Exploring Implicit Relationships In a Relational Database *

Andy Huang, Qiang Xue

May 2, 2002

1 Introduction

The Google search engine [BP98] has brought the enormous web pages an order that well reflects the degree of relevance and importance of the web pages with respect to search keywords. The technique used to determine the order is called PageRank [PBMW97], which origins from the idea of academic citation: a paper should receive more attention if it is cited by many (important) papers. By analyzing the link structures among the web pages which resemble the paper citations, a global order measuring the importance can be computed for all the web pages.

Inspired by the idea of PageRank, in last semester we proposed to bring to relational databases the similar global order, TupleRank, which represents the relative importance of each tuple in a relational database. In particular, we computed TupleRank based on the primary/foreign key relations among tuples and observed very interesting and promising applications. The successful use of TupleRank relies on well-defined data schema and highly related data entries.

However, as we have noticed in real applications, primary/foreign key relations only capture part of the real relationships between data entries. Many relationships are modeled through creating views and running selection queries on tables and views. For example, a view or selection query joining tables `EMPLOYEE` and `DEPARTMENT` implies the `...WORK AT...` relationship between data entries in these two tables[1]. In order to produce more precise PageRank, we need to explore relationships that are hidden among data entities and not modeled as primary/foreign key relations. We call such relationships as *implicit relationships*, and in this report we introduce two approaches that can explore implicit relationships.

*CPS 296 Spring 2002 Course Project Final Report

[1]Although it is possible to use primary/foreign key relation to model such relationships, in real applications the relationships may not be explicitly modeled.

2 TupleRank Algorithm

To include the new relationships induced by database queries, we slightly modify the computation of TupleRank by introducing *weights* to each relationship between tuples. A weight represents the probability for one tuple to be related with another one. It is a measure of the credibility of relationships: Tuples that are related by primary/foreigh key constraints will be assigned the maximal weight of 1, meaning the most reliable (strongest) relationship between two tuples, while a weight of 0 means no relationship at all between two tuples. Given N tuples we extend the definition of connectivity matrix A by,

$$A_{ij} = \frac{w_{ij}}{\sum_{k=1}^{N} w_{kj}}, \tag{1}$$

where w_{ij} is the weight for the relationship between tuple i and j. As before, an iterative procedure is used to determine the tuple rank $\{r_i\}$,

$$\mathbf{r}^{(k+1)} = d\mathbf{A}\mathbf{r}^{(k)} + (1-d), \tag{2}$$

in which $\mathbf{r}^{(k)} = [r_1, r_2, \ldots, r_N]^T$ is the rank vector for the k-th iteration, and d is a damping factor between 0 and 1 [PBMW97]. The iteration is ended when $\|\mathbf{r}^{(k+1)} - \mathbf{r}^{(k)}\|$ is small enough.

3 Exploring Implicit Relationships

Consider a relational database D with tables $\{T_i\}_{i=1:N}$, each consisting of N_{T_i} attributes $\{a_j^{T_i}\}_{j=1:N_{T_i}}$. Assume $Q = \{Q_k\}_{k=1:M}$ is a set of queries that have been executed on D. We present two methods to determine the weights: query-based and histogram-based approaches.

3.1 Query-based Approach

An important observation is that *two data entities are considered related if they appear in a single row of some query result.* More formally, we claim that an implicit relationship exists between two tuples if each one has some attributes that appear in the same row of the result of a query.

Based on the above observation we can determine implicit relationships in a relational database by examining the results of the queries that are executed on the database. If two tuples are joined together by some query then we assign a weight of 1 between them, otherwise the weight is zero.

In implementation, we can record all the queries that have been executed and re-execute them at a later time to avoid lowering query performance. Or we may simply construct the weights when the query results are being retrieved by users. For the latter method, we need to provide wrapper functions for all database related functions. We can keep the weights in a database table and when computing TupleRanks they are retrieved to construct the connectivity matrix.

The primary/foreign key relations can also be captured by this approach. From the metadata of the database, we can determine what are the primary keys and foreign keys, and construct a set of queries that join tables based on the primary/foreign key relations. Using the dynamic approach described above, we can thus find out the weights of the relationships.

Although this method is simple and precise, it is very expensive because we have to examine each row of the query results.

It is also possible to find out implicit relationships using data mining techniques. There are numerous research efforts on this topic (e.g. [AS94, SON95, AY98, LHM99]). However, data mining techniques still require considerable storage and computational resources.

3.2 Histogram-based Approach

To avoid the high complexity of the query-based approach, we present an algorithm that computes approximate weights based on database histograms.

