One Size Does Not Fit All: Towards User- and Query-Dependent Ranking For Web Databases

Aditya Telang, Chengkai Li, Sharma Chakravarthy
Department of Computer Science and Engineering, University of Texas at Arlington
aditya.telang@mavs.uta.edu, cli@cse.uta.edu, sharma@cse.uta.edu

Abstract—With the emergence of the deep Web, searching Web databases in domains such as vehicles, real estate, etc. has become a routine task. One of the problems in this context is ranking the results of a user query. Earlier approaches for addressing this problem have used frequencies of database values, query logs, and user profiles. A common thread in most of these approaches is that ranking is done in a user- and/or query-independent manner.

This paper proposes a novel query- and user-dependent approach for ranking query results in Web databases. We present a ranking model, based on two complementary notions of *user* and *query* similarity, to derive a ranking function for a given user query. This function is acquired from a sparse workload comprising of several such ranking functions derived for various user-query pairs. The model is based on the intuition that similar users display comparable ranking preferences over the results of similar queries. We define these similarities formally in alternative ways and discuss their effectiveness analytically and experimentally over two distinct Web databases.

Index Terms—Automated Ranking, Web Databases, User Similarity, Query Similarity, Workload.

1 Introduction

The emergence of the deep Web [7] [9] has led to the proliferation of a large number of Web databases for a variety of applications (e.g., airline reservations, vehicle search, real estate scouting). These databases are typically searched by formulating query conditions on their schema attributes. When the number of results returned is large, it is time-consuming to browse and choose the most useful answer(s) for further investigation. Currently, Web databases simplify this task by displaying query results sorted on the values of a single attribute (e.g., Price, Mileage, etc.). However, most Web users would prefer an ordering derived using multiple attribute values, which would be closer to their expectation.

Consider Google Base's [15] *Vehicle* database that comprises of a table with attributes Make, Price, Mileage, Location, Color, etc. where each tuple represents a vehicle for sale. We use the following two scenarios as our running examples.
Example-1: *Two users – a company executive (U_1) and a student (U_2), seek answers to the same query (Q_1): "Make = Honda AND Location = Dallas, TX", for which more than 18,000 tuples are typically returned in response. Intuitively, U_1 would typically search for **new** vehicles with specific **color** choices (e.g., only **red** colored vehicles), and hence would prefer vehicles with* "Condition = New AND Color = Red" *to be ranked and displayed higher than the others. In contrast, U_2 would most likely search for **old** vehicles **priced** under a specific amount (e.g., "Price $<$ 5,000$"); hence, for U_2, vehicles with* "Condition = Old AND Price $<$ 5,000$" *should be displayed before the rest.*
Example-2: *The same student user (U_2) moves to Google for an internship and asks a different query (say Q_4): "Make = Pontiac AND Location = Mountain View, CA". We can presume (since he has procured an internship) that he may be willing to pay a slightly higher **price** for a lesser **mileage** vehicle (e.g., "Mileage $<$ 100,000"), and hence would prefer vehicles with* "Condition = Old AND Mileage $<$ 100,000" *to be ranked higher than others.*

Example-1 illustrates that different Web users may have contrasting ranking preferences towards the results of the same query. *Example-2* emphasizes that the same user may display different ranking preferences for the results of different queries. Thus, it is evident that in the context of Web databases, where a large set of queries given by varied classes of users is involved, the corresponding results should be ranked in a *user*- and *query*-dependent manner.

The current sorting-based mechanisms used by Web databases do not perform such ranking. While some extensions to SQL allow *manual* specification of attribute weights [30] [21] [23] [33], this approach is cumbersome for **most** Web users. *Automated* ranking of database results has been studied in the context of relational databases, and although a number of techniques [10] [11][27] [34] perform *query-dependent ranking*, they do not differentiate between users and hence, provide a single ranking order for a given query across *all users*. In contrast, techniques for building extensive user profiles [20] as well as requiring users to order data tuples [18], proposed for *user-dependent* ranking, do not distinguish between queries and provide a single ranking order for any query given by the same user. Recommendation (i.e., collaborative [17] [5] [8] and content filtering [4] [6] [14]) as well as information retrieval systems use the notions of user- and object/item-similarity for recommending objects to users. Although our work is inspired by this idea, there are differences that prevent its direct applicability to database ranking (elaborated in Section 2).

In this paper, we propose a *user*- and *query-dependent* approach for ranking the results of Web database queries. For a

query Q_j given by a user U_i, a relevant ranking function ($\mathcal{F}_{xy}$) is identified from a workload of ranking functions (inferred for a number of user-query pairs), to rank Q_j's results. The choice of an appropriate function is based on a novel *similarity-based* ranking model proposed in the paper. The intuition behind our approach is: i) for the results of a given query, similar users display comparable ranking preferences, and ii) a user displays analogous ranking preferences over results of similar queries.

We decompose the notion of similarity into: 1) ***query similarity***, and 2) ***user similarity***. While the former is estimated using either of the proposed metrics – *query-condition* or *query-result*, the latter is calculated by comparing individual ranking functions over a set of common queries between users. Although each model can be applied independently, we also propose a unified model to determine an improved ranking order. The ranking function used in our framework is a *linear weighted-sum* function comprising of: i) *attribute-weights* denoting the significance of individual attributes and ii) *value-weights* representing the importance of attribute values.

