Volume 4, Issue 6, June 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Estimation of Ranking Score for Database Query Results

Mr. Dr. S. PREM KUMAR, M-Tech, M-Phill, Ph. D, MISTE, Professor & Hod, Department of computer science and Information Technology , G. Pullaiah College Of Engineering &Technology, Kurnool, India
Mrs. K. GAYATHRI, Department of Computer Science and Engineering, G. Pullaiah College Of Engineering &Technology, Kurnool, India
Mr. I.S. RAGHURAM, M.Tech, MISTE, Assistant Professor , Department of Computer Science and Engineering G. Pullaiah College Of Engineering &Technology, Kurnool, India

Abstract:-Ranking and returning the most relevant results of a query is a popular paradigm in Information Retrieval. We discuss challenges and investigate several approaches to enable ranking in databases, including adaptations of known techniques from information retrieval. We present results of preliminary experiments. We investigate the problem of ranking the answers to a database query when many tuples are returned. In particular, we present methodologies to tackle the problem for conjunctive and range queries, by adapting and applying principles of probabilistic models from information retrieval for structured data. This solution is domain independent and leverages data and workload statistics and correlations. Evaluate the quality of our approach with a user survey on a real database. Furthermore, present experimentally evaluate algorithms to efficiently retrieve the top ranked results, which demonstrate the feasibility of this ranking system.

Keywords: ranking, Query, Performance

I. INTRODUCTION

Automated ranking of the results of a query is a popular aspect of the query model in Information Retrieval (IR) that we have grown to depend on. In contrast, database systems support only a Boolean query pattern. For an illustration, a choice query on a SQL database returns all tuples that satisfy the conditions in the query. Therefore, the following two scenarios are not gracefully handled by a SQL system:

1. ***NULL answers***: When the query is too selective, the answer may be empty. In that case, it is desirable to have the option of requesting a ranked list ofapproximately matching tuples without having tospecify the ranking function that captures“proximity” to the query. An FBI agent or an analystinvolved in data exploration will find such functionality appealing.

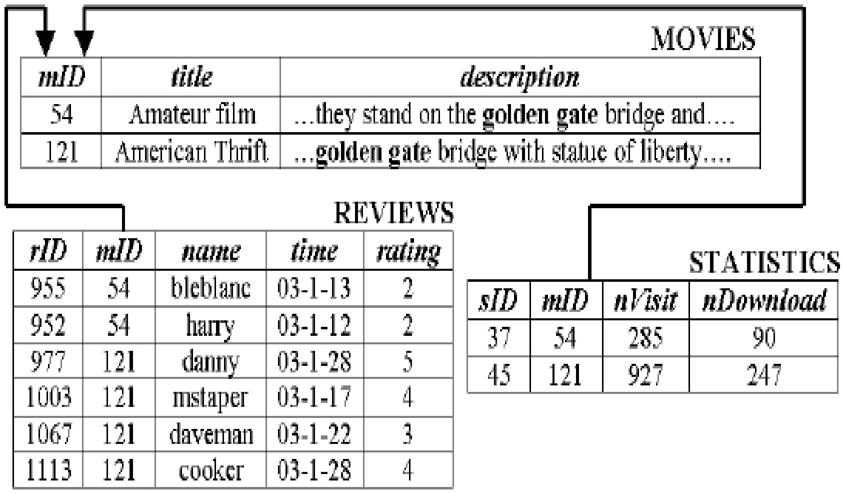
2. ***Many answers***: When the query is not too selective, too many tuples may be in the answer. In that case, it will be advantageous to have the option of ordering the matches automatically that ranks more “globally important” answer tuples higher and returning only the best matches. A customer browsing a product catalog will find such functionality attractive.

A major concern of this paper is the query processing techniques for supporting ranking. Several techniqueshave been previously developed in database research for the Top-*K* problem.

