Open access • Proceedings Article • DOI:10.1145/1247480.1247498

Effective keyword-based selection of relational databases — Source link

Bei Yu, Guoliang Li, Karen R. Sollins, Anthony K. H. Tung

Institutions: Tsinghua University, Massachusetts Institute of Technology

Published on: 11 Jun 2007 - International Conference on Management of Data

Topics: Search-oriented architecture, View, Database model, Query optimization and Database design

Effective Keyword-based Selection of Relational Databases

Bei Yu
National University of Singapore

Guoliang Li
Tsinghua University

Karen Sollins
MIT

Anthony K. H. Tung
National University of Singapore

ABSTRACT

The wide popularity of free-and-easy keyword based searches over World Wide Web has fueled the demand for incorporating keyword-based search over structured databases. However, most of the current research work focuses on keyword-based searching over a single structured data source. With the growing interest in distributed databases and service oriented architecture over the Internet, it is important to extend such a capability over multiple structured data sources. One of the most important problems for enabling such a query facility is to be able to select the most useful data sources relevant to the keyword query. Traditional database summary techniques used for selecting unstructured data sources developed in IR literature are inadequate for our problem, as they do not capture the structure of the data sources. In this paper, we study the database selection problem for relational data sources, and propose a method that effectively summarizes the relationships between keywords in a relational database based on its structure. We develop effective ranking methods based on the keyword relationship summaries in order to select the most useful databases for a given keyword query. We have implemented our system on PlanetLab. In that environment we use extensive experiments with real datasets to demonstrate the effectiveness of our proposed summarization method.

Categories and Subject Descriptors: H.2 [Database Management]: Miscellaneous

General Terms: Design

Keywords: keyword query, summarization, database selection

1. INTRODUCTION

Keyword search over structured data such as relational databases is an increasingly important capability [1, 2, 3, 13, 14, 19, 21], taking advantage of a combination of DB and IR techniques. While these projects focus on keyword-based query processing in a centralized database, the increasing deployment of P2P networks and service oriented architectures has made it equally important to extend such keyword-based search capabilities to distributed databases. Analogous to distributed IR systems [5, 12, 22, 30], keyword-based database selection is a critical step towards locating useful databases for answering a keyword query, and on which existing centralized keyword search methods can be directly applied.

For effective selection of useful data sources in distributed IR systems, a common approach is to summarize document collections with a list of keywords associated with some intra-collection (e.g., frequency) or inter-collection (e.g., inverse collection frequency (ICF) [5]) weightings. Data sources are then ranked by comparing the keyword queries with their summaries, which can be stored either at the data sources or the querying clients (allowing for different tradeoffs in terms of communication cost and workload size).

Summarizing a relational database with the simple keyword-list method as in IR systems is, however, inadequate for two reasons. First, relational tables in a database are typically normalized. Therefore, the keyword frequency statistics, which are used in most IR-based summaries for textual documents, cannot really measure the importance of keywords in a relational database. Consider the case where a keyword appears only once, and it is in a tuple that is referenced by many other tuples. Such a keyword is likely to be important since it is related to many other keywords in the connected tuples. Second, the results from a relational database with respect to a keyword query must take into account the number of join operations that must be done in order for all the keywords to appear in the result (often represented as an evaluation tree) [1, 3, 13, 14]. This can only be obtained if the relationship between keywords in the relational database is somehow captured in the summary.

For illustration, let us look at the two example databases $DB1$ and $DB2$ shown in Figure 1, in which the arrowed lines drawn between tuples indicate their connections based on foreign key references. Suppose we are given a keyword query $Q = \{multimedia,\ database,\ VLDB\}$. We can observe that $DB1$ has a good result to Q, which is the result of joining tuple t_1 with t_3. On the contrary, $DB2$ cannot provide relevant results to Q — there are no trees of connected tuples containing all the query keywords. But, if we evaluate the two databases for Q based on the keyword frequency

SIGMOD'07, June 12–14, 2007, Beijing, China.

style summaries (denoted as *KF-summary* in this paper, and *KF-summary(DB1)* = { $\cdots$ *multimedia:1, database:2, VLDB:1,* $\cdots$ }, and *KF-summary(DB2)* = { $\cdots$ *multimedia:3, database:3, VLDB:1,* $\cdots$ }), $DB2$ will be selected over $DB1$. Therefore, we can observe that the usefulness of a relational database in answering a keyword query is not only decided by whether it has all the query keywords, but more importantly, it depends on whether the query keywords can be *connected* meaningfully in the database.

In this paper, we define *keyword relationship* for representing such connections between keywords in a relational database and look at how summarizing keyword relationships can help us to effectively select relevant structured sources in a distributed setting. This work is part of our BestPeer project [25] for supporting P2P-based data sharing services. BestPeer is a P2P platform that can be configured to support either structured [18] and unstructured overlays, and it provides a set of tools for building data sharing applications.

We make the following contributions in this paper.

- We propose to look at the problem of structured data sources selection for keyword based queries. To the best of our knowledge, this is the first attempt to address this problem.
- We propose a method for summarizing the relationship between keywords in a relational database. The technique for generating the database summary can be done by issuing SQL statements and thus can be performed directly on the DBMS without modification to the database engine.
- We define metrics for effectively ranking source databases given a keyword query according to the keyword relationship summary.
- We implemented the system in real distributed settings on PlanetLab [7] and evaluate the effectiveness of the proposed summarization method with real datasets.

The rest of the paper is organized as follows. In Section 2, we present the way to discover the relationships between keywords in a relational database in order to effectively evaluate its usefulness in answering a given keyword query. We also show how to create the keyword relationship summary using SQL. In Section 3, we describe the metrics to rank databases based on the keyword relationship summaries. We present our experimental study in Section 4. Then, we discuss related work in Section 5 and conclude the paper and discuss our future work in Section 6.

2. SUMMARIZING A RELATIONAL DATABASE

We consider a set of relational databases $\{DB_1, DB_2, \cdots, DB_N\}$. Given a keyword query $Q = (k_1, k_2, \cdots, k_q)$, we would like to rank the databases based on their usefulness to answer query Q. Basically, a database is useful given Q if it has high quality results to the keyword query. Therefore, we measure the usefulness of a database DB to Q as the total score of the top-K results it can return, i.e.,

$$score(DB, Q) = \sum_{i=1}^{K} score(T_i, Q), \qquad (2\text{-}1)$$

where T_i is the i-th top result of DB given Q, and $score(T_i, Q)$ measures the relevance of T_i to Q.

Ideally, given a query Q, the databases should be ranked based on their scores calculated according to Equation 2-1, in order that the most useful databases can be selected to which the query will be forwarded. However, in a real distributed setting, it is not feasible to get the ideal score defined by Equation 2-1 for every database in the system, since it needs to execute the query over all the databases in the system. A feasible solution is to construct summaries for the source databases, and estimate the usefulness of them for a given keyword query by comparing the query with the summaries.

Therefore, the type of summary we need to construct in this case is highly dependent on how the relevancy or usefulness of the querying result is measured. It is necessary to answer this question by examining the results returned by a relational database to a given keyword query. An answer from a relational database to a keyword query is a minimal tree of tuples containing all the query keywords, where tuples in the tree are connected according to their relationships defined in the database schema, such as foreign key relationships. The foreign key relationships, the most common type of relationships between tuples, are resulted from the schema normalization, and hence they reflect the semantics of the database. Although we only consider foreign key relationships for simplicity of presentation, other types of relationships between tuples that are related to the semantics of the database could also be considered, such as inclusion relationships or any other kinds of implicit relationships. The number of tuples of the tuple tree (referred as *size* of the tree), reflecting the number of joins between keywords is inversely proportional to the relevance of the relationship. In other words, more distant relationships are reflective of weaker relationships connecting the tuples [13, 14, 19, 21].

To estimate this measure of relevancy, it is easy to see that a summary of the relationships between all pairs of keywords in each database must be obtained. Two keywords in a relational database, if they can be connected, are always related with a joining sequence of tuples where the two end tuples contain each of the two keywords. We define *distance* as the the number of join operations in a joining sequence of tuples. For example, the distance of a joining sequence $t_1 \bowtie t_2 \bowtie t_3$ is 2. For each single tuple, we define its distance as 0. Note that the distance of a joining sequence to connect two keywords is bounded by the number of tuples in the database, instead of the number of tables [14]. In a database, two keywords could be connected with different joining sequences at various distances. For example, in database $DB2$ of Figure 1, the two keywords, *multimedia* and *binder*, can be connected at 2 distance in two different ways. The joining sequences are $t_1 \bowtie t_5 \bowtie t_{12}$, and $t_4 \bowtie t_9 \bowtie t_{12}$. They can also be connected at 4 distance in one way, with the joining sequence $t_{15} \bowtie t_{10} \bowtie t_{t_1} \bowtie t_5 \bowtie t_{12}$.