In order to make our approach practically useful, a minimal workload is important. One way to acquire such a workload is to adapt relevance feedback techniques [16] used in document retrieval systems. However (as elaborated in Section 6), there exist several challenges in applying these techniques to Web databases directly. Although the focus of this paper is on the usage, instead of the acquisition of such workloads, we discuss and compare some potential approaches (in Section 6) for establishing such workloads and elaborate on a learning method for deriving individual ranking functions.

Contributions: The contributions of this paper are:

1) We propose a *user-* and *query-dependent* approach for ranking query results of Web databases.

2) We develop a ranking model, based on two complementary measures of *query similarity* and *user similarity*, to derive functions from a workload containing ranking functions for several user-query pairs.

3) We present experimental results over two Web databases supported by Google Base to validate our approach in terms of efficiency as well as quality for real-world use.

4) We present a discussion on the approaches for acquiring/generating a workload, and propose a learning method for the same with experimental results.

Roadmap: Section 2 discusses the related work and Section 3 formally defines the ranking problem. Sections 4 and 5 explain the *similarity-based* ranking model and discuss our experimental results. In Section 6, we highlight the challenges in generating appropriate workloads and present a *learning method* with preliminary results for deriving a ranking function. Section 7 concludes the paper.

2 Related Work

Although there was no notion of ranking in traditional databases, it has existed in the context of information retrieval for quite some time. With the advent of the Web, ranking gained prominence due to the volume of information being searched/browsed. Currently, ranking has become ubiquitous and is used in document retrieval systems, recommender systems, Web search/browsing, and traditional databases as well. Below, we relate our effort to earlier work in these areas.

Ranking in Recommendation Systems: Given the notion of *user-* and *query-*similarity, it appears that our proposal is similar to the techniques of **collaborative** [17] [5] [8] and **content** filtering [4] [6] [14] used in recommendation systems. However, there are some important differences (between ranking tuples for database queries versus recommending items in a specific order) that distinguish our work. For instance, each cell in the *user-item* matrix of recommendation systems represents a single scalar value that indicates the rating/preference of a particular user towards a specific item. Similarly, in the context of recommendations for social tagging [2] [24] [35], each cell in the corresponding *user-URL/item-tag* matrix indicates the presence or absence of a tag provided by a user for a given URL/item. In contrast, each cell in the *user-query* matrix (used for database ranking) contains an ordered set of tuples (represented by a ranking function). Further, although the rating/relevance given to each tuple (in the results of a given query) by a user can be considered to be similar to a rating given for an item in recommendation systems, if the same tuple occurs in the results of distinct queries, it may receive different ratings from the same user. This aspect of the same item receiving varied ratings by the same user in different contexts is not addressed by current recommendations systems to the best of our knowledge.

Another important distinction that sets our work apart from recommendation systems is the notion of *similarity*. In content filtering, the similarity between items is established either using a domain expert, or user profiles [14], or by using a feature recognition algorithm [4] over the different features of an item (e.g., author and publisher of a book, director and actor in a movie, etc.). In contrast, since our framework requires establishing similarity between actual SQL queries (instead of simple keyword queries), the direct application of these techniques does not seem to be appropriate. To the best of our knowledge, a model for establishing similarity between database queries (expressed in SQL) has not received attention. In addition, a user profile is unlikely to reveal the kind of queries a user might be interested in. Further, since we assume that the same user may have different preferences for different queries, capturing this information via profiles will not be a suitable alternative.

The notion of user similarity used in our framework is identical to the one adopted in collaborative filtering; however, the technique used for determining this similarity is different. In collaborative filtering, users are compared based on the ratings given to individual items (i.e., if two users have given a positive/negative rating for the same items, then the two users are similar). In the context of database ranking, we propose a rigorous definition of user similarity based on the similarity between their respective ranking functions, and hence ranked orders. Furthermore, this work extends user-personalization using context information based on user and query similarity instead of static profiles and data analysis.

Ranking in Databases: Although ranking query results for relational and Web databases has received significant attention over the past years, simultaneous support for automated *user-*

and *query-dependent* ranking has not been addressed in this context. For instance, [10] address the problem of *query-dependent* ranking by adapting the vector model from information retrieval, whereas [11][27] do the same by adapting the probabilistic model. However, for a given query, these techniques provide the same ordering of tuples across all users.

Employing user personalizations by considering the context and profiles of users for *user-dependent* ranking in databases has been proposed in [20]. Similarly, the work proposed in [18] requires the user to specify an ordering across the database tuples, without posing any specific query, from which a global ordering is obtained for each user. A drawback in all these works is that they do not consider that the same user may have varied ranking preferences for different queries.

The closest form of *query-* and *user-dependent* ranking in relational databases involves manual specification of the ranking function/preferences as part of SQL queries [30] [21] [23] [33]. However, this technique is unsuitable for Web users who are not proficient with query languages and ranking functions. In contrast, our framework provides an automated *query-* as well as *user-dependent* ranking solution without requiring users to possess knowledge about query languages, data models and ranking mechanisms.