We start by briefly reviewing this standard IRtechnique. Given a set of documents and a query (thelatter specified as a set of keywords), the problem is to retrieve the Top-*K* documents most relevant, or most *similar* to the query. Similarity between a document and the query is formalized as follows. Given a vocabulary of *m*words, a document is treated as an *m*-dimensional vector, where the *i*th component is the frequency of occurrence (also known as *term frequency*, or TF) of the *i*th glossary word in the document. Since a query is a set of words, it too has a vector representation. The*Cosine Similarity* between a query and a document isdefined as the normalized dot-product of the twocorresponding vectors. The Cosine Similarity may befurther refined by scaling each component with the*inverse document frequency* (IDF) of the corresponding word (IDF(w) of a word w is defined as log(N/F(w)) where N is the number of documents, and F(w) is the number of document in which w appears). IDF has been used in IR to suggest that commonly occurring words convey less information about user’s needs than rarely occurring words, and thus should be weighted less. We can also adopt these techniques for our problem. More formally, for every value t in the domain of attribute Ak, we define IDFk(t) as log(n/Fk(t)), where n is the number of tuples in the database and Fk(t) is the frequencyof tuples in the database where Ak= t. For any pair ofvalues u and v in Ak’s domain, let the quantity $Sk(u,v)$ be defined as IDFk(u) if u = v, and 0 otherwise. Consider tuple T = <$t1$,…,tm> and query Q = <$q1$,…,qm> (i.e. the latter has a C-condition of the form “WHERE $A1$ = $q1$ AND … AND Am = qm”). The similarity between *T* and *Q* is defined in Equation (1). We refer to the quantities *Sk*(*u*,*v*) as similarity coefficients; thus the similarity between *T* and *Q* is simply the sum of corresponding similarity coefficients over all attributes. (To improve readability in the rest of the paper, we shall omit thesubscript *k* where ever possible. Thus *S*(*t*,*q*) will refer to the similarity coefficient *Sk*(*t*, *q*), while *A* will refer to the attribute *Ak*).

$$SIM\ (T, Q) = \sum_{k=1}^{m} S_k (t_k, q_k)$$

This similarity function closely resembles the IR-likeCosine Similarity with TF-IDF weightings, except that the dot-product is un-normalized. Also note that in our case, the term frequency TF is irrelevant since each tuple is treated as a small document in which a word, i.e. a <attribute, value> pair can only occur once. Henceforth we refer to this similarity function as *IDF Similarity*. IDF Similarity can be very effective in certain database ranking applications. For example, if we query an automobile database for a “CONVERTIBLE” made by “NISSAN”, the system first returns all Nissan convertibles, pursues by other convertibles, and pursue by other Nissan cars. This is because “CONVERTIBLE” is a rare car type and consequently has higher IDF than “NISSAN”, a common car manufacturer.



MOVIES

mID	*title*	*description*
54	Amateur film	…they stand on the **golden gate** bridge and….
121	American Thrift	…**golden gate** bridge with statue of liberty….

REVIEWS

rID	*mID*	*name*	*time*	*rating*
955	54	bleblanc	03-1-13	2
952	54	harry	03-1-12	2
977	121	danny	03-1-28	5
1003	121	mstaper	03-1-17	4
1067	121	daveman	03-1-22	3
1113	121	cooker	03-1-28	4

STATISTICS

sID	*mID*	*nVisit*	*nDownload*
37	54	285	90
45	121	927	247

Figure 1: Example Database

The proposed indices and algorithms can trade off query time for update time, can support both conjunctive and disjunctive keyword search queries, can support a combination of Structured Value Ranking(SVR) and term scores (such as TF–IDF), and can be tightly integrated with a relational database by reusing relational features such as B+–trees. Further, the indices can also efficiently support incremental *document* insertions, deletions and updates. Our implementation using Berkeley and our evaluation using both synthetic and real datasets specify that the preferred indices can effectively support Structured Value Ranking in update–intensive relational databases.

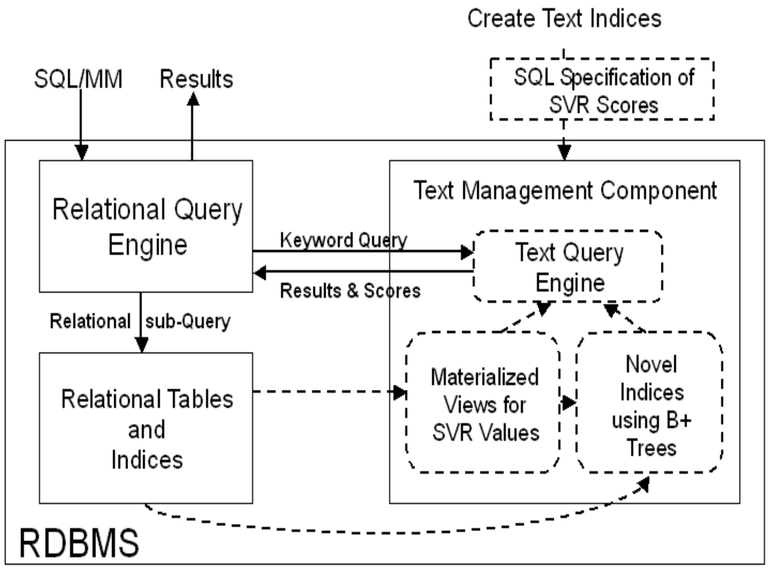

Create Text Indices
SQL/MM
Results
SQL Specification of SVR Scores
Relational Query Engine
Text Management Component
Keyword Query
Text Query Engine
Results & Scores
Relational sub-Query
Relational Tables and Indices
Materialized Views for SVR Values
Novel Indices using B+ Trees
RDBMS