Assume two tables T_a and T_b have been joined by a query. We need to determine the weights between tuples in T_a and tuples in T_b. Rather than re-executing the join query as in query-based approach, we estimate the probability of a tuple in T_a to be joined with another tuple in T_b. According to the definition in section 2, the probability value will be the value of the weight. We will compute the estimations from the information provided by database histograms.

Histograms are widely used in most commercial database systems for summarizing the contents of large relations and efficiently estimating the sizes of the query results for use in query optimizers. A *one-dimensional histogram* on attribute X is constructed by partitioning the range of X into mutually disjoint subsets called *buckets*. Each bucket can be represented by (B, f_B), where B is a range of X and f_B is the number of tuples whose X attribute falls into the range B[2]. A *multi-dimensional histogram* consists of buckets that represent the joint distribution of multiple attributes of a single table.

In the following, we will assume that each table has at most one one-dimensional histograms. Several one-dimensional histograms can be considered as a special case of multi-dimensional histograms.

3.2.1 One-dimensional Case

Assume that tables T_a and T_b have ever been joined by attribute X, and each table has built a histogram on X whose values are uniformly distributed within buckets. Let $t_a \in T_a$ be a randomly selected tuple in bucket (B_a, f_{B_a}), and (B_b, f_{B_b}) is a bucket of T_b's histogram. If the possible values of X are *enumerable*, then the probability for t_a to join with a tuple in bucket (B_b, f_{B_b}) is,

$$p_{ab} = \frac{|B_a \cap B_b|}{|B_a| \cdot |B_b|}, \tag{3}$$

where $|\cdot|$ represents the size of a set. We thus have a network between the two buckets as shown in figure 1.

The network indicates that the tuples within the same bucket have the same TupleRank. Therefore, we can simplify the network by treating each bucket as a node and assigning weights between

[2]Such a histogram is called one-dimensional histogram. A multi-dimensional histogram consists of buckets that record the combinational data distribution of several attributes. The above definitions are based on continuous value assumption [SAC+79].

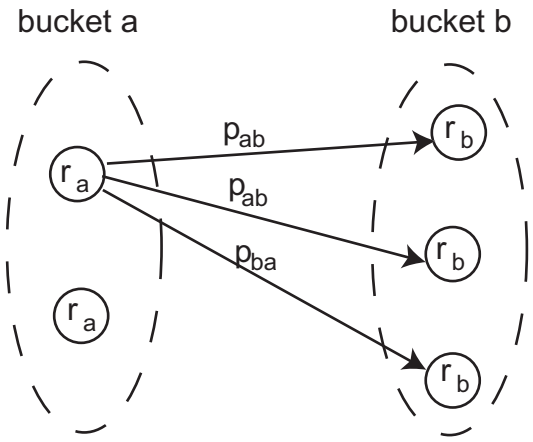

Figure 1: The relationship network between two buckets. The eclipses are buckets containing tuples represented by small circles. We need to determine the TupleRanks r_a and r_b of the tuples. In the graph we only depict possible relationships from a tuple in B_a to tuples in B_b. The two groups of tuples should be full-connected.

buckets (see figure 2). For example, the weight from bucket a to b is computed by

$$w_{ab} = f_b p_{ab},$$

where f_a is the number of tuples in bucket a.

Because each table has at most one one-dimensional histogram, we can estimate the TupleRank of a tuple by solving the network as shown in figure 2. If a table has no histogram, we simply assign a default TupleRank to its tuples.

3.2.2 Multi-dimensional Case

Assume a multi-dimensional histogram is built on attributes $X_1, X_2, \ldots, X_n$. We can still use (B, f_B) to represent a bucket, where $B = R_{X_1} \times R_{X_2} \times \ldots \times R_{X_n}$ and R_{X_k} is a range of X_k.

Let $t_a \in T_a$ be a randomly selected tuple in bucket (B_a, f_{B_a}), and (B_b, f_{B_b}) is a bucket of T_b's histogram. Without loss of generality, we can assume $B_a = R_{a,X_1} \times R_{a,X_2} \times \ldots \times R_{a,X_n}$, and $B_b = R_{b,X_1} \times R_{b,X_2} \times \ldots \times R_{b,X_k} \times R_{b,Y_{k+1}} \times \ldots \times R_{a,Y_m}$, in which X_i and Y_i are attributes, and R means the range of an attribute.