Ranking in Information Retrieval: Ranking has been extensively investigated in the domain of information retrieval. The cosine-similarity metric [3] is very successful in practice, and we employ its variant [1] for establishing similarities between attribute-value pairs as well as query results in our framework. The problem of integrating information retrieval system and database systems have been attempted [13] with a view to apply the ranking models (devised for the former) to the latter; however, the intrinsic difference between their underlying models is a major problem.

Relevance Feedback: Inferring a ranking function by analyzing the user's interaction with the query results originates from the concepts of relevance feedback [16] [25] [31] [32] [26] [22] in the domain of document and image retrieval systems. However, the direct application of either explicit or implicit feedback mechanisms for inferring database ranking functions has several challenges (Section 6.2), and to the best of out knowledge have not been addressed in literature.

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
U_1	??	$\mathcal{F}_{12}$	–	–	$\mathcal{F}_{15}$	–	$\mathcal{F}_{17}$	–
U_2	$\mathcal{F}_{21}$	$\mathcal{F}_{22}$	–	$\mathcal{F}_{24}$	–	$\mathcal{F}_{26}$	$\mathcal{F}_{27}$	–
U_3	$\mathcal{F}_{31}$	$\mathcal{F}_{32}$	$\mathcal{F}_{33}$	$\mathcal{F}_{34}$	–	–	$\mathcal{F}_{37}$	–

TABLE 1
Sample *Workload-A*

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
U_1	??	–	$\mathcal{F}_{13}$	–	$\mathcal{F}_{15}$	$\mathcal{F}_{16}$	$\mathcal{F}_{17}$	$\mathcal{F}_{18}$
U_2	$\mathcal{F}_{21}$	$\mathcal{F}_{22}$	–	$\mathcal{F}_{24}$	–	–	–	$\mathcal{F}_{28}$
U_3	$\mathcal{F}_{31}$	$\mathcal{F}_{32}$	–	$\mathcal{F}_{34}$	–	–	$\mathcal{F}_{37}$	$\mathcal{F}_{38}$

TABLE 2
Sample *Workload-B*

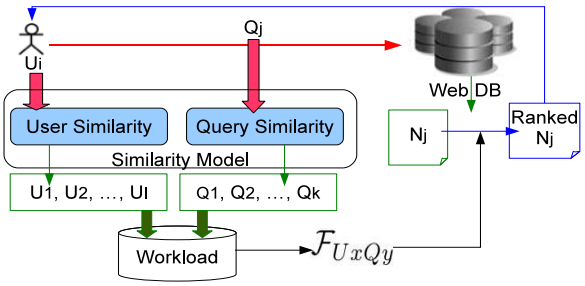

Fig. 1. Similarity Model for Ranking

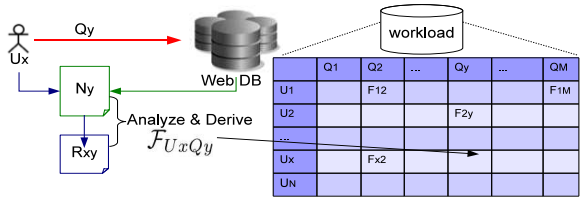

Fig. 2. Workload Generation for Similarity-based Ranking

3 Problem Definition and Architecture

3.1 Problem Definition

Consider a Web database table *D* over a set of *M* attributes, *A* = $\{A_1, A_2, ..., A_M\}$. A user U_i asks a query Q_j of the form: SELECT * FROM *D* WHERE $A_1 = a_1$ AND $\cdots$ AND $A_s = a_s$, where each $A_i \in A$ and a_i is a value in its domain. Let $N_j = \{t_1, t_2, ..., t_n\}$ be the set of result tuples for Q_j, and $\mathbf{W}$ be a workload of ranking functions derived across several user-query pairs (refer to Tables 1 and 2 for an example).

The ranking problem can be stated as: "*For the query Q_j given by the user U_i, determine a* ranking function $\mathcal{F}_{U_iQ_j}$ *from* $\mathbf{W}$". Given the scale of Web users and the large number of queries that can be posed on *D*, $\mathbf{W}$ will not possess a function for every user-query pair; hence the need for a *similarity-based* method to find an acceptable function ($\mathcal{F}_{U_xQ_y}$) in place of the missing $\mathcal{F}_{U_iQ_j}$. The ranking problem, thus, can be split into:

1. *Identifying a ranking function using the similarity model:* Given $\mathbf{W}$, determine a user U_x similar to U_i and a query Q_y similar to Q_j such that the function $\mathcal{F}_{U_xQ_y}$ exists in $\mathbf{W}$.

2. *Generating a workload of ranking functions:* Given a user U_x asking query Q_y, based on U_x's preferences towards Q_y's results, determine, explicitly or implicitly, a ranking function $\mathcal{F}_{U_xQ_y}$. $\mathbf{W}$ is then established as a collection of such ranking functions learnt over different user-query pairs.

The above description refers to *point* queries with conjunctive conditions. However, queries may contain range/IN conditions and several Boolean operators (AND, OR, NOT). However, our focus is on the problem of point queries over a single table. Extensions are being explored as future work.