Figure 2: System Architecture

II. SYSTEM ARCHITECTURE

One of our primary design goals was to tightly integrate SVR with a relational database. Towards this end, we build upon the architecture used by many commercial relational database systems for managing text data; this architecture is shown in Figure (ignore the dashed lines and boxes for now). The text fields are indexed by a text managementcomponent (called “extender” in DB2, “cartridge”in Informix, and “data blade” in Oracle). Users can point SQL/MM queries include *both* keyword search and other structured SQL sub–queries to the relational database engine. Given this joint query, the query engine first optimizes the query (rank–aware optimization can be used). Then, during query evaluation, the relational query processor invokes the text management component with the query keywords to obtain the top–ranked (or all)text documents along with their scores, and merges the results with the other structured sub–query results.

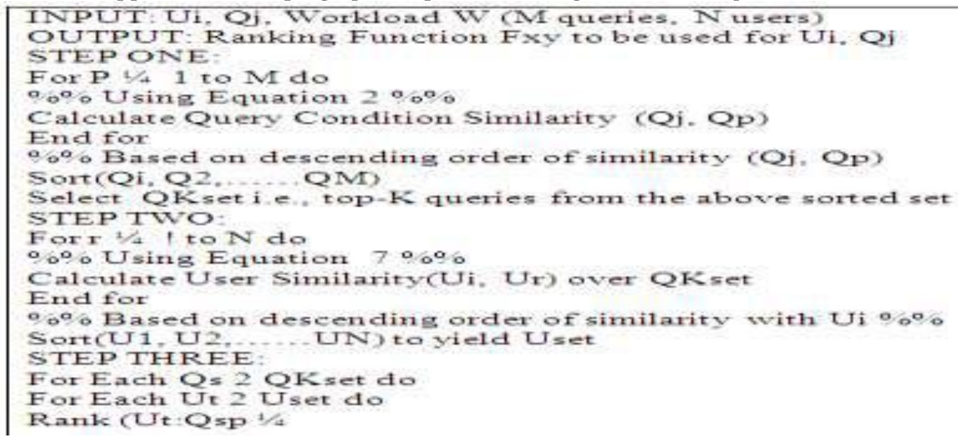
In this paper, we focus on the text management component of the above architecture and show how it can be extended to efficiently support SVR. There are two main extensions that need to be made. First, need to specify how the text keyword search results are to be ranked based on structured data values. Towards this end, we present a SQL–based framework for specifying SVR scores when creating a text index on a relational table (see dashed box in top right of Figure 2); system administrators or automatic ranking tools can use this framework to specify SVR scores for a given application. Second, need efficient index structures and associated query processing algorithms that produce the top–ranked results for keyword search queries based on the *latest* structured data values. Towards this end, we devise new index structures that can be rapidly updated while still providing good performance for top keyword search queries. These indices can be implemented using traditional B+–trees (see dashed box in Figure 2) and can thus be easily integrated with a relational database. We now describe our SQL–based framework for specifying and maintaining SVR scores using relational materializedviews. In the next section (which constitutes the bulkof the paper), describe our index structures and query processing algorithms.

III. RANKING ALGORITHM

In this paper the ranking model is based on two notions such as user similarity and query similarity.User similarity indicates that different users can have samepreferences. Query similarity indicates that different users can have identical queries. In order to accomplish this ranking of users and queries are to be maintained. We have developed a workload file that contains the user and query ranking functions. When new record is entered into database, obviously that is disposed by a user. There possibly many users who delivered that query previously and there might be same queries delivered earlier. The workload file is in tabular form and it gets updated with ranking functions as per the proposed algorithm as and when new queries are made. The proposed model has two patterns mixed. They are known as user addictive ranking model and query addictive ranking model.

