cdot \log \frac{r}{\alpha}) \]

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

The proof is available at [8]. Since the time cost of any algorithm must be at least as great as the time for writing the output, which is at least as great as $r$, the Möbius Join algorithm adds at most a logarithmic factor to this lower bound. This means that if the number $r$ of sufficient statistics is a feasible 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.

<table>
<thead>
<tr>
<th>lines</th>
<th>Operation</th>
<th>Resulting ct-table</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Reg$(S,C) = T, RA(P,S) = T$</td>
<td>Current ct</td>
</tr>
<tr>
<td>13-14</td>
<td>$i = 1, Reg(S,C) = *, RA(P,S) = T$</td>
<td>ct_i</td>
</tr>
<tr>
<td>20</td>
<td>PIVOT</td>
<td>Current ct</td>
</tr>
<tr>
<td>16-18</td>
<td>$i = 2, RA(P,S) = *$</td>
<td>ct_i</td>
</tr>
<tr>
<td>20</td>
<td>PIVOT</td>
<td>Final ct-table for Reg$(S,C), RA(P,S)$</td>
</tr>
</tbody>
</table>

Figure 6: Illustrates the relationship chain loop of Algorithm 2 (lines 11-21) for the chain $R = Reg(S,C), RA(P,S)$. This loop is executed for each relationship chain at each level.

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 [3].

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Relationship Tables/ Total</th>
<th>#Self Relationships</th>
<th>#Tuples</th>
<th>#Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movielens</td>
<td>1 / 4</td>
<td>0</td>
<td>1,090,051</td>
<td>7</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>2 / 4</td>
<td>0</td>
<td>19,340</td>
<td>11</td>
</tr>
<tr>
<td>Financial</td>
<td>3 / 7</td>
<td>0</td>
<td>229,952</td>
<td>16</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>3 / 7</td>
<td>0</td>
<td>12,927</td>
<td>19</td>
</tr>
<tr>
<td>IMDb</td>
<td>4 / 7</td>
<td>0</td>
<td>1,399,134</td>
<td>17</td>
</tr>
<tr>
<td>Mondial</td>
<td>2 / 4</td>
<td>1</td>
<td>58,0</td>
<td>18</td>
</tr>
<tr>
<td>UW-CSK</td>
<td>2 / 4</td>
<td>2</td>
<td>712</td>
<td>14</td>
</tr>
</tbody>
</table>

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

5.1 Datasets

We used seven benchmark real-world databases. For detailed descriptions and the sources of the databases, please see reference [11]. 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 [11]). IMDB is the largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. It combines the MovieLens database\(^1\) with data from the Internet Movie Database (IMDB)\(^2\) following [6].

5.2 Contingency Tables With Negative Relationships: Cross Product vs. Möbius Join

In this subsection we compare two different approaches for constructing the joint contingency tables for all variables together, for each database: Our Möbius Join algorithm (MJ) vs. materializing the cross product (CP) of the entity tables for each first-order variable (primary keys). Cross-checking the MJ contingency tables with the cross-product continuing

\(^1\)www.grouplens.org, 1M version
\(^2\)www.imdb.com, July 2013
dexes on the both methods used a B+tree index built on each column computation time for all statistics. For faster processing, the largest database IMDB. the number of statistics in the using an SQL query. The ratio of the cross product size to vs. the CP approach. The cross product was materialized with an infeasible amount of space resources. The compression the contingency table for each the number of statistics in the -table provides a substantial compression of the statistical Computation Time.

Table 3: Constructing the contingency table for each dataset. M = million. N.T. = non-termination. Compress Ratio = CP-#tuples/#Statistics.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MJ-time(s)</th>
<th>CP-time(s)</th>
<th>CP-#tuples</th>
<th>#Statistics</th>
<th>Compress Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>2.94</td>
<td>703.99</td>
<td>23M</td>
<td>292</td>
<td>33,053.34</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>1.09</td>
<td>1096.00</td>
<td>3M</td>
<td>1.83</td>
<td>195.00</td>
</tr>
<tr>
<td>Financial</td>
<td>1742.93</td>
<td>N/A</td>
<td>1999996.56M</td>
<td>3111381</td>
<td>99,469,255.50</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>3586.56</td>
<td>N/A</td>
<td>17.86M</td>
<td>12374832</td>
<td>1342.79</td>
</tr>
<tr>
<td>IMDb</td>
<td>7467.85</td>
<td>N/A</td>
<td>9890417.86M</td>
<td>16558436</td>
<td>322740092.08</td>
</tr>
<tr>
<td>Movielens</td>
<td>3214348</td>
<td>192.41</td>
<td>15M</td>
<td>1940278</td>
<td>256</td>
</tr>
<tr>
<td>UW-CSE</td>
<td>184</td>
<td>390.30</td>
<td>10M</td>
<td>2828</td>
<td>3097.32</td>
</tr>
</tbody>
</table>