3.2 Ranking Architecture

The Similarity model (shown in Figure 1) forms the core component of our ranking framework. When the user U_i poses the query Q_j, the *query-similarity* model determines the set of queries ($\{Q_j, Q_1, Q_2, ..., Q_p\}$) most similar to Q_j. Likewise, the *user-similarity* model determines the set of users ($\{U_i, U_1, U_2, ...U_r\}$) most similar to U_i. Using these two ordered sets of similar queries and users, it searches the workload to identify the function $\mathcal{F}_{U_xQ_y}$ such that the

combination of U_x and Q_y is most similar to U_i and Q_j. $\mathcal{F}_{U_x}Q_y$ is then used to rank Q_j's results for U_i.

The workload used in our framework comprises of ranking functions for several user-query pairs. Figure 2 shows the high level view of deriving an individual ranking function for a user-query pair (U_x, Q_y). By analyzing U_x's preferences (in terms of a selected set of tuples (R_{xy})) over the results (N_y), an approximate ranking function ($\mathcal{F}_{U_x}Q_y$) can be derived.

As our ranking function is of the linear weighted-sum type, it is important that the mechanism used for deriving this function captures the: i) significance associated by the user to each attribute i.e., an *attribute-weight* and ii) user's emphasis on individual values of an attribute i.e., a *value-weight*. These weights can then be integrated into a ranking function $\mathcal{F}_{xy}$ to assign a *tuple score* to every tuple t in N_y using Equation 1:

$$tuplescore(t) = \sum_{i=1}^{m} w_i * v_i \quad (1)$$

where w_i represents the *attribute-weight* of A_i and v_i represents the *value-weight* for A_i's value in tuple t.

The workload **W** is populated using such ranking functions. Tables 1 and 2 show two instances of the workload (represented in the form of a matrix of users and queries). Cell [x,y] in the workload, if defined, consists of the ranking function $\mathcal{F}_{xy}$ for the user-query pair U_x and Q_y.

4 Similarity Model for Ranking

The concept of similarity-based ranking is aimed at situations when the ranking functions are known for a small (and hopefully representative) set of user-query pairs. At the time of answering a query asked by a user, if no ranking function is available for this user-query pair, the proposed *query-* and *user-similarity* models can effectively identify a suitable function to rank the corresponding results.

4.1 Query Similarity

For the user U_1 from *Example-1*, a ranking function does not exist for ranking Q_1's results (N_1). From the sample *workload-A*[1] shown in Table 1, ranking functions over queries Q_2, Q_5, Q_7 (shown in Table 3) have been derived; thus, forming U_1's workload. It would be useful to analyze if any of $\mathcal{F}_{12}$, $\mathcal{F}_{15}$, or $\mathcal{F}_{17}$ can be used for ranking Q_1's results for U_1. However, from *Example-2*, we know that a user is likely to have displayed different ranking preferences for different query results. Consequently, a randomly selected function from U_1's workload is not likely to give a desirable ranking order over N_1. On the other hand, the ranking functions are likely to be comparable for queries similar to each other.

We advance the hypothesis that if Q_1 is most similar to query Q_y (in U_1's workload), U_1 would display similar ranking preferences over the results of both queries; thus, the ranking function ($\mathcal{F}_{1y}$) derived for Q_y can be used to rank N_1. Similar to recommendation systems, our framework can utilize the an aggregate function, composed from the functions corresponding to the *top-k* most similar queries to Q_1, to rank N_1. Although the results of our experiments showed that an aggregate function works well for certain individual instances of users asking particular queries, on average across all users asking a number of queries, using an individual function proved better than an aggregate function. Hence, for the reminder of the section, we only consider the most similar query (to Q_1). We translate this proposal of *query similarity* into a principled approach via two alternative models: i) *query-condition* similarity, and ii) *query-result* similarity.

4.1.1 Query-Condition Similarity

In this model, the *similarity* between two queries is determined by comparing the attribute values in the query conditions. Consider *Example-1* and the queries from Table 3. Intuitively, "Honda" and "Toyota" are vehicles with similar characteristics i.e., they have similar prices, mileage ranges, and so on. In contrast, "Honda" is a very different from "Lexus". Similarly, "Dallas" and "Atlanta", both being large metropolitan cities, are more similar to each other than "Basin", a small town.

From the above analysis, Q_1 appears more similar to Q_2 than Q_5. In order to validate this intuitive similarity, we examine the relationship between the different values for each attribute in the query conditions. For this, we assume independence of schema attributes, since, availability of appropriate knowledge of functional dependencies and/or attribute correlations is not assumed.