However, we prefer applying both of them for more excellent results. The recommended ranking model in this paper is a linear weighted sum function. It includes attribute heaviness and value heaviness. Attribute weights indicate the importance of attributes while the value weight indicates the importance of values of attributes. Relevance feedback techniques are utilized for making the workload minimal. The main contributions of this paper are

- User and query dependent accession for ranking web databases.
- Ranking model based on user correspondence and query correspondence notions.
- Two synthetic databases such as college and hospital used for experiments. However, the model can be tested with web databases.
- We used a new workload approach for keeping up the updated ranking of users and queries.



```
INPUT: Ui, Qj, Workload W (M queries, N users)
OUTPUT: Ranking Function Fxy to be used for Ui, Qj
STEP ONE:
For P ¼ 1 to M do
%% Using Equation 2 %%
Calculate Query Condition Similarity (Qj, Qp)
End for
%% Based on descending order of similarity (Qj, Qp)
Sort(Qi, Q2,......QM)
Select QKset i.e., top-K queries from the above sorted set
STEP TWO:
For r ¼ ! to N do
%% Using Equation 7 %%
Calculate User Similarity(Ui, Ur) over QKset
End for
%% Based on descending order of similarity with Ui %%
Sort(U1, U2,......UN) to yield Uset
STEP THREE:
For Each Qs 2 QKset do
For Each Ut 2 Uset do
Rank (Ut:Qsp ¼
```

Figure 3: Ranking Algorithm

This algorithm is implemented in the prototype application which shows both user and query dependent rankings for query results of web databases.

IV. SAMPLE WORKLOAD FILE

The sample workload file is given in fig. which shows queries, users and the ranking functions calculated as per the algorithm given in listing 1.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
U1	??	F12	--	--	F15	--	F17	
U2	F21	F22	--	F24	--	F26	F27	--
U3	F31	F32	F33	F34	--	--	F37	--

Figure 4: Sample database

V. EXPERMENTAL EVALUATION

The experiments are made using a prototype application with two synthetic web databases such as college and hospital. The experimental results are evaluated by visualizing the results in the form of graphs. Figures shows the ranking quality of query similarity models for both databases with 10% work load.

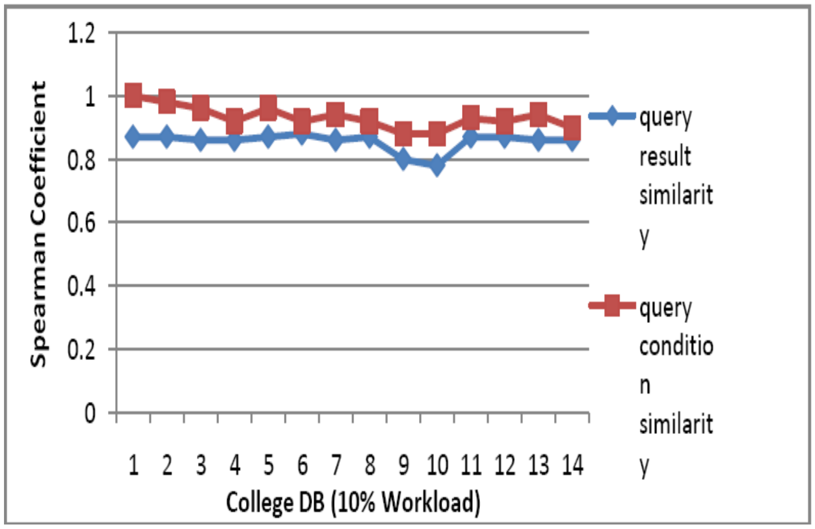


Figure 5: Ranking quality of query similarity (College DB)

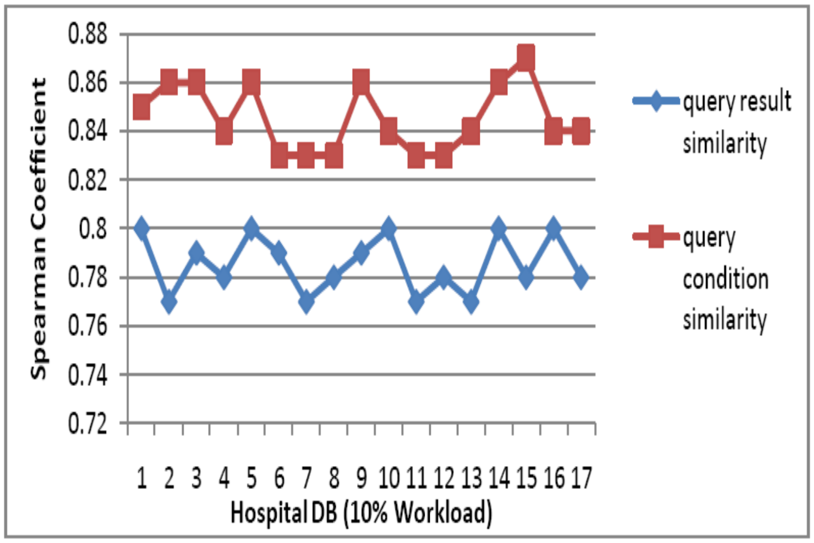


Figure 6 – Ranking quality of query similarity (Hospital DB)

As can be seen in fig. 5 and 6, query condition similarity average is found across all queries. The Xaxis shows queries while the Y axis shows spearman coefficient. As it is obvious in the graphs, the query condition model out performs query result model. The lost of quality is due the limited workload that is 10%.When workload increases, the quality also increases.