Based on the observation above, we measure the strength of keyword relationships between each pair of different keywords according to the combination of two factors — the *proximity* factor and the *frequency* factor. The proximity factor is defined as a parameter that is inverse to the distance of the joining sequence that connects the two keywords. The frequency factor, with respect to a particular distance d, is the number of combinations of exactly $d + 1$

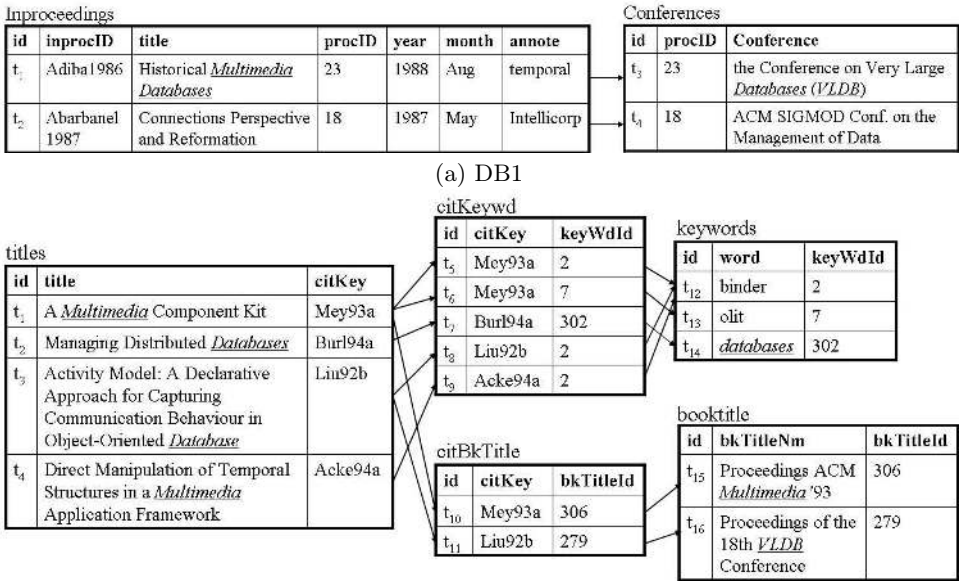


Figure 1: Example databases

number of distinct tuples that can be joined in a sequence to include the two keywords in the end tuples.

In this manner, each database is summarized by a list of keyword pairs that have significant relationship scores. We will present how to discover and create such relationships between keywords in the following two subsections.

2.1 KRM: The Keyword Relationship Matrix

We model each relational database DB as two matrices $\mathcal{D}(m \times n)$ and $\mathcal{T}(n \times n)$, where m is the number of distinct keywords contained in all the tuples $t \in DB$, and n is the total number of tuples.

The $\mathcal{D}$ matrix, illustrated as,

$$\mathcal{D} = (d_{ij})_{m\times n} = \begin{Bmatrix} & t_1 & t_2 & \cdots & t_n \\ k_1 & 1 & 0 & \cdots & 0 \\ k_2 & 0 & 1 & \cdots & 1 \\ \vdots & \vdots & & & \\ k_m & 1 & 0 & \cdots & 0 \end{Bmatrix}$$

represents the presence or absence of each keyword in each tuple in DB. This is closely to the term-document matrix of the vector space model (VSM) in IR literature, with the change that documents are replaced by tuples in a relational database, in our work. Although there are also various weighting schemes developed by IR and DB community for measuring the importance of the keywords in either documents or tuples [13, 21, 27], in our case we have simplified this to being only 0 or 1 for absence or presence.

The $\mathcal{T}$ matrix, shown below,

$$\mathcal{T} = (t_{ij})_{n\times n} = \begin{Bmatrix} & t_1 & t_2 & \cdots & t_n \\ t_1 & 0 & 1 & \cdots & 1 \\ t_2 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \\ t_n & 1 & 0 & \cdots & 0 \end{Bmatrix}$$

represents the relationships between tuples in a relational database, the most obvious kind being foreign key reference between tuples. The entry $\mathcal{T}[i, j]$ of 1 denotes that tuple t_i references (or is referenced by) tuple t_j $(1 \le i, j \le n)$.

$\mathcal{D}$ and $\mathcal{T}$ model the content and structure information in DB, respectively, and they will help us derive the relationships between every pair of keywords in DB, which is represented with another matrix $\mathcal{R}(m \times m)$, called the *keyword relationship matrix (KRM)*.

Definition 1. *Let δ be a user-supplied parameter denoting the maximum number of allowed join operations, and K be the maximum number of results expected from a database. For each distance d ($0 \le d \le \delta$), $\omega_d(k_i, k_j)$ is the frequency of d-distance joining sequences to connect the pair of keywords k_i and k_j. A* **Keyword Relationship Matrix (KRM)**, *$\mathcal{R}=(r_{ij})_{m*m}$, represents the relationships between any pair of two keywords in a relational database with respect to δ and K. When $\sum_{d=0}^{\delta} \omega_d(k_i, k_j) \le K$,*

$$\mathcal{R}[i, j] = r_{ij} = \sum_{d=0}^{\delta} \varphi_d * \omega_d(k_i, k_j),$$

in which φ_d is a function of d that measures the impact of d to the relationship between k_i and k_j; and otherwise when $\sum_{d=0}^{\delta} \omega_d(k_i, k_j) > K$, we have $\exists \delta' \le \delta$, $\sum_{d=0}^{\delta'} \omega_d(k_i, k_j) \ge K$ and $\sum_{d=0}^{\delta'-1} \omega_d(k_i, k_j) < K$,

$$\mathcal{R}[i, j] = r_{ij} = \sum_{d=0}^{\delta'-1} \varphi_d * \omega_d(k_i, k_j) + \varphi_{\delta'} * (K - \sum_{d=0}^{\delta'-1} \omega_d(k_i, k_j)).$$

It is obvious that when two keywords are further apart based on the number of join operations, the relationship between them is weaker. Accordingly, φ_d should be a monotonically decreasing function with respect to increasing d. We propose to set φ_d as

$$\varphi_d = \frac{1}{d+1}. \tag{2-2}$$

Note that φ_d could also be set differently based on specific requirements.

In this way, the KRM measures the total scores of up to top-K results within δ distance for each pair of keywords as query in a relational database, where each result, a joining sequence, has the score φ_d. A database with a higher relationship score for a given pair of keywords will generate better results. The reason we set an upperbound of the number of results, K, is to enable a user to control the quality of the results. For example, if for a pair of keywords k_1 and k_2, one database A has 5 joining sequences to connect k_1 and k_2 at 1 distance, and the other database B has 40 joining sequences to connect k_1 and k_2 at 4 distance. If $K = 40$, the score of the pair in A is $5 \times \frac{1}{2} = 2.5$, while the score of B is $40 \times \frac{1}{5} = 8$, as a result, we will choose B over A. However, one may very possibly prefer A to B because it has results with higher quality. If we decrease K to 10, the score of A is the same, but the score of B now becomes $10 \times \frac{1}{5} = 2$, such that A can be ranked higher than B. In general, K defines the number of top results users expected from a database.

2.2 Computation of KRM

We next look at how the KRM can be computed, i.e., $\omega_d(k_i, k_j)$ between every pair of keywords k_i and k_j in DB. As said, we can derive such information based on $\mathcal{D}$ and $\mathcal{T}$. We first define the *d-distance tuple relationship matrix* as follows.

DEFINITION 2. *The d-distance tuple relationship matrix, denoted as $\mathcal{T}_d(n \times n)$, is a symmetric matrix with binary entries, such that for any $1 \le i, j \le n$ and $i \ne j$, $\mathcal{T}[i,j] = \mathcal{T}[j,i] = 1$ if and only if the shortest joining sequence to connect the two tuples t_i and t_j is of distance d, and $\mathcal{T}_d[i,j] = \mathcal{T}_d[j,i] = 0$, otherwise.*

According to the definition, $\mathcal{T}_d$ actually records whether there is a shortest path with d hops between any pair of two tuples in a database, if we view the database as a graph in which the nodes are the tuples and the edges between nodes denote the reference relationships between tuples. Obviously, $\mathcal{T}_1 = \mathcal{T}$, and $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_\Delta$ is the transitive closure of $\mathcal{T}_1$, where Δ is the longest distance of the path between two tuples in the database graph. Taking database $DB2$ in Figure 1 as an example, its tuple relationship matrices $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are shown in Figure 2.