Definition Given two queries Q and Q', each with the conjunctive selection conditions, respectively of the form "WHERE A_1=a_1 AND $\cdots$ AND A_m=a_m" and "WHERE A_1=a'_1 AND $\cdots$ AND A_m=a'_m" (where a_i or a'_i is 'any'[2] if A_i is not specified), the *query-condition* similarity between Q and Q' is given as the conjunctive similarities between the values a_i and a'_i for every attribute A_i (Equation 2).

$$similarity(Q, Q') = \prod_{i=1}^{m} sim(Q[A_i = a_i], Q'[A_i = a'_i]) \quad (2)$$

In order to determine the right-hand-side (RHS) for the above equation, it is necessary to translate the intuitive similarity between values (e.g., "Honda" is more similar to "Toyota" than it is with "Lexus") to a formal model. This is achieved by determining the similarity between databases tuples corresponding to point queries with these attribute values. For instance, consider the values "Honda", "Toyota" and "Lexus" for the attribute "Make". The model generates three distinct queries (Q_H, Q_T and Q_L) with the conditions – "Make =

1. For Web databases, although the workload matrix can be extremely large, it is very sparse as obtaining preferences for large number of user-query pairs is practically difficult. We have purposely shown a dense matrix to make our model easily understandable.

query	make	location	price	mileage	color
Q_1	Honda	Dallas	any	any	any
Q_2	Toyota	Atlanta	any	any	any
Q_5	Lexus	Basin	any	any	any
Q_7	any	Little Rock	any	any	Grey

TABLE 3
Input query (Q_1) and U_1's Workload

2. The value 'any' represents a union of all values for the domain of the particular attribute. For example, a value of 'any' for the Transmission attribute retrieves cars with 'manual' as well as 'auto' transmission.

tupleID	make	location	price (in $)	mileage	color
t1	Honda	Dallas	20-25K	10-25K	red
t2	Honda	Atlanta	20-25K	25-50K	red
t3	Honda	Boston	25-30K	0-10K	green
...	...	...	...	...	...

TABLE 4
Sample results (N_H) for query "make = Honda"

tupleID	make	location	price (in $)	mileage	color
t1	Toyota	Little Rock	5-10K	125-150K	red
t2	Toyota	Raleigh	15-20K	25-50K	green
t3	Toyota	Atlanta	20-25K	10-25K	silver
...	...	...	...	...	...

TABLE 5
Sample results (N_T) for query "make = Toyota"

tupleID	make	location	price (in $)	mileage	color
t1	Lexus	Boston	35-40K	0	silver
t2	Lexus	Detroit	35-40K	0	black
t3	Lexus	Urbana	30-35K	0-10K	red
...	...	...	...	...	...

TABLE 6
Sample results (N_L) for query "make = Lexus"

Honda", "Make = Toyota" and "Make = Lexus" respectively, and obtains the individual sets of results N_H, N_T and N_L (shown[3] in Tables 4, 5, and 6). It can be observed that the tuples for "Toyota" and "Honda" display a high degree of similarity over multiple attributes as compared to the tuples for "Lexus" indicating that the former two attribute values are more similar to each other than the latter. The similarity between each pair of query results (i.e., [N_H, N_T], [N_H, N_L], [N_T, N_L]) is then translated as the similarity between the respective pairs of attribute values[4].

Formally, we define the similarity between any two values a_1 and a_2 for an attribute A_i as follows. Two queries Q_{a_1} and Q_{a_2} with the respective selection conditions: "WHERE $A_i = a_1$" and "WHERE $A_i = a_2$" are generated. Let N_{a_1} and N_{a_2} be the set of results obtained from the database for these two queries. The similarity between a_1 and a_2 is then given as the similarity between N_{a_1} and N_{a_2}, and is determined using the variant of the *cosine-similarity* model [1]). Given two tuples $T =< t_1, t_2, ..., t_m >$ in N_{a1} and $T' =< t'_1, t'_2, ..., t'_m >$ in N_{a2}, the similarity between T and T' is:

$$sim(T, T') = \sum_{i=1}^{m} sim(t_i, t'_i) \tag{3}$$

where

$$sim(t_i, t'_i) = \begin{cases} 1 & \text{if } t_i = t'_i, \\ 0 & \text{if } t_i \neq t'_i. \end{cases} \tag{4}$$

It is obvious that Equation 4 will work improperly for numerical attributes where exact matches are difficult to find across tuple comparisons. In this paper, we assume that numerical data has been discretized in the form of histograms (as done for query processing) or other meaningful schemes (as done by Google Base; shown for the values of 'price' and 'mileage' in Tables 4, 5 and 6). The existence of a (reasonable) discretization is needed for our model (instead of its justification which is beyond the scope of the paper).

3. In the interest of space, we have displayed only the top-3 query results returned by Google Base. For the sake of readability, only five out of the eleven attributes from the Vehicle database schema are shown.

4. The similarity between an attribute value (e.g., Honda) and the value 'any' is estimated as the average similarity between Honda and every other value in the domain of the corresponding attribute.

tupleID	make	location	price (in $)	mileage	color
t1	Honda	Dallas	15-20K	25-50K	red
t2	Honda	Dallas	15-20K	25-50K	red
t3	Honda	Dallas	20-25K	0-10K	green
...	...	...	...	...	...

TABLE 7
Sample results of Q_1 from Table 3

tupleID	make	location	price (in $)	mileage	color
t1	Toyota	Atlanta	15-20K	25-50K	red
t2	Toyota	Atlanta	15-20K	25-50K	green
t3	Toyota	Atlanta	20-25K	10-25K	silver
...	...	...	...	...	...