Table 3: Constructing the contingency table for each dataset. M = million. N.T. = non-termination. Compress Ratio = CP-#tuples/#Statistics.

gency tables confirmed the correctness of our implementation. Table 3 compares the time and space costs of the MJ vs. the CP approach. The cross product was materialized using an SQL query. The ratio of the cross product size to the number of statistics in the -table measures how much compression the -table provides compared to enumerating the cross product. It shows that cross product materialization requires an infeasible amount of space resources. The -table provides a substantial compression of the statistical information in the database, by a factor of over 4,500 for the largest database IMDB.

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

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

Table 4: Number of Sufficent Statistics for Link Analysis On and Off. Extra Time refers to the total MJ time (Table 3 Col.2) minus the time for computing the positive statistics.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Link On</th>
<th>Link Off</th>
<th>#extra statistics</th>
<th>extra time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>252</td>
<td>210</td>
<td>42</td>
<td>0.27</td>
</tr>
<tr>
<td>Mutageness</td>
<td>1.03</td>
<td>0.96</td>
<td>1.096</td>
<td>0.99</td>
</tr>
<tr>
<td>Financial</td>
<td>3.01</td>
<td>8.73</td>
<td>3.084</td>
<td>1416.21</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>12374832</td>
<td>2.487</td>
<td>12374832</td>
<td>3555.51</td>
</tr>
<tr>
<td>IMDb</td>
<td>15,538,430</td>
<td>1.098,132</td>
<td>14,440,298</td>
<td>4558.62</td>
</tr>
<tr>
<td>Mondial</td>
<td>1,746,870</td>
<td>0</td>
<td>1,746,870</td>
<td>1112.31</td>
</tr>
<tr>
<td>UW-CSE</td>
<td>2,828</td>
<td>2</td>
<td>2,826</td>
<td>3.41</td>
</tr>
</tbody>
</table>
5.3 Contingency Tables with Negative Relationships vs. Positive Relationships Only

In this section we compare the time and space costs of computing both positive and negative relationships, vs. positive relationships only. 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^+ 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 off statistics is the number of Extra Statistics. The Extra Time column shows how much time the MJ algorithm requires to compute the Extra Statistics after the contingency tables for positive relationships are constructed using SQL joins. As Figure 7 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 8 shows that most of the MJ 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: Möbius Join Extra Time (s)

5.4 Cross-table Association Rules and Negative Relationships

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], given a contingency table. The idea is that if the existence of relationships is relevant to classification, then there should be a difference between the set selected with link analysis on and that selected with link analysis off. We measure how different two feature sets are by 1-Jaccard’s coefficient:

\[
\text{Distinctness}(A, B) = 1 - \frac{A \cap B}{A \cup B}
\]

Table 5: Selected Features for Target variables for

\[
\begin{array}{cccc}
\text{Dataset} & \text{Target variable} & \text{# Selected Attributes} & \text{Distinctness} \\
\hline
\text{MovieLens} & \text{Horror(M)} & 2 / 0 & 0.0 \\
\text{Mutagenesis} & \text{r1da(M)} & 3 / 0 & 0.0 \\
\text{Financial} & \text{balance(T)} & 3 / 1 & 1.0 \\
\text{Hepatitis} & \text{sex(D)} & 1 / 1 & 0.5 \\
\text{IMDB} & \text{avg_revenue(D)} & 5 / 1 & 1.0 \\
\text{Medicaid} & \text{percentage(C)} & 4 / 0 & 1.0 \\
\text{UW-LSE} & \text{courseLevel(C)} & 4 / 2 & 1.0 \\
\end{array}
\]

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

While Table 5 provides evidence that relationship features are relevant to the class label, it is not straightforward to evaluate their usefulness by adding them to a relational classifier. The reason for this is that relational classification requires some kind of mechanism for aggregating/combining information from a target entity’s relational neighborhood. There is no standard method for performing this aggregation [1], so one needs to study the interaction of the aggregation mechanism with relationship features. We leave for future work experiments that utilize relationship features in combination with different relational classifiers.

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. An example of a cross-table association rule for Financial is

\[
\text{statement_freq.}(\text{Acc}) = \text{monthly} \rightarrow \text{HasLoan(Acc, Loan)} = \text{T.}
\]

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 variable is relevant for other features, it should appear in an association rule. With link analysis off, all relationship variables always have the value T, so they do not appear in any association rule. 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 features with link analysis on, out of the top 20 rules. In all cases, a majority of rules utilize relationship variables, in Mutagenesis and IMDB all of them do.