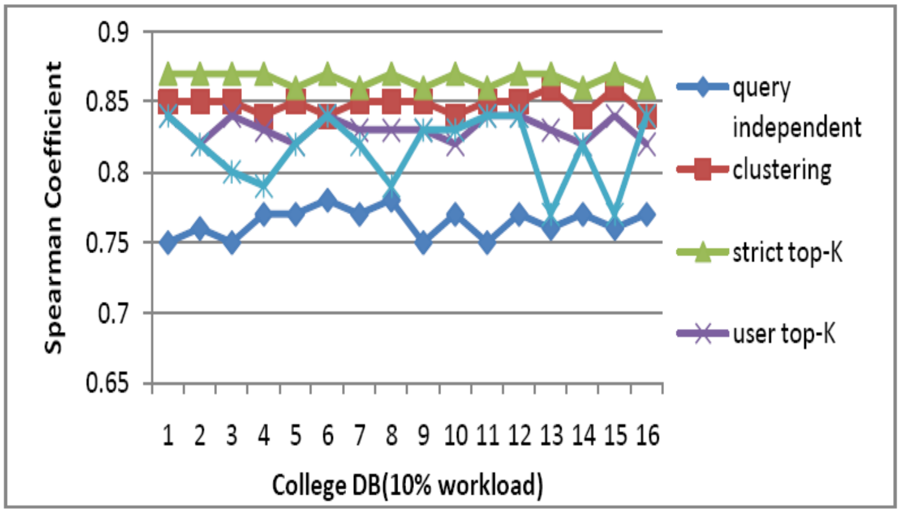


Figure 7 – Ranking quality of user similarity model (College DB)

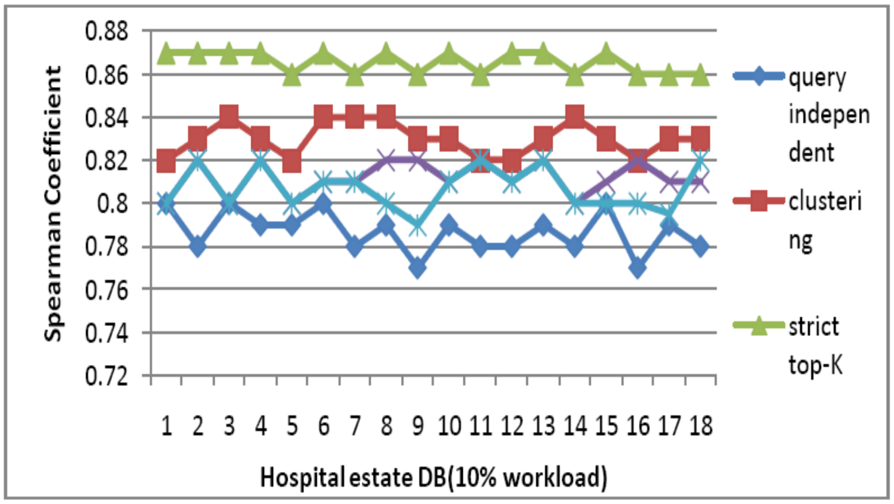


Figure 8 – Ranking quality of user similarity model (Hospital D)

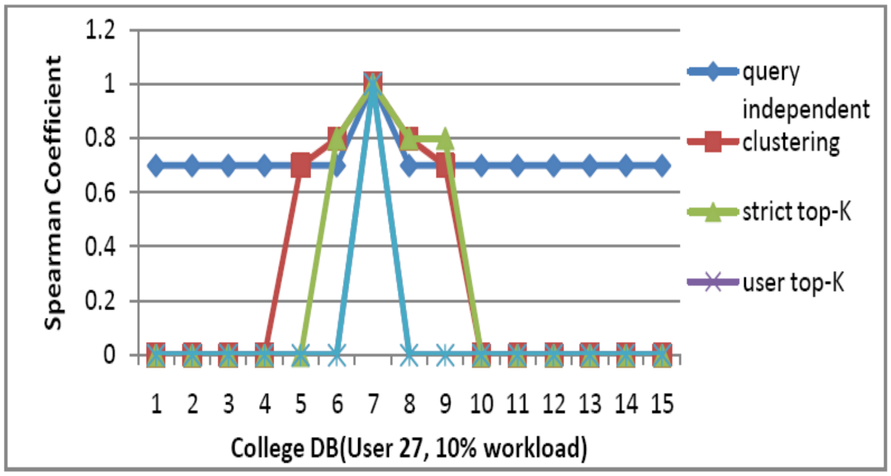


Figure 9 – Ranking quality of user similarity model (College DB)