TABLE 8
Sample results of Q_2 from Table 3

tupleID	make	location	price (in $)	mileage	color
t1	Lexus	Basin	35-40K	new	silver
t2	Lexus	Basin	35-40K	new	black
t3	Lexus	Basin	30-35K	0-10K	red
...	...	...	...	...	...

TABLE 9
Sample results of Q_5 from Table 3

Using Equation 3, the similarity between the two sets N_{a1} and N_{a2} (which in turn, corresponds to the similarity between the values a_1 and a_2) is estimated as the average pair-wise similarity between the tuples in N_{a1} and N_{a2} (Equation 5).

$$sim(N_{a1}, N_{a2}) = \frac{\sum_{i=1}^{|N_{a1}|} \sum_{j=1}^{|N_{a2}|} sim(T_i, T'_j)}{|N_{a1}| \cdot |N_{a2}|} \tag{5}$$

These similarities between attribute values can then be substituted into Equation 2 to estimate *query-condition* similarity.

4.1.2 Query-Result Similarity

In this model, *similarity* between a pair of queries is evaluated as the similarity between the tuples in the respective query results. The intuition behind this model is that if two queries are similar, the results are likely to exhibit greater similarity.

For the queries in Table 3, let the results shown in Tables 7, 8, and 9, respectively, correspond to a sample set (again top-3 results and five attributes displayed) for Q_1, Q_2 and Q_5. We observe that there exists certain similarity between the results of Q_1 and Q_2 for attributes such as 'price' and 'mileage' (and even 'color' to a certain extent). In contrast, the results of Q_5 are substantially different; thus, allowing us to infer that Q_1 is more similar to Q_2 than Q_5. Formally, we define *query-result similarity* below.

Definition Given two queries Q and Q', let N and N' be their query results. The *query-result* similarity between Q and Q' is then computed as the similarity between the result sets N and N', given by Equation 6.

$$similarity(Q, Q') = sim(N, N') \tag{6}$$

The similarity between the pair of results (N and N') is estimated using Equations 3, 4 and 5.

4.1.3 Analysis of Query Similarity Models

The computation of similarity for the two models discussed above is summarized in Figure 3. While the *query-condition similarity* uses the conjunctive equivalence between individual

attribute values, the *query-result similarity* performs a holistic comparison between the results. Below we discuss the accuracy and computational efficiency of the two models.

Accuracy: Intuitively, similarity between queries depends on the proximity between their respective query conditions. For instance, "Honda" and "Toyota" are cars with a lot of common features which reflects in the tuples representing these values in the database, and hence, queries that search for these vehicles are more similar than queries asking for "Lexus". In contrast, two queries may return very similar results although their conditions could be quite dissimilar. For example, the following two queries on Google Base – "Make = Mercedes AND Color = Lilac", and "Location = Anaheim, CA AND Price $>$ 35,000$", end up returning exactly the same set of results. Although these are very similar from the *query-result similarity* definition, the queries, in fact, are intuitively not similar. In general, similar queries are likely to generate similar results; however, the converse is not necessarily true. Hence the *query-condition similarity* model is expected to be more accurate and consistent than the *query-result similarity* model which is also borne out by the experiments.

Computational Efficiency: Both query similarities can be computed by applying queries over the Web database i.e., direct access to the data is not needed, which can be difficult for Web database such as Google Base (a collection of multiple databases). One alternative toward efficiently implementing these similarities would be to pre-compute them and use the respective values at query time[5].

In order to distinguish the two models, consider the workload $\mathbf{W}$ and assume a schema of M attributes. For the sake of simplicity, let the domain of each attribute have n values. The *query-condition* model relies purely on the pairwise similarities between attribute values, and hence can be independently pre-computed (generating $M * n$ queries, one for each value of every attribute, and performing $M * n^2$ computations). At query-time, the similarity between input (Q_i) and every query in $\mathbf{W}$ can be estimated by a lookup of the attribute-value similarities. In contrast, the *query-result* model requires a comparison between the query results. Since an input query cannot be predicted, pre-computation is not possible (unless every possible query on the database can be generated and compared – a combinatorial explosion scenario); hence, at query-time, a costly computation of similarities between Q_i with every query in $\mathbf{W}$ would be required.

Thus, intuitively, the *query-condition* model is superior to the *query-result* model. Also, computation of similarity by the *query-condition* model is tractable as compared to the *query-result* model, an observation substantiated by our experiments.

4.2 User Similarity

For *Example-1*, we need to determine a function ($\mathcal{F}_{11}$) to rank the results (N_1) for U_1. The *workload-A* (Table 1) shows that for users U_2 and U_3, functions $\mathcal{F}_{21}$ and $\mathcal{F}_{31}$ are available for

5. Pre-computation assumes that the underlying data does not change significantly over time. Considering the size of these databases, small percent of changes are unlikely to affect the similarity computations; however it is possible to re-compute the similarities periodically.