PROPOSITION 1. *Let $\mathcal{T}_{d_1}$ and $\mathcal{T}_{d_2}$ $(d_1 \ne d_2)$ be two tuple relationship matrices in a database. For any i and j, $i \ne j$, if $\mathcal{T}_{d_1}[i,j] = 1$, then $\mathcal{T}_{d_2}[i,j] = 0$.*

From Proposition 1, we derive matrices $\mathcal{T}_2$, $\mathcal{T}_3$, $\cdots$, inductively, based on $\mathcal{T}_1 = \mathcal{T}$.

PROPOSITION 2. *Given $\mathcal{T}_1 = \mathcal{T}$, and supposing $\mathcal{T}_d^* = \bigvee_{k=1}^{d} \mathcal{T}_k$, we have for all $1 \le i, j \le n$ and $i \ne j$,*

$$\mathcal{T}_{d+1}[i,j] = \begin{cases} 0 & \text{if } \mathcal{T}_d^*[i,j] = 1, \\ 1 & \text{if } \mathcal{T}_d^*[i,j] = 0 \text{ and } \exists r(1 \le r \le n), \mathcal{T}_d[i,r] * \mathcal{T}_1[r,j] = 1. \end{cases}$$

PROOF. Suppose we already have $\mathcal{T}_d$ $(d \ge 1)$, and we can get $\mathcal{T}_d^* = \bigvee_{k=1}^{d} \mathcal{T}_k$.

When $\mathcal{T}_d^*[i,j] = 1$, it implies $\exists k(1 \le k \le d), \mathcal{T}_k[i,j] = 1$. According to Proposition 1, for any $1 \le k \le d$, when $\mathcal{T}_k[i,j] = 1$, $\mathcal{T}_{d+1}[i,j] = 0$. Therefore, $\mathcal{T}_d^*[i,j] = 1$ implies $\mathcal{T}_{d+1}[i,j] = 0$.

When $\mathcal{T}_d^*[i,j] = 0$, it means tuples t_i and t_j must be connected with more than d number of connections. If there exists another tuple t_r such that $\mathcal{T}_d[i,r] * \mathcal{T}_1[r,j] = 1$, it means that $\mathcal{T}_d[i,r] = 1$ and $\mathcal{T}_1[r,j] = 1$, i.e., t_i and t_r can be connected with at least d connections, and there is a direct connection between t_r and t_j. Therefore, there must be at least $d+1$ connections between t_i and t_j with the route $t_i \to t_r \to t_j$, and consequently $\mathcal{T}_{d+1}[i,j] = 1$. □

Now we can derive the frequencies of every keyword pair in DB at various distances based on $\mathcal{D}$ and $\mathcal{T}$. In this case, $\mathcal{W}_d$ is the matrix of frequencies of keyword pairs at distance d.

PROPOSITION 3. *Let $\mathcal{W}_0 = \mathcal{D} \times \mathcal{D}^T$. ($\mathcal{D}^T$ represents the transposition of matrix $\mathcal{D}$.) We have*

$$\forall i, j, 1 \le i, j \le m \text{ and } i \ne j, \omega_0(k_i, k_j) = \mathcal{W}_0[i,j].$$

For $d \ge 1$, let $\mathcal{W}_d = \mathcal{D} \times \mathcal{T}_d \times \mathcal{D}^T$. We have

$$\forall i, j, 1 \le i, j \le m \text{ and } i \ne j, \omega_d(k_i, k_j) = \mathcal{W}_d[i,j].$$

PROOF. First, we prove $\mathcal{W}_0 = \mathcal{D} \times \mathcal{D}^T = \omega_0(k_i, k_j)$. For every $1 \le i, j \le m$,

$$\begin{aligned} \mathcal{W}_0[i,j] &= \sum_{k=1}^{n} D[i,k] * D^T[k,j] \\ &= \sum_{k=1}^{n} D[i,k] * D[j,k], \end{aligned}$$

which is the frequency of pair of k_i and k_j at distance 0, i.e., they appear in the same tuples. Therefore, $\omega_0(k_i, k_j) = \mathcal{W}_0[i,j]$.

Next, we prove $\mathcal{W}_d = \mathcal{D} \times \mathcal{T}_d \times \mathcal{D}^T = \omega_d(k_i, k_j)$. Let $\mathcal{M} = \mathcal{D} \times \mathcal{T}_d$. So for every $1 \le i \le m$ and $1 \le r \le n$,

$$\mathcal{M}[i,r] = \sum_{k=1}^{n} \mathcal{D}[i,k] * \mathcal{T}_d[k,r].$$

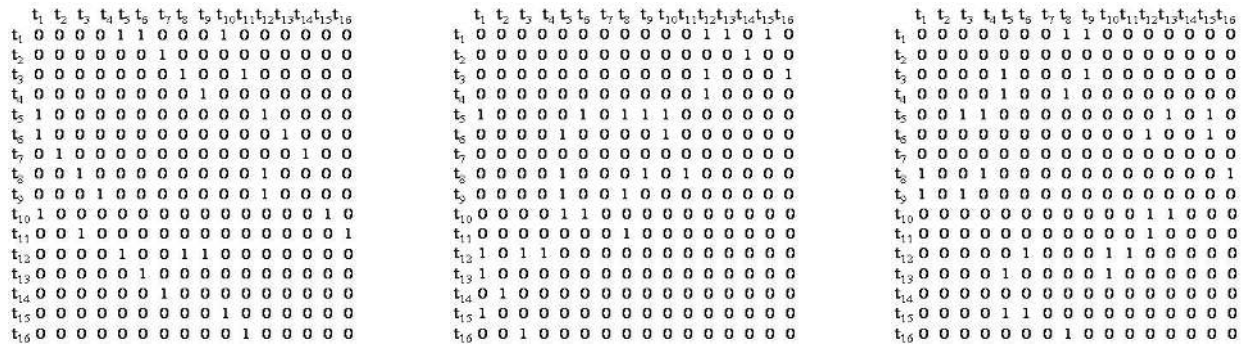
Then $\mathcal{W}_d = \mathcal{M} \times \mathcal{D}^T$, i.e., for every $1 \le i, j \le m$,

$$\begin{aligned} \mathcal{W}_d[i,j] &= \sum_{r=1}^{n} \mathcal{M}[i,r] * \mathcal{D}^T[r,j] \\ &= \sum_{r=1}^{n} \sum_{k=1}^{n} \mathcal{D}[i,k] * \mathcal{T}_d[k,r] * \mathcal{D}^T[r,j] \\ &= \sum_{r=1}^{n} \sum_{k=1}^{n} \mathcal{D}[i,k] * \mathcal{T}_d[k,r] * \mathcal{D}[j,r] \end{aligned}$$

Since $\mathcal{T}_d[k,r] = 1$ indicates there needs at least d connections to connect tuples t_k and t_r, and $D[i,k] = 1, D[j,r] = 1$ represent the presence of keywords k_i and k_j in t_k and t_r, respectively, $\sum_{r=1}^{n} \sum_{k=1}^{n} \mathcal{D}[i,k] * \mathcal{T}_d[k,r] * \mathcal{D}[j,r]$ is the number of combinations of tuples that can be joined to include k_i and k_j at distance d. Therefore, $\mathcal{W}_d = \mathcal{D} \times \mathcal{T}_d \times \mathcal{D}^T = \omega_d(k_i, k_j)$. □

In Figure 3, we show the frequencies of the pairs of query keywords at various distances of the two example databases $DB1$ and $DB2$ in Figure 1. By comparing Figure 3(a) and 3(b), we can easily tell that the query keywords are related more closely in $DB1$ than in $DB2$.

Finally, given a maximum distance parameter δ and the upperbound of the number of desired results, K, the relationship score between each pair of keywords k_i and k_j, $rel(k_i, k_j)$, in a database DB can be computed according



$\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_3$

Figure 2: Tuple relationship matrices of $DB2$ in Figure 1

keyword pair	d = 0	d = 1	d = 2	d = 3	d = 4
database:multimedia	1	1	-	-	-
multimedia:VLDB	0	1	-	-	-
database:VLDB	1	1	-	-	-

(a) Frequencies of keyword pairs in $DB1$

keyword pair	d = 0	d = 1	d = 2	d = 3	d = 4
database:multimedia	0	0	0	0	2
multimedia:VLDB	0	0	0	0	0
database:VLDB	0	0	1	0	0

(b) Frequencies of keyword pairs in $DB2$

Figure 3: Compare the frequencies of keyword pairs of $DB1$ and $DB2$ in Figure 1 at distances $d = 0, 1, 2, 3, 4$

keyword pair	DB1	DB2
database:multimedia	1.5	0.4
multimedia:VLDB	0.5	0
database:VLDB	1.5	0.33

Figure 4: Compare the relationship scores of keyword pairs of $DB1$ and $DB2$ in Figure 1

to Definition 1, i.e., $rel(k_i, k_j) = \mathcal{R}[i, j]$. The higher the score, the stronger the relationship between them. For the two databases $DB1$ and $DB2$ in Figure 1, the relationship scores of the query keyword pairs are shown in Figure 4, where we set $\delta = 4$ and $K = 10$.