Fig. 7, 8, and 9 show the average ranking quality achieved from both college and hospital database acrossall queries for all users. The results disclose that authoritarian top-K model performs bittern than other models. However,the strict top-K has no ranking functions for many queries.

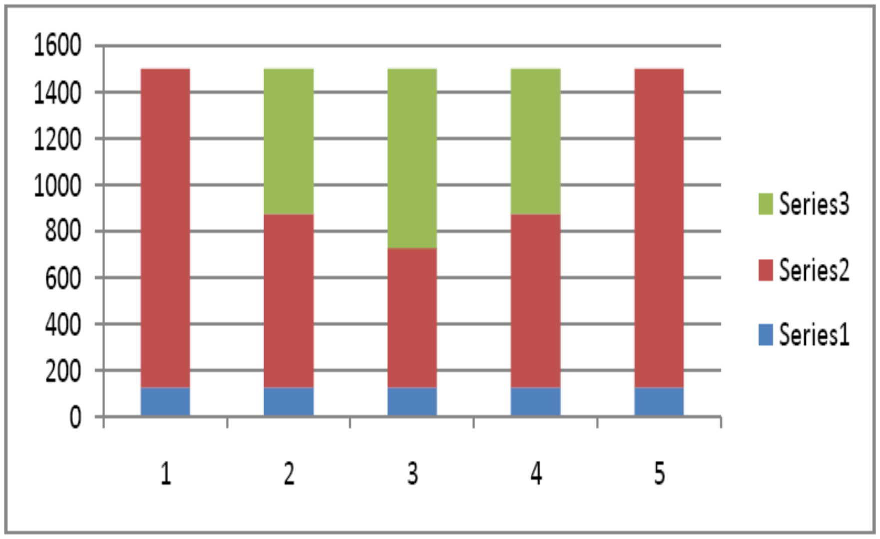


Figure10 – Ranking functions derived for user similarity (College DB)

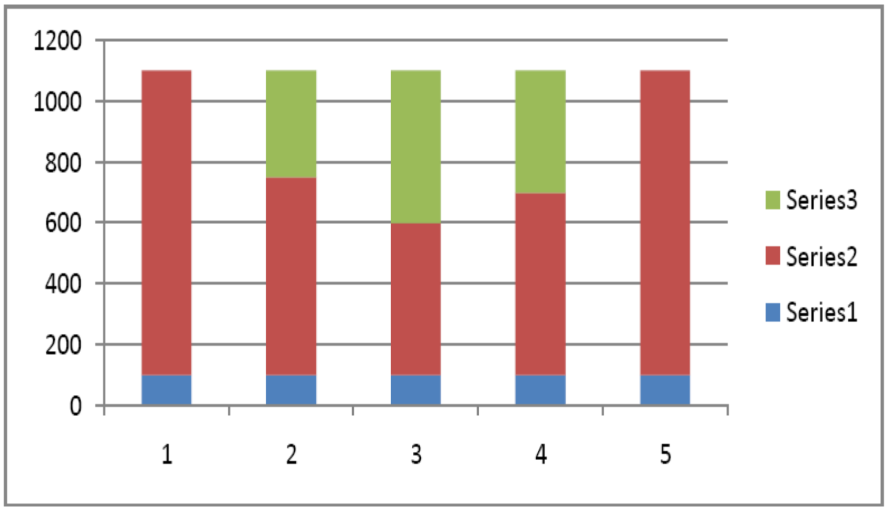


Figure 11 – Ranking functions derived for user similarity (Hospital DB)

Fig. 10 and 11 confirm the fact that different models have different abilities for determining rankingfunctions across the workload. However, the strict top-K is precise and superior to all other models from the perspective of ranking function.