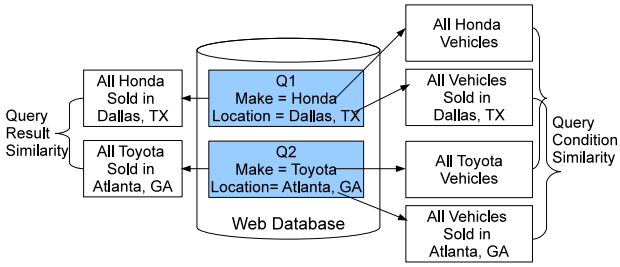

Fig. 3. Summarized view of the *query-similarity* models

Q_1. It would be useful to determine if any of these functions can be used to in place of $\mathcal{F}_{11}$. However, we know from *Example-1* that different users may display different ranking preferences towards the same query. Thus, to determine a function, instead of randomly picking one from $\mathbf{W}$, that would provide a reasonable ranking of Q_1's results for U_1, we propose the notion of *user similarity*.

We put forward the hypothesis that if U_1 is similar to an existing user U_x, then, for the results of a given query (say Q_1), both users will show similar ranking preferences; therefore, U_x's ranking function ($\mathcal{F}_{x1}$) can be used to rank Q_1's results for U_1 as well. Again, as explained in Section 4.1, instead of using a *single* most similar user (to U_1), our framework can be extended to determine *top-k* set of most similar users to establish *user-similarity*. However, like *query-similarity*, an aggregate ranking function did not provide significant improvement in the ranking quality; hence, we only consider the most similar user (to U_1) in our discussion.

In order to translate the hypothesis of *user-similarity* into a model, we need to understand how to compute similarity between a given pair of users. In this paper, we estimate it based on the similarity of users' individual ranking functions over different common queries in the workload.

Definition Given two users U_i and U_j with the set of common queries – $\{Q_1, Q_2, ..., Q_r\}$,[6] for which ranking functions ($\{\mathcal{F}_{i1}, \mathcal{F}_{i2}, ..., \mathcal{F}_{ir}\}$ and $\{\mathcal{F}_{j1}, \mathcal{F}_{j2}, ..., \mathcal{F}_{jr}\}$) exist in $\mathbf{W}$, the *user similarity* between U_i and U_j is expressed as the average similarity between their individual ranking functions for each query Q_p (shown in Equation 7):

$$similarity(U_i, U_j) = \frac{\sum_r sim(\mathcal{F}_{ip}, \mathcal{F}_{jp})}{r} \quad (7)$$

In order to determine the right-hand-side of the Equation 7, it is necessary to quantify a measure that establishes the similarity between a given pair of ranking functions. We use the *Spearman's rank correlation coefficient (ρ)* to compute similarity between the sets obtained by applying these ranking functions on the query results. We choose the *Spearman coefficient* based on the observations [12] regarding its usefulness, with respect to other metrics, in comparing ranked lists.

Consider two functions $\mathcal{F}_{i1}$ and $\mathcal{F}_{j1}$ derived for a pair of users for the same query Q_1 that has results N_1. We apply these two functions individually on N_1 to obtain two ranked sets of results – $N_{R_{i1}}$ and $N_{R_{j1}}$. If the number of tuples in the result sets is $\mathcal{N}$, and d_i is the difference between the ranks of the same tuple (t_i) in $N_{R_{i1}}$ and $N_{R_{j1}}$, then we express

6. Without loss of generality, we assume $\{Q_1, Q_2, ..., Q_r\}$ are the common queries for U_i and U_j, although they can be any queries.

For Query	User Similarity
Q_2	sim(U_1, U_2) > sim(U_1, U_3)
Q_7	sim(U_1, U_2) < sim(U_1, U_3)
Q_2, Q_7	sim(U_1, U_2) < sim(U_1, U_3)

TABLE 10
Drawbacks of Query-Independent User Similarity

the similarity between $\mathcal{F}_{i1}$ and $\mathcal{F}_{j1}$ as the *Spearman's rank correlation coefficient* given by Equation 8:

$$sim(F_{i1}, F_{j1}) = 1 - \frac{6 * \sum_{i=1}^{\mathcal{N}} d_i^2}{\mathcal{N} * (\mathcal{N}^2 - 1)} \quad (8)$$

In our method for estimating *user similarity*, we have considered **all** the queries that are common to a given pair of users. This assumption forms one of our models for user similarity termed *query-independent user similarity*. However, it might be useful to estimate user similarity based on only those queries that are similar to the input query Q_1. In other words, in this hypothesis, two users who may not be very similar to each other over the *entire workload* comprising of similar and dissimilar queries, may in fact, be very similar to each other over a smaller set of *similar queries*. We formalize this hypothesis using two different models – i) *clustered*, and ii) *top-K* – for determining *user similarity*.

Before explaining these models, we would like to point out that given a workload, if no function exists for any query for a user, estimating similarity between that user and any other user is not possible. Consequently, no ranking is possible in such a scenario. For the rest of the discussion, we assume that all users have at least one ranking function in the workload.

4.2.1 Query-Independent User Similarity

This model follows the simplest paradigm and estimates the similarity between a pair of users based on ***all*** the queries common to them. For instance, given *workload-A* in Table 1, this model determines the similarity between U_1 and U_2 using the ranking functions of Q_2 and Q_7. From the queries in Table 3, let the *query-similarity* model indicate that Q_2 is *most* similar to Q_1 whereas Q_7 is *least* similar to Q_1, and let us consider the *user-similarity* results be as shown in Table 10.