2.3 Implementation with SQL

The generation of the matrices $\mathcal{D}$, $\mathcal{T}_1$, $\mathcal{T}_2$, $\cdots$, $\mathcal{T}_\delta$ and $\mathcal{W}_0$, $\mathcal{W}_1$, $\cdots$, $\mathcal{W}_\delta$, for each DB, can be performed conveniently inside the local RDBMS using SQL.

2.3.1 Creation of $\mathcal{D}$

We use relation R_D(`kId, tId`) to represent the non-zero entries of the $\mathcal{D}$ matrix. Each record (`kId, tId`) corresponds to the occurrence of a keyword (`kId`) in a tuple (`tId`). A separate table R_K(`kId, keyword`) stores all the keywords and their associated ids in the database. These two relations can be populated by scanning all the native tables of the local database, parsing each tuple to extract the keywords, removing stop words, stemming each keyword, and inserting the keyword id (`kId`) and tuple id (`tId`) pair into R_D(`kId, tId`) and the pair of `kId` and keyword into R_K(`kId, keyword`). (Each native table is inserted with a field `tId` as the identity of the tuples in the database.)

The main cost includes a sequential read of all the tuples, and two sequential writes of the tables R_K and R_D.

2.3.2 Creation of $\mathcal{T}_1$, $\mathcal{T}_2$, $\cdots$, $\mathcal{T}_\delta$

Matrices $\mathcal{T}_1$, $\mathcal{T}_2$, $\cdots$, $\mathcal{T}_\delta$ are represented with relations R_{T_1}(`tId1, tId2`), R_{T_2}(`tId1, tId2`), $\cdots$, R_{T_δ}(`tId1, tId2`), separately. In each relation R_{T_d}(`tId1, tId2`)$(1 \le d \le \delta)$, the tuples represent the non-zero entries of the upper-right half of $\mathcal{T}_d$, since it is symmetric.

The tables R_{T_1}, R_{T_2}, $\cdots$, R_{T_δ} are generated incrementally. First, R_{T_1}(`tId1, tId2`) is generated by performing join operations on the pairs of native tables based on their foreign key reference relationships, and the joined results, pairs of tuple ids, are inserted into R_{T_1}(`tId1, tId2`). Next, R_{T_2}(`tId1, tId2`) is populated by self-joining R_{T_1}(`tId1, tId2`). When $d \ge 3$, R_{T_d} is generated by joining $R_{T_{d-1}}$ with R_{T_1}, and excluding the tuples already in $R_{T_{d-1}}$, $R_{T_{d-2}}$, $\cdots$, and R_{T_1}. Figure 5 shows the sample SQL statements for creating R_{T_3}.

Assuming each field is a 3-byte unsigned integer, the maximum space overhead of storing all the tables R_{T_1}, R_{T_2}, $\cdots$, R_{T_δ} is $\frac{6*n^2}{2} = 3n^2$ bytes, where n is the total number of tuples in DB, while the actual size is far less than the number, as the matrices are very sparse.

```
INSERT INTO R_T3(tId1, tId2)
SELECT s1.tId1, s2.tId2
FROM R_T2 s1, R_T1 s2
WHERE s1.tId2 = s2.tId1

INSERT INTO R_T3(tId1, tId2)
SELECT s1.tId1, s2.tId1
FROM R_T2 s1, R_T1 s2
WHERE s1.tId2 = s2.tId2 AND s1.tId1 < s2.tId1

INSERT INTO R_T3(tId1, tId2)
SELECT s2.tId1, s1.tId2
FROM R_T2 s1, R_T1 s2
WHERE s1.tId1 = s2.tId2

INSERT INTO R_T3(tId1, tId2)
SELECT s1.tId2, s2.tId2
FROM R_T2 s1, R_T1 s2
WHERE s1.tId1 = s2.tId1 AND s1.tId2 < s2.tId2

DELETE a FROM R_T3 a, R_T2 b, R_T1 c
WHERE (a.tId1 = b.tId1 AND a.tId2 = b.tId2) OR
      (a.tId1 = c.tId1 AND a.tId2 = c.tId2)
```

Figure 5: SQL for creating R_{T_3}

2.3.3 Creation of $\mathcal{W}_0$, $\mathcal{W}_1$, $\cdots$, $\mathcal{W}_\delta$

$\mathcal{W}_0$ is represented with a relation R_{W_0}(`kId1, kId2, freq`), where tuple (`kId1, kId2, freq`) records the pair of keywords (`kId1,kId2`)$(kId1 < kId2)$ and its frequency (`freq`)

```
INSERT INTO R_W0(kId1, kId2, freq)
SELECT s1.kId AS kId1, s2.kId AS kId2, count(*)
FROM R_D s1, R_D s2
WHERE s1.tId = s2.tId AND s1.kId < s2.kId
GROUP BY kId1, kId2
```

Figure 6: SQL for creating R_{W_0}

```
INSERT INTO R_Wd(kId1, kId2, freq)
SELECT s1.kId AS kId1, s2.kId AS kId2, count(*)
FROM R_D s1, R_D s2, R_Td r
WHERE ((s1.tId = r.tId1 AND s2.tId = r.tId2) OR
       (s1.tId = r.tId2 AND s2.tId = r.tId1)) AND s1.kId < s2.kId
GROUP BY kId1, kId2
```

Figure 7: SQL for creating R_{W_d}

at 0 distance, where `freq` is greater than 0. R_{W_0} is the result of self-joining R_D`(kId, tId)`. The SQL statement for populating R_{W_0} is shown in Figure 6.

Similarly, $\mathcal{W}_d$ $(1 \leq d \leq \delta)$ is represented as a relation R_{W_d}`(kId1, kId2, freq)`. Its records are populated by joining R_D and R_{T_d}. Figure 7 shows the SQL statement.

The dominating cost for creating R_{W_d} $(0 \leq d \leq \delta)$ is the cost used for grouping the tuples resulted from the WHERE clause according to `(kId1, kId2)`, as sorting is needed in order to group the same keyword pairs together. The total space overhead for storing R_{W_0}, R_{W_1}, $\cdots$, R_{W_δ} is at most $3m^2$, where m is the total number of distinct keywords, with `kId` using 3-byte unsigned integer. The actual storage will be much less, since only a small portion of keyword pairs will have relationships within δ distance.

The final resulting KRM, $\mathcal{R}$, is stored in a relation R_R`(kId1, kId2, score)`, consisting of pairs of keywords and their relationship score, where the score is greater than 0. It is the union of R_{W_0}, R_{W_1}, $\cdots$, and R_{W_δ} with the score of each distinct keyword pair calculated according to Definition 1 and Equation 2-2. The keyword pairs and their associated scores stored in R_R are the keyword relationship summary of the local database, which we named as *KR-summary*. A threshold τ can also be specified such that only the keyword pairs with relationship scores greater than τ will be exported.

2.3.4 Update issues

The tables for storing the matrices $\mathcal{D}$, $\mathcal{T}_1$, $\mathcal{T}_2$, $\cdots$, $\mathcal{T}_\delta$ and $\mathcal{W}_0, \mathcal{W}_1, \cdots, \mathcal{W}_\delta$, can be maintained dynamically when new tuples are added to the database or old tuples are deleted.

When a new tuple t is inserted into a table in DB, we can find out its relationships with other related tuples at various distances step by step. First, we identify the set of tuples that directly reference (or are referenced by) t, denoted as S_1, and insert all the pairs — t with each tuple in S_1 — into the table R_{T_1}. Then, we can further get the set of tuples, S_2, that are directly connected to any one of the tuples in S_1, and they have 2-distance relationships with t. So we insert the corresponding pairs into R_{T_2}. In addition, since the tuples in S_1 are all connected to t, they themselves are connected to each other at 2 distance. Therefore, we also insert the pairs of tuples from S_1 into R_{T_2}. We can repeat the process until R_{T_δ} is updated. Update of R_{W_0}, R_{W_1}, $\cdots$, and R_{W_δ} can be done together with the update of corresponding R_{T_d}, with the keywords pairs appearing in the affected tuples.