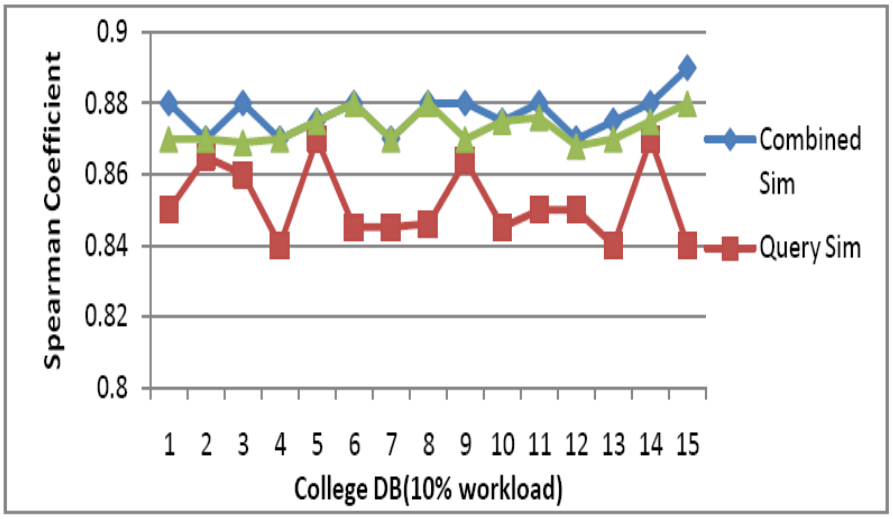


Figure 12 – Ranking quality of combined similarity model

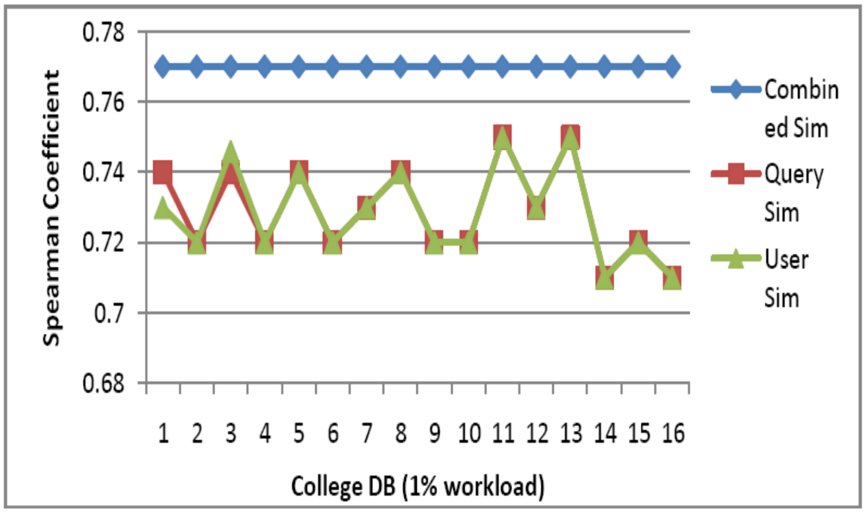


Figure 13 – Ranking quality of combined similarity model

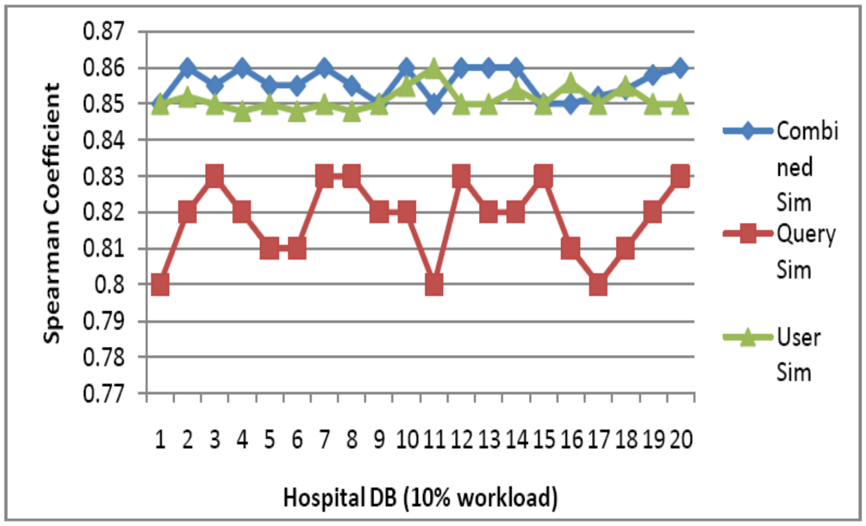


Figure 14 – Ranking quality of combined similarity model

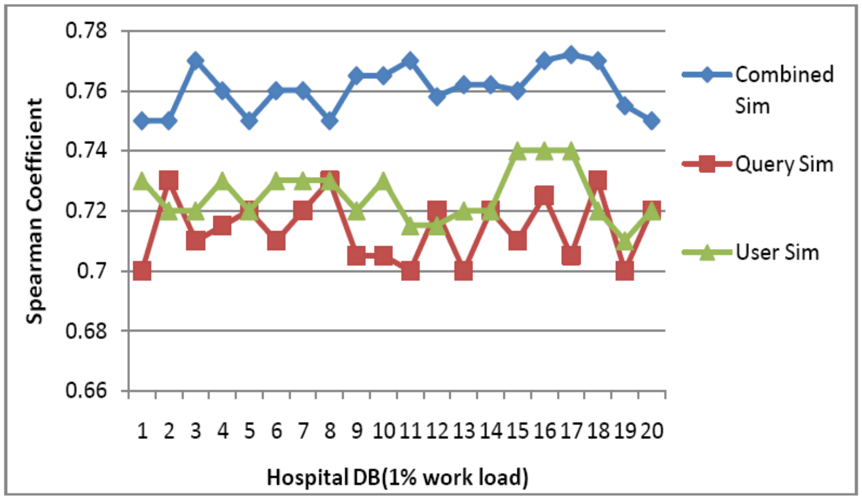


Figure 15 – Ranking quality of combined similarity model

Fig. 12, 13, 14 and 15 show the quality of combined models for both databases with 1% and 10% workload. Theimportant observation is that the composite model is performing better than other individual models. Anotherfact established here is that with more ranking functions in workload better similarity and quality of results isachieved.

VI. CONCLUSION

This paper proposed a new ranking model for ranking query results of databases. We used two synthetic databases for examinations. They are college database and hospital database. The model is basedon both query similarity and user correspondence. We have also created a prototype web based application thatdemonstrates the efficiency of the proposed ranking pattern. A workload file is kept up that continually stores updated ranking functions for both user similarity and query similarity. When a new query is made, this workload file is used for giving ranking to the query results. Designing and keeping up a workload is challenging in the context of web databases. We have implemented an algorithm for estimating user and query similarities and update workload persistently. The examination results disclose that our new ranking pattern works well and it can be explored for real world web databases.

REFERENCES

[1] S. Agrawal, S. Chaudhuri and G. Das.DBXplorer: A Systemfor Keyword Based Search over Relational Databases. ICDE2002.