This model will pick U_3 as the most similar user to U_1. However, if only Q_2 (which is most similar to Q_1) is used, U_2 is more similar to U_1. Based on our premise that similar users display similar ranking preferences over the results of similar queries, it is reasonable to assume that employing $\mathcal{F}_{21}$ to rank Q_1's results would lead to a better ranking order (from U_1's viewpoint) than the one obtained using $\mathcal{F}_{31}$. The failure to distinguish between queries is thus a potential drawback of this model, which the following models aim to overcome.

4.2.2 Cluster-based User Similarity

In order to meaningfully restrict the number of queries that are similar to each other, one alternative is to *cluster* queries in the workload based on query similarity. This can be done using a simple *K-means* clustering method [19]. Given an existing workload of m queries (Q_1, Q_2, ... Q_m), each query (Q_j) is represented as a m-dimensional vector of the form $<s_{j1}, ..., s_{jm}>$ where s_{jp} represents the *query-condition similarity* score between the queries Q_j and Q_p (by Equation 2). Using *K-means*, we cluster m queries into K clusters based on a pre-defined K and number of iterations.

Consider *Example-1* and the queries in Table 3. Assuming the similarities specified in Section 4.2.1 (Q_2 and Q_7 are most and least similar to Q_1 respectively), for a value of $K = 2$, the *simple K-means* algorithm will generate two clusters – C_1 containing Q_1 and Q_2 (along with other similar queries), and C_2 containing Q_7 (in addition to other queries not similar to Q_1). We then estimate the similarity between U_1 and every other user only for the cluster C_1 (since it contains queries most similar to the input query). Using the scenario from Table 10, U_2 would be chosen as the most similar user and $\mathcal{F}_{21}$ would be used to rank the corresponding query results.

The above model assumes that ranking functions are available for reasonable number of queries in each cluster. However, as the workload is likely to be sparse for most Web databases, it is possible that no ranking functions are available in the cluster most similar to the incoming query. For example, considering the *workload-B* in Table 2 and assuming a cluster C_1 of queries Q_1, Q_2, Q_3 and Q_4, due to the lack of ranking functions, no similarity can be established between U_1 and other users. Consequently, the similarities would then be estimated in other clusters, thus hampering the quality of the ranking achieved due to dissimilarity between the input query and the queries in the corresponding clusters.

A well-established drawback of using a cluster-based alternative is the choice of K. In a Web database, a small value of K would lead to a large number of queries in every cluster, some of which may not be very similar to the rest, thus, affecting the overall user similarity. In contrast, a large value of K would generate clusters with few queries, and in such cases, the probability that there exist no users with any function in the cluster increases significantly.

4.2.3 Top-K User Similarity

Instead of finding a reasonable K for clustering, we propose a refinement, termed *top-K user similarity*. We propose three measures to determine *top-K* queries that are most similar to an input query (say Q_1 from *Example-1*), and estimates the similarity between the user (U_1) and every other user.

Strict top-K user similarity: Given an input query Q_1 by U_1, *only* the *top-K* most similar queries to Q_1 are selected. However, the model does not check the presence (or absence) of ranking functions in the workload for these K queries. For instance, based on assumption in Section 4.2.4, and using $K = 3$, Q_2, Q_3 and Q_4 are the three queries most similar to Q_1, and hence, would be selected by this model.

In the case of *workload-A*, similarity between U_1 and U_2 as well as between U_1 and U_3 will be estimated using Q_2. However, in the case of *workload-B*, similar to the problem in the *clustering* alternative, there is no query common between U_1 and U_2 (as well as U_3). Consequently, similarity cannot be established and hence, no ranking is possible.

User-based top-K user similarity: In this model, we calculate *user similarity* for a given query Q_1 by U_1, by selecting *top-K* most similar queries to Q_1, each of which has a ranking function for U_1. Consequently for *workload-A*, using $k = 3$, the queries Q_2, Q_5 and Q_7 would be selected. Likewise, in

the case of *workload-B*, this measure would select Q_3, Q_5 and Q_6 using the 'top-3' selection. However, since there exist no function for users U_2 and U_3 (in *workload-B*) given these queries, no similarity can be determined, and consequently, no ranking would be possible.

Workload-based top-K user similarity: In order to address the problems in previous two models, we propose a *workload-based top-K* model that provides the stability of the *query-independent model* (in terms of ensuring that ranking is *always* possible, assuming there is at least one non-empty cell in the workload for that user) and ensures that similarity between users can be computed in a *query-dependent* manner.

Given a Q_1 by U_1, the *top-K* most similar queries to Q_1 are selected such that for each of these queries, there exists: i) a ranking function for U_1 in the workload, and ii) a ranking function for *at least one* other user (U_i) in the workload. Considering $k = 3$, this model will select Q_2, Q_5 and Q_7 in the case of *workload-A* and the queries Q_7 and Q_8 for *workload-B*, and ensure a ranking of results in every case.

Algorithm 1 Deriving Ranking Functions from Workload

INPUT: U_i, Q_j, Workload $\mathbf{W}$ (M queries, N users)
OUTPUT: Ranking Function $\mathcal{F}_{xy}$ to be used for