When a tuple t is deleted from a table in DB, we similarly first identify the set of tuples, S_1, that directly reference (or are referenced by) t. We remove the corresponding tuple pairs from R_{T_1} and update R_{W_1} with the corresponding keyword pairs appearing in the tuple pairs. Then we find the set of tuples, S_2, that has 2-distance with t in R_{T_2}, and delete the pairs containing t. Note that we also need to delete those tuple pairs consisting of tuples from S_1, because they are connected at 2-distance via t (We assume there is no cycles in the tuple graph.). We update R_{W_2} as well in a similar manner. When we update R_{T_3}, we delete all tuple pairs containing t in it, and also delete all tuple pairs composed of the tuples from S_1 and S_2, respectively. This update process is repeated until R_{T_δ} and R_{W_δ} is updated.

3. DATA SOURCES SELECTION USING KRM

In this section, we present our strategy to effectively select useful data sources based on our KR-summary.

3.1 Estimating multi-keywords relationships

In the previous section, we discussed how the relationship between keywords in a relational database can be represented and computed. In general, given a query of multiple keywords, we need to present the relationship among these multiple query keywords. In a relational database, multiple keywords are connected with steiner trees [15], where the leaf nodes are the tuples containing the keywords, which are results, often called *tuple trees*, of the database to the query keywords. However, it is a NP-complete problem to find minimum steiner trees in graphs [15], and most current keyword search algorithms are based on heuristics to find the (approximate) top-K results in a database for a given keyword query. Therefore, we estimate the relationship among multiple keywords with our derived keyword pair relationships.

PROPOSITION 4. *Given a set of keywords $Q = \{k_1, k_2, \cdots, k_q\}$, the number of edges of the tuple tree T_Q that contains all the keywords in Q, is no less than*

$$\max_{1 \leq i,j \leq q, i \neq j} \{\min\{d | d \geq 0 \& \omega_d(k_i, k_j) > 0\}\}.$$

According to Proposition 4, we can determine a lower bound of the size of the tuple trees given a keyword query. If a pair of query keywords cannot be found in the KR-summary (i.e., their relationships score is 0 or below τ), the number of edges of the tuple tree including all the query keywords must be greater than δ, and therefore its score should be set as 0, in order that the data source can be safely pruned from selection. Consider our previous example in Figure 1, given the query $Q = \{multimedia, database, VLDB\}$, we will not choose $DB2$ since the relationship score between $multimedia$ and $VLDB$ in it is 0 (shown in Figure 4).

On the other hand, when each pair of query keywords appears in the KR-summary, we estimate the score of the data source DB to Q through the scores of individual pairs. Specifically, we propose four kinds of estimations, denoted as $rel_{min}(Q, DB)$, $rel_{max}(Q, DB)$, $rel_{sum}(Q, DB)$ and $rel_{prod}(Q, DB)$, which are defined as follows, respectively,

$$rel_{min}(Q, DB) = \min_{\{k_i, k_j\} \subseteq Q, i<j} rel(k_i, k_j), \quad (3\text{-}3)$$

$$rel_{max}(Q, DB) = \max_{\{k_i, k_j\} \subseteq Q, i<j} rel(k_i, k_j), \quad (3\text{-}4)$$

$$rel_{sum}(Q, DB) = \sum_{\{k_i,k_j\}\subseteq Q, i<j} rel(k_i, k_j), \quad (3\text{-}5)$$

$$rel_{prod}(Q, DB) = \prod_{\{k_i,k_j\}\subseteq Q, i<j} rel(k_i, k_j). \quad (3\text{-}6)$$

These estimations assume different degrees of intersections of the joining sequences for connecting pairs of query keywords in the database, where intersections between joining sequences of keyword pairs lead to steiner trees that contain all the keywords. The $rel_{min}(Q)$ is most conservative as it assumes few intersections, while $rel_{prod}(Q)$ assumes the highest degree of intersection. Note that since these estimations are only used for ranking, their accuracy compared to the actual value is not so important as long as the ranking is correct and we will show that this is the case in the experiment section. Interested readers are referred to [8] which in the same spirit demonstrates why simple Bayesian classifier works well even on datasets where the assumption of attribute independency is invalid.

3.2 Databases ranking and indexing

With the KR-summary, we can effectively rank a set of databases $D = \{DB_1, DB_2, \cdots, DB_N\}$ for a given keyword query. Specifically, the ranking is a mapping from D to $\{1, 2, \cdots, N\}$, such that $rank(DB_i) < rank(DB_j) \Leftrightarrow rel\ (Q, DB_i) \geq rel\ (Q, DB_j)$, where $rel(Q, DB_i)$ denotes the relationship score of Q in DB_i. With a user provided number l, we can select the top l number of databases with highest ranks.

In order to support efficient ranking, we have different choices of indexing mechanism depending on the network structure. We generalize them into two types.

Global Index

For efficient selection of the top l number of databases, a global index can be built on the summaries of local source databases. The index contains a list of distinct keyword pairs that appear in the local database summaries. In a manner similar to the "inverted file" used in IR, for each keyword pair, there is an inverted list of pairs containing the source database identifier in which the pair of keywords appears and the relationship score between them in that database. A keyword query Q of multiple keywords is evaluated by fetching the inverted lists for each pair of different keywords in Q, and then intersecting the lists of database identifiers and calculating the relationships scores for each database. As the page accesses for answering a query through the index is typically $O(logn)$, a global index for keyword pairs is expected to have at most 2 times more page accesses comparing to a simple keyword index when answering keyword based query.

A global index is typically stored in a central server which querying clients will submit their queries to. As it is, this paradigm is well studied and our interest in this paradigm will be limited to the discussion here.

Decentralized Index

Another possible paradigm is a decentralized index where query clients are communicating with each other and are able to share the indexing workload. A typical example of this paradigm is a peer-to-peer (P2P) network. In this case, the global index can be distributed among the peers and each of them can store a certain portion of the keyword pairs and the associated inverted lists.

To do so, each local database is attached to the P2P network or to a server node in the network, and publishes its summary, i.e., the keyword pairs and the associated scores, which are disseminated to other nodes. When a query is received at some node, a set of search messages is sent out for each pair of keywords in the query. The corresponding inverted lists are returned from different nodes in order that the most useful databases can be selected. For experimental purpose, in the next section, we evaluate our implementation of this paradigm over Chord [28], a DHT-based P2P overlay system in order to see the usefulness and feasibility of KR-summary being applied in such a context.

4. EXPERIMENTS

In order to evaluate the performance of our proposed approach for the selection of relational databases, we have implemented the system over PlanetLab [7], which is a testbed for large-scale distributed systems. All codes are written in Java. We built the distributed index for KR-summaries over a Chord-based [28] P2P network as described in Section 3.2. We selected sixteen physical nodes on Planetlab in various areas, and each physical node is used to simulate several to hundreds of Chord nodes, where each Chord node shares one relational database.

We use real life DBLP[1] dataset to generate 82 relational databases by dividing the whole dataset according to different bibliography types such as inproceedings, articles, books, etc.. Figure 8 shows the schema of the databases storing inproceedings papers. The schemas of other bibliography types are similar. There is no overlap between different generated databases. The average number of tuples per database is 46735, and the average number of distinct keywords extracted from each database is 19817, after removing stop words and stemming. The numbers of tuples of different databases are similar. Keyword queries are composed of randomly selected keywords from the databases. We tested with a set of 112 queries in total, which consists of 30 2-keyword queries, 34 3-keyword queries, 21 4-keyword queries, and 27 5-keyword queries.

We use MySQL[2] to store all the databases and generate KR-summaries for them.

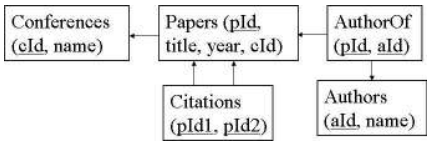


Figure 8: Schema of the databases storing inproceedings papers of DBLP dataset

4.1 Effectiveness of the KR-summary

To evaluate the effectiveness of KR-summary on relational databases selection, we compare our method with the brute force selection, that is, sending the given query to all the source databases, which process the query and return top-K results, in order that we can get the "real" score of each database based on Equation 2-1. Note that the execution time of such a brute force selection is orders of magnitude longer than that of the summary-based selection. With the real score, we define the *real ranking* of the databases as

[1] http://dblp.uni-trier.de/

[2] http://www.mysql.com/

$real\_rank(DB_i) < real\_rank(DB_j) \Leftrightarrow real\_score(Q, DB_i) \geq real\_score(Q, DB_j)$.

Our algorithm for processing keyword query on a relational database is implemented based on the approach of the DISCOVER project [13, 14] and [21].

To compare our estimated ranking with real ranking, we use the metrics defined in IR for evaluating text data source selection algorithms [12, 26], which is in the same style as the well known precision and recall definitions. The recall is defined as

$$recall(l) = \frac{\sum_{DB \in Top_l(S)} score(Q, DB)}{\sum_{DB \in Top_l(R)} score(Q, DB)},$$

where S and R denote summary-based rankings and real rankings of all the source databases respectively, while $Top_l(S)$ and $Top_l(R)$ represent the l databases with highest ranks in S and R. Note that $score(Q, DB)$ is the real score generated according to formula 2-1. This recall definition compares the accumulated score of the top l databases selected based on the summaries of the source databases against the total available score when we select top l databases according to the real ranking. The precision measure is defined as

$$precision(l) = \frac{|\{DB \in Top_l(S) | score(Q, DB) > 0\}|}{|Top_l(R)|}.$$

It measures the fraction of the top l selected databases with non-zero real scores, which have results with acceptable quality (controlled by K) to the query.

We also compare the effectiveness of our KR-summary against the keyword frequency summary, which is typically used as the summary of textual document collection for text data source selection in IR [12], denoted as *KF-summary*. The KF-summary of each relational database is a list of keywords that appear in the database associated with their frequencies, i.e., the number of tuples that contain the keyword. Based on the KF-summary, we estimate the score of a database DB for a given query $Q = \{k_1, k_2, \cdots, k_q\}$ in two ways. One is by summing the frequencies of all query keywords in DB, i.e.,

$$kf\_score_{sum}(Q, DB) = \sum_{i=1}^{q} freq(k_i). \quad (4\text{-}7)$$

The other is to take the product of the frequencies, i.e.,

$$kf\_score_{prod}(Q, DB) = \prod_{i=1}^{q} freq(k_i). \quad (4\text{-}8)$$

We study the effectiveness of our method along three dimensions. First, we examine the impact of δ on the ranking quality by comparing the precision and recall of the rankings generated with different values of δ. Second, we compare the ranking effectiveness for queries with different number of keywords. Third, we compare the performance of the four different estimations, MIN, MAX, SUM, and PROD, which correspond to the formulas 3-3, 3-4, 3-5, and 3-6, for measuring the relationship on multiple keywords.

4.1.1 Effects of δ

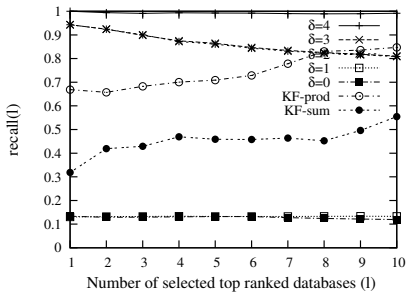
Figure 9 shows the average precisions and recalls of 2-keyword queries with KR-summary when δ is set to 0, 1, 2, 3, and 4, separately, and with KF-summary. We have the following three observations with regard to the effects of δ. First, the selection performance of KR-summaries generally gets better when δ grows larger. When $\delta = 4$, both precision and recall stay close to 1. Second, the precision and recall performance for different values of δ tends to cluster into groups. We can see that the precisions and recalls of KR-summaries when $\delta = 0$ and $\delta = 1$ are in a group and belong to another group when δ is set to 2 and 3. Third, there are big gaps in both precisions and recalls between KR-summaries when $0 \leq \delta \leq 1$ and when δ is greater.

These phenomena should all be related to the sizes of the KR-summaries with different δ values, i.e., the numbers of keyword pairs in the summaries. When δ is larger, there will be more keyword pairs and thus the KR-summary can capture more relationships between keywords, which results in better performance. However, it is not true that increasing δ will always result in an increase of the size of KR-summary. On the contrary, the size is largely dependent on the structure of the databases. With our data set, the sizes of KR-summaries with δ equals to 0 and 1 do not vary much, likewise for KR-summaries with δ set to 2 and 3. However, when δ is increased from 1 to 2, there is a big increase of the size of KR-summaries. To explain this, let's refer to the schema shown in Figure 8. The 2-distance joining sequences include the results of $Papers \bowtie AuthorOf \bowtie Authors$, $Papers \bowtie Citations \bowtie Papers$, $Papers \bowtie Conferences \bowtie Papers$, and $Authors \bowtie AuthorOf \bowtie Authors$, all which will lead to lots of keyword pairs as the tables $Papers$, $Conferences$, and $Authors$ are all text-rich. This explains the similarities of the performance between $\delta = 0$ and $\delta = 1$, between $\delta = 2$ and $\delta = 3$, and also the big jump in the performance between $\delta = 1$ and when δ is set higher.

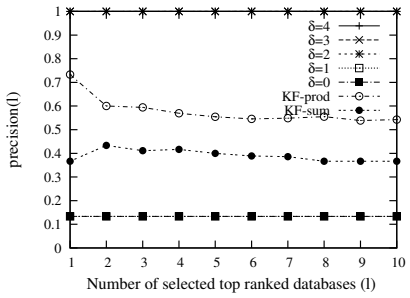
In addition, we can see from Figure 9 that KF-summary with production estimation (formula 4-8) outperforms that with summation estimation (formula 4-7). However, comparing the performance of KR-summary and KF-summary, we find that the former can do much better. For example, when $l = 3$, KR-summary with $\delta = 2$ and $\delta = 3$ outperforms KF-summary 67% in precision and 28% in recall, while KR-summary with $\delta = 4$ improves 43% over KF-summary in recall. As we have explained, the inferior performance of KF-summary is due to the existence of many "false positives" in its results, since it cannot identify unrelated keywords in the databases.

Another notable phenomena is that in Figure 9(a), the recall of KR-summary when $\delta = 2$ and $\delta = 3$ declines as l increases, while the recall of KF-summary increases, such that it outperforms KR-summary ($\delta = 2$ and $\delta = 3$) slightly when l is greater than 8. This shows that KR-summary ($\delta = 2$ and $\delta = 3$) tends to rank databases with higher scores below those with lower scores when l is larger. This should be attributed to insufficient keyword relationship information in the KR-summary for small δ, and hence it underestimates the scores of some databases. However, it can still identify databases with very good scores, which is revealed in its high recall when l is small. This is because very relevant databases tend to contain results of small sizes, which has already been captured in the KR-summary with smaller δ.

We next perform the same test with keyword queries consisting of more keywords. Figure 10 shows the results of queries with 3 to 5 keywords. Similar to Figure 9, the curves of $\delta = 0$ and $\delta = 1$ still group together, likewise for those of $\delta = 2$ and $\delta = 3$. In addition, it is interesting to note that the precision of KR-summary when $\delta = 2$ and $\delta = 3$ is better than that when $\delta = 4$, which means that more

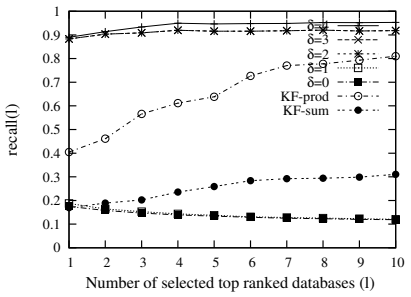


(a) recall

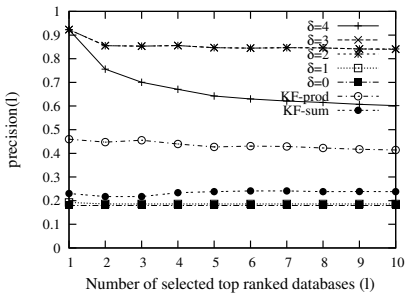


(b) precision

Figure 9: Recall and precision of 2-keyword queries using KR-summaries at different values of δ and KF-summaries (KF-prod and KF-sum denote production and summation estimation)

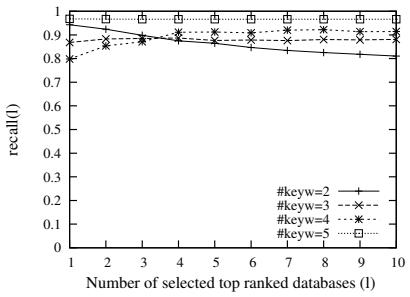


(a) recall

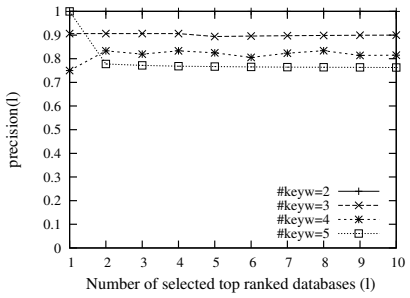


(b) precision

Figure 10: Recall and precision of queries with 3-5 number of keywords with KR-summaries at different values of δ (with SUM estimation) and with KF-summary



(a) recall



(b) precision

Figure 11: Recall and precision of queries with different number of keywords for $\delta = 3$ and with SUM estimation

"false positives" are selected with $\delta = 4$ than with $\delta = 2$ and $\delta = 3$. This indicates that having more information in the KR-summary when $\delta = 4$ can lead to over estimating the usefulness of some databases, when more keywords are found in the query. This is because the relationship scores are based on estimation from keyword pair scores. We also note that KR-summary still greatly outperforms KF-summary when number of keywords are more than 2. In fact, we found that the superiority of KR-summary performance over that of KF-summary is more obvious when the number of query keywords is increased.