If the possible values of X_i and Y_i are all *enumerable*, then the probability for t_a to join with at least one tuple in bucket (B_b, f_{B_b}) is,

$$p_{ab} = \frac{J_1 J_2}{|B_a| \cdot |B_b|}, \tag{4}$$

where

$$|B| = |R_{X_1}| \times |R_{X_2}| \times \ldots \times |R_{X_n}|$$

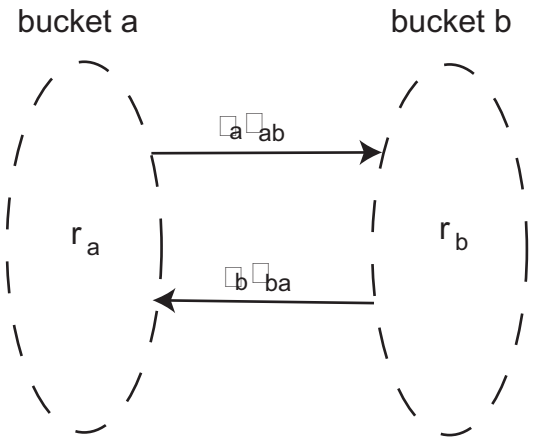

Figure 2: Simplified relationship network. Each bucket is treated as a network node and we modify the weights accordingly.

$$J_1 = |R_{a,X_1} \cap R_{b,X_1}| \times \ldots \times |R_{a,X_1} \cap R_{b,X_k}| \times |R_{a,X_{k+1}}| \times \ldots \times |R_{a,X_n}|$$
$$J_2 = |R_{b,Y_{k+1}}| \times \ldots \times |R_{b,Y_m}|$$

We can then get the relationship network the same as in figure 1. Using the simplified network as in figure 2, we can determine the TupleRanks.

3.3 Combining Frequencies of Database Queries

A data entry that has been selected more frequently than another one should be considered more important. Similarly concepts are also used in many Web search engines that combine factors, such user ratings and hit rates, into their final order of the Web pages. It is possible to record the frequency of a query being executed in histograms. Therefore, we can simply add to the computed TupleRanks a number that reflects the frequencies that tuples have been selected.

3.4 Computing Histograms

Nearly all commercial database systems maintain histograms by computing histograms periodically (e.g. every night). It is obvious that such an approach is computationally inefficient and may cause low accuracy. People have thus investigated computing histograms incrementally based on a self-tuning scheme [GMP97, AC99].

Their basic idea is as follows,

- Initially assume uniform distribution of attribute X on the whole table and partition the range of X into several subsets;
- When a query is executed, we use the tuple count information provided by DBMS to determine the estimation error of our initial histogram;

- Buckets are thus refined and restructured to maintain the uniformity assumption.

Therefore, the accuracy of the histogram is improved incrementally as more queries are executed.

4 Implementation

We have selected IBM DB2 as our DBMS and built a relational database of the world geographic information, Mondial[3]. We choose Java and JDBC as our programming language and database connection links.

We have planned to test the feasibility of computing TupleRanks from histograms. Using the query-based approach as a standard, we will investigate the accuracy of histogram-based TupleRanks. Furthermore, we want to check the effectiveness of combining frequency information of database queries.

We have implemented most part of the planned work. However, the high computational cost for TupleRank has prevented us from conducting the scheduled experiments and delayed our overall progress.

5 Discussion and Conclusion

In this project, we have mainly devised a histogram-based approach to explore implicit relationships among relational databases. The results can be used to further our TupleRank algorithm.

Our algorithm is basically a tradeoff between accuracy and efficiency: histogram-based relationships are less determinant, but they can be computed faster than query-based relationships. Using coarser histograms will lead to faster computation yet lower accuracy.

In our algorithm we require that the ranges in consideration are enumerable. This is necessary because non-enumerable ranges will bring zero probability for two tuples to join together.

We require uniform distribution in our algorithm. Our algorithm implicitly assumes equality joins. Moreover, our approach of combining query frequency information seems awkward. Our future work will try to remove these constraints.

References

[AC99] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *SIGMOD 1999*, pages 181–192, 1999.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[3] http://www.informatik.uni-freiburg.de/∼may/Mondial/

[AY98] Charu C. Aggarwal and Philip S. Yu. Online generation of association rules. In *ICDE*, pages 402–411, 1998.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World Wide Web Conference*, Brisbane, Australia, 1998.

[GMP97] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Proc. 23rd Int. Conf. Very Large Data Bases, VLDB*, pages 466–475. Morgan Kaufmann, 25–27 1997.

[LHM99] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Knowledge Discovery and Data Mining*, pages 337–341, 1999.

[PBMW97] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Stanford Digital Libraries working paper 1997-0072*, 1997.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. T. Price. Access path selection in a relational database management system. In *Proc. of ACM SIGMOD Conf*, pages 23–34, 1979.

[SON95] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *The VLDB Journal*, pages 432–444, 1995.