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], given a contingency table. The idea is that if the existence of relationships is relevant to classification, then there should be

\[
y = 0.0003x + 196.56 \\
R^2 = 0.976
\]
6.3 Learning Bayesian Networks

Our most challenging application is constructing a Bayesian network (BN) for a relational database. For single-table data, Bayesian network learning has been considered as a benchmark application for precomputing sufficient statistics [5, 4]. 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 [13, 15]. 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.

We used the previously existing learn-and-join method (LAJ), which is the state of the art for Bayes net learning in relational databases [11]. The LAJ method takes as input a contingency table for the entire database, so we can apply it with both link analysis on and off, to obtain two different BN 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. Structure learning is fast, even for the largest contingency table IMDB (less than 10 minutes run-time). With link analysis on, structure learning takes more time as it processes more information. In both modes, the run-time for building the contingency tables (Table 3) dominates the structure learning cost. For the Mondial database, there is no case where all relationship variables are simultaneously true, so with link analysis off the contingency table is empty.

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 \( L(\hat{G}, d) \), where \( \hat{G} \) is the BN \( G \) with its parameters instantiated to be the maximum likelihood estimates given the dataset \( d \), and the quantity \( L(\hat{G}, d) \) is the log-likelihood of \( \hat{G} \) on \( d \). We use the relational log-likelihood score defined in [10], 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 link relationship variables—R2R—and that link attribute variables to relationships—A2R.

### Table 6: Number of top 20 Association Rules that utilize relationship variables.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Link Analysis On</th>
<th>Link Analysis Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>1.55</td>
<td>1.34</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>1.78</td>
<td>1.90</td>
</tr>
<tr>
<td>Financial</td>
<td>36.31</td>
<td>3.17</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>416.90</td>
<td>3.97</td>
</tr>
<tr>
<td>IMDB</td>
<td>56.64</td>
<td>26.16</td>
</tr>
<tr>
<td>Mondial</td>
<td>139.56</td>
<td>97.74</td>
</tr>
<tr>
<td>UW-CSE</td>
<td>2.29</td>
<td>2.17</td>
</tr>
</tbody>
</table>

### Table 7: Model Structure Learning Time in seconds.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Link Analysis Off</th>
<th>Link Analysis On</th>
<th>#Parameter</th>
<th>R2R</th>
<th>A2R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movielens</td>
<td>N/A</td>
<td>-6.68</td>
<td>164</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>N/A</td>
<td>-5.13</td>
<td>499</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Financial</td>
<td>-10.74</td>
<td>11,572</td>
<td>2433</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>-15.61</td>
<td>962</td>
<td>569</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>IMDB</td>
<td>-13.65</td>
<td>181,896</td>
<td>60,589</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Mondial</td>
<td>N/A</td>
<td>-18.2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UW-CSE</td>
<td>-6.68</td>
<td>305</td>
<td>241</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### 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 best-known are
ADtrees [5]. An 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. Schulte et al. review previous methods for computing statistics with negative relationships [12]. They show that the fast Möbius transform can be used in the case of multiple negative relationships. Their evaluation considered only Bayes net parameter learning with only one relationship. We examined computing joint sufficient statistics over the entire database. Other novel aspects are the ct-table operations and using the relationship chain lattice to facilitate dynamic programming.

8. CONCLUSION

Utilizing the information in a relational database for statistical modelling and pattern mining requires fast access to multi-relational sufficient statistics, that combine information across database tables. We presented an efficient dynamic program that computes sufficient statistics for any combination of positive and negative relationships, starting with a set of statistics for positive relationships only. Our dynamic program performs a virtual join operation, that counts the number of statistics in a table join without actually constructing the join. We showed that the run time of the algorithm is $O(r \log r)$, where $r$ is the number of sufficient statistics to be computed. The computed statistics are stored in contingency tables. We introduced contingency table algebra, an extension of relational algebra, to elegantly describe and efficiently implement the dynamic program. Empirical evaluation on seven benchmark databases demonstrated the scalability of our algorithm; we compute sufficient statistics with positive and negative relationships in databases with over 1 million data records. Our experiments illustrated how access to sufficient statistics for both positive and negative relationships enhances feature selection, 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 Möbius 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 prespecified relatively small relationship chain length. Another possibility is to use postcounting [4]: 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 Möbius 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.

Acknowledgments

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.

9. REFERENCES