The experiments of this section provide us guidelines in choosing a good value of δ to achieve satisfied performance. Ideally, δ should be chosen based on the schema of the database, such that the connections of tables with rich textual information can be revealed in the summary. For example, $\delta = 3$ is a proper value for the schema shown in Figure 8. When $\delta = 4$, more distant relationships between keywords from tables are included, where the pairs of tables the keyword pairs appear in have already been summarized at lower distance. Such information tends to overestimate the usefulness of the databases when it is used to estimate multiple-keyword relationships in them. The problem of constructing an effective model for deriving an "optimal" δ value based on specific database schemas is complex and large by itself. We will address it in our future work.

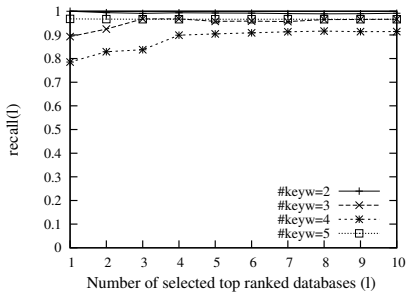
4.1.2 Effects of the number of query keywords

Next, we compare the performance of KR-summary for queries with different number of keywords. Figure 11 and 12 show the results when δ is set to 3 and 4, respectively. From the figures, we found that the performance of 2-keyword queries is generally better than that of 3-keyword and 4-keyword queries. Between 3-keyword and 4-keyword queries, the former is generally better. It is surprising however to see that 5-keyword queries yield better recall than 3-keyword and 4-keyword queries, and also better than 2-keyword queries when $\delta = 3$. It is natural to expect that the performance should degrade when the number of query keywords increases because it becomes harder to estimate the relationships among the keywords.

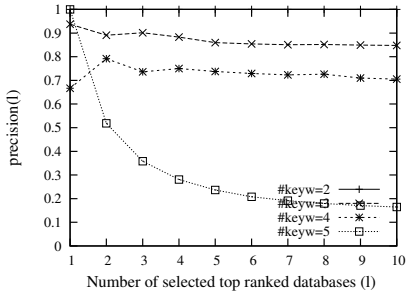
After investigation, we realize that 5-keyword queries are generally more selective, i.e., only a few source databases have non-zero scores for the queries and this results in higher recall. This is also the reason that the precision of 5-keyword queries decreases greatly when l increases, especially for $\delta = 4$ since more "false positives" are included. Generally, the difference in the recall of queries with different number of keywords is less than that of precision. This shows that the estimation method is effective in assigning high ranks to useful databases, although less relevant or irrelevant databases might also be selected.

4.1.3 Comparing four kinds of estimations

In this experiment, we compare the 4 proposed estimations of database scores using KR-summary — MIN, MAX, SUM, and PROD. Figure 13 and Figure 14 present the results with queries consisting of 3-5 keywords and when δ is set to 3 and 4, respectively. We can see that SUM and PROD methods have very similar behavior, and they consistently outperform the other two methods. This shows that it is more effective to take into account the relationship information of every pair of keywords in the query when estimating the overall score of the databases to the query. We also note that the different estimation methods affect recall more than precision, which means that SUM and PROD

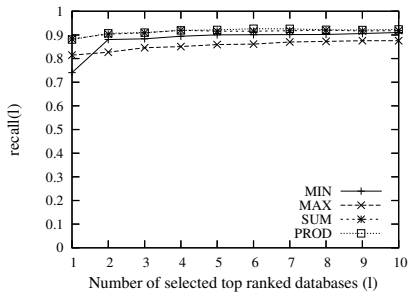


(a) recall

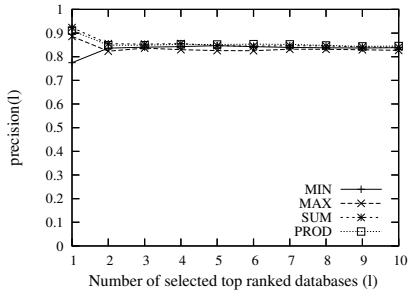


(b) precision

Figure 12: Recall and precision of queries with different number of keywords for $\delta = 4$ and with SUM estimation

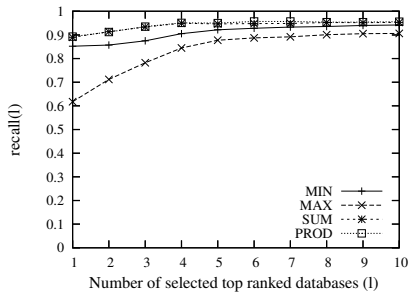


(a) recall

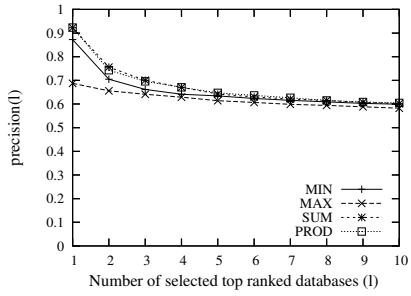


(b) precision

Figure 13: Recall and precision of queries with KR-summaries with different estimation methods ($\delta = 3$)



(a) recall



(b) precision

Figure 14: Recall and precision of queries with KR-summaries with different estimation methods ($\delta = 4$)

methods are more capable of placing databases with higher scores to the front of the results list.

4.2 Performance in P2P settings

In this section, we present the cost of indexing and query processing with our KR-summaries over the PlanetLab testbed.

4.2.1 Cost of indexing KR-summary for each database

We first look at the cost for distributing the the KR-summary of each database over the Chord [28] structure varying the number of nodes from 1000 to 10000. Figure 15 shows the number of messages transmitted over the network. From the figure, we see that when the number of nodes in the network gets larger, the number of messages increases slightly, due to the fact that the keyword pairs from a single database will be distributed over more Chord nodes. Also, not surprisingly, we can see that when δ is larger, more messages are used since the size of KR-summary increases.

Correspondingly, Figure 16 shows the actual elapsed time for indexing a database when the number of nodes varies. The increase in number of messages causes longer indexing time when the number of nodes increases, or when δ is larger. Generally, the increase in time is linear to the number of nodes, and therefore the indexing method is feasible.

4.2.2 Cost of query processing

We evaluate the cost for processing each query with the Chord-based index in this section. Figure 17 shows the average number of messages required for processing a query. We used four pairs of queries each consisting of 2,3,4 and 5 keywords respectively making eight queries in total. We can see that the number of messages increases very gradually with the increase of nodes, since more nodes must be accessed to answer the queries. Figure 18 reports the average elapsed time for processing each query. We note that the waiting time increases slowly when the number of nodes increases. Also, when δ becomes larger, the processing time increases. This is because more results are returned when δ is larger, which incur more transmission time.

4.3 Time and space cost for generating KR-summaries

We take one of the experimental databases with 52106 tuples and 20956 distinct keywords as an example to present the time and space cost for generating KR-summaries. Other databases have similar statistics, since their sizes are similar.

The KR-summaries are generated with MySQL running on a server with 3.2GHz Intel Pentium processor and 2G RAM. The average time for generating KR-summaries is 9.7, and 20.0 seconds when δ is set to 0 and 1. It jumps to 103 and 146 minutes when δ is set to 2 and 3, and it takes about 9 hours when $\delta = 4$. The main reason of the dramatic increase of time as the increases of δ is that when δ is larger, the number of related keyword pairs increases greatly, which causes much more time being spent in generating table R_{W_δ}.

Next, we compare the storage size of the tuple relationship matrices at different distances (Table 1) and the final KR-summaries with δ set to different values (Table 2).

d	#entries	size on disk (compressed)	index size (compressed)	% of n^2
1	61765	0.17MB	0.64MB	0.002
2	4500687	3.77MB	22.89MB	0.16
3	21788108	13.77MB	124.50MB	0.80
4	41898415	36.04MB	322.22MB	1.54

Table 1: Size of the tables for storing tuple relationship matrices at different distances (d)

From Table 1, we can see that the size of tuple relationship matrices becomes larger when the distance increases, but

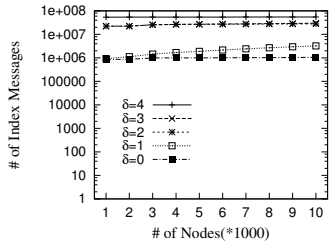


Figure 15: Number of messages for indexing one database

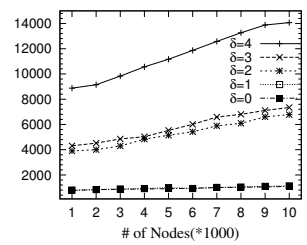


Figure 16: Elapsed time for indexing one database

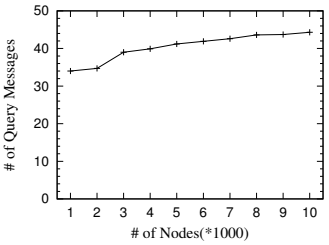


Figure 17: Average number messages for processing each query

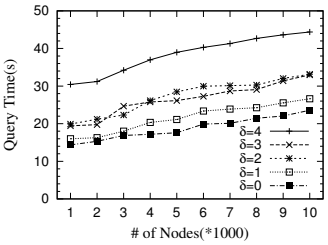


Figure 18: Average elapsed time for processing each query

δ	#entries	size on disk (compressed)	% of m^2
0	207037	0.64MB	0.047
1	212176	0.65MB	0.048
2	6270482	15.03MB	1.42
3	6275887	15.06MB	1.42
4	25622178	57.62MB	5.83

Table 2: Size of KR-summaries with different δ

the matrices are still very sparse. As intermediate results, the tuple relationship matrices can be deleted after the KR-summary is generated to save storage space. However, they can also be kept for efficient updating.

As mentioned earlier, the increase in the size of KR-summary is less smooth as δ increases, which can be seen from Table 2. The sizes of KR-summaries are similar when $\delta = 0$ and $\delta = 1$ and likewise for $\delta = 2$ and $\delta = 3$. We also see that the number of keyword pairs included in KR-summary is a small portion of all possible combinations of keyword pairs. Note that the size of the KR-summary is mostly related to the number of keywords in the source database, not the original size of the database.

5. RELATED WORK

We discuss the related work to our approach in this section.

5.1 Keyword search in relational databases

Keyword search over relational databases has been studied extensively recently [1, 2, 3, 13, 14, 19, 21]. All these works focus on efficiently and effectively generating (approximate) top-K results for a given keyword query in a single relational database. They differ from each other in their specific search algorithms, and the ranking functions for returning most relevant top-K results. The DISCOVER approach generates tuple trees by enumerating and evaluating Candidate Networks (CN), which represent join expressions that can generate potential answers, based on the schema graph of the database. The BANKS system represents all the tuples in a database as a graph where the nodes are tuples and links between nodes denoting references between corresponding tuples. The answers are generated by searching steiner trees in the tuple graph that contains all the query keywords. Our proposed summary-based database selection technique complements these search engines on a single database, in order that keyword search to large number of distributed databases can be effectively supported.

5.2 Data sources selection in data integration

The problem of data source selection is also studied in the data integration literature [20, 24], but with very different settings. A typical data integration system consists of a set of heterogeneous data sources containing data, a global virtual schema exposed to users, and a set of mappings between each of the source schema and the global schema. The set of schema mappings is essentially the descriptions of the sources, with which the system will generate query execution plan that can access multiple useful data sources, given a query issued against the global schema. [20] presents a way to declaratively describe the content and capabilities of data sources in terms of the global schema, which is later formulated as Local-as-View (LAV) approach. [24] studies the coverage problem of information sources, and propose a completeness model to measure the usefulness of data sources. While such data integration system aims to support integrated access of heterogeneous data sources via a structured global interface, our approach is for providing free-and-easy keyword search capability to various data sources. The advantage of our approach is that all the operations are fully automatic. In contrast, the mappings needed in data integration system can only be built manually or semi-automatically, which limits its applicability to large-scale and dynamic data sources.

5.3 Selection of unstructured text data sources

There have been many summary-based solutions developed in IR literature for selection of unstructured text data sources [5, 12, 29, 30]. Most summary techniques are based on keyword frequency statistics for estimating the usefulness of each data source in answering a given keyword query, e.g., GLOSS [12] and CVV [30]. CORI [5] summary also relies on keyword frequency statistics, together with the ICF (Inverse Collection Frequency) value, which is the inverse of the number of data sources that contain a particular keyword, to rank databases in a way similar to the standard TF.IDF measure for ranking documents in a single source. The ICF statistics could help identify the importance of a keyword across difference collections, but it cannot capture the structure information that is necessary for measuring the ability of structured data sources in answering a keyword query. These selection algorithms are examined and compared extensively in [9, 26] with various datasets. In addition, [29] proposes a summary technique for document collections where linkages among documents are available (e.g., web pages), and therefore documents with more references should be ranked higher. It thus incorporates the rank of documents in the summary to achieve better performance.

The construction of the summary for a text data source can be performed easily by scanning once all the documents to extract all the keywords and their associated frequencies, if the data source can be fully accessed. In some occasions, the data source can only be accessed via a limited search interface, query probing and sampling based methods are

needed to construct its summary [4, 17]. In [16], detailed cost models are given for these two types of approaches. In contrast, our approach for constructing KR-summaries for relational databases can either be performed in the local DBMS if full access is allowed, otherwise if an SQL query interface is provided, it can be done by crawling all the tables and creating the KR-summary in a foreign site.

5.4 Capturing keyword dependence for information retrieval

There have been research works [6, 11, 23] on capturing the dependence relationships between keywords for improving the effectiveness document retrieval tasks, considering that words occurring in a sentence are not independent in our natural language. For example, there is some degree of dependence between the occurrences of the keywords "computer" and "programming". Such relationships among keywords can be discovered by directly collecting information on co-occurrences of keywords from corpus, or by manually building thesauri to recognize synonyms, compound terms, etc., such as WordNet [10]. While such relationship captures the dependences of keywords in natural language, the keyword relationship defined in our work describes associations of keywords specific to a particular relational databases based on references between tuples, which may or may not be related to their dependence in natural language.

6. CONCLUSION AND FUTURE WORK

In this paper, we introduce a novel summary technique for relational databases for enabling keyword-based selection of distributed data sources. Different from traditional summary used in IR for text databases that mainly relies on keyword frequency information, our summary exploits the structure of the relational database and contains pairs of keywords with scores indicating the strength of their relationship, which are measured based on the references between tuples. We also propose an estimation method for ranking the usefulness of databases in answering a keyword query with our KR-summary. Our experimental results with real dataset demonstrate the effectiveness of our proposed summary approach. Further, our evaluation of the distributed indexing mechanism for the KR-summaries implemented over PlanetLab [7] confirms its feasibility, scalability and efficiency over a real distributed environment. Indeed, unlike traditional data integration techniques, our free-and-easy keyword based selection method requires no human intervention, hereby enabling scalability over a large network of distributed relational data sources.

Our summary technique can be extended in two directions. First, we can further incorporate IR-based weighting methods and weightings based on link structure into the KR-summary. For example, the $\mathcal{D}$ matrix could include the weighting of each keyword in the tuples it appears in, and the $\mathcal{T}$ matrix could use real numbers to indicate the importance of different links, instead of binary values. Second, we can exploit some sampling-based methods for constructing the KR-summary more efficiently and making it more compact.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[4] J. Callan and M. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.

[5] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR*, 1995.

[6] G. Cao, J.-Y. Nie, and J. Bai. Integrating word relationships into language models. In *SIGIR*, 2005.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[8] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proc. 1996 Int. Conf. Machine Learning (ML'96)*, pages 105–112, 1996.

[9] D. D'Souza, J. Thom, and J. Zobel. Collection selection for managed distributed document databases. *Information Processing & Management*, 40:527–546, 2004.

[10] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[11] J. Gao, J.-Y. Nie, G. Wu, and G. Cao. Dependence language model for information retrieval. In *SIGIR*, 2004.

[12] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, 1999.

[13] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.

[14] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.

[15] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. North-Holland, 1992.

[16] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To search or to crawl?: towards a query optimizer for text-centric tasks. In *SIGMOD*, 2006.

[17] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.

[18] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *VLDB*, 2005.

[19] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, 2006.

[20] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, 1996.

[21] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[22] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computer Survey*, 34(1):48–89, 2002.

[23] R. Nallapati and J. Allan. Capturing term dependencies using a language model based on sentence trees. In *CIKM*, 2002.

[24] F. Naumann, J.-C. Freytag, and U. Leser. Completeness of integrated information sources. *Information Systems*, 29(7):583–615, 2004.

[25] W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A self-configurable peer-to-peer system. In *ICDE*, 2002. Poster Paper.

[26] A. L. Powell and J. C. French. Comparing the performance of collection selection algorithms. *ACM Transactions on Information Systems*, 21(4):412–456, 2003.

[27] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[29] C. Yu, W. Meng, W. Wu, and K.-L. Liu. Efficient and effective metasearch for text databases incorporating linkages among documents. In *SIGMOD*, 2001.

[30] B. Yuwono and D. L. Lee. Server ranking for distributed text retrieval systems on the Internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, 1997.