[2] S. Agrawal, S. Chaudhuri, G. Das and A. Gionis.AutomatedRanking of Database Query Results.Technical Report,Microsoft Research, in preparation.
[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I.Verkamo.Fast Discovery of Association Rules.Advances inKnowledge Discovery and Data Mining, 1995.
[4] R. Baeza-Yates and B. Ribeiro-Neto.Modern InformationRetrieval.ACM Press, 1999.
[5] J. Breese, D. Heckerman and C. Kadie.Empirical Analysisof Predictive Algorithms for Collaborative Filtering.14thConference on Uncertainty in Artificial Intelligence, 1998.
[6] N. Bruno, L. Gravano, and S. Chaudhuri
[7] Top-K Selection Queries over Relational Databases: MappingStrategies and Performance Evaluation. ACM Transactions on Database Systems (TODS), vol. 27, no. 2, June 2002.
[8] N. Bruno, L. Gravano, A. Marian.Evaluating Top-K Queriesover Web-Accessible Databases.ICDE 2002.
[9] M. J. Carey and D. Kossmann. On Saying "EnoughAlready!" in SQL. SIGMOD 1997, 219-230.
[10] M. J. Carey and D. Kossmann. Reducing the BrakingDistance of an SQL Query Engine.VLDB 1998.158-169.
[11] K. Chakrabarti, K. Porkaew and S. Mehrotra.EfficientQuery Ref. in Multimedia Databases.ICDE 2000.
[12] S. Chaudhuri and V. Narasayya.Program for TPC-D DataGeneration with Skew. http://research.microsoft.com/dmx/AutoAdmin.
[13] W. Cohen. Integration of Heterogeneous DatabasesWithout Common Domains Using Queries Based on textualSimilarity. SIGMOD, 1998.
[14] W. Cohen. Providing Database-like Access to the WebUsing Queries Based on Textual Similarity.SIGMOD 1998.
[15] R. Fagin. Fuzzy Queries in Multimedia Database Systems.PODS 1998.
[16] R. Fagin, A. Lotem and M. Naor.Optimal AggregationAlgorithms for Middleware.PODS 2001.
[17] C. Faloutsos and K-I. Lin. Fastmap: A Fast Algorithm forIndexing, Data mining and Visualization of Traditional andMultimedia Datasets. SIGMOD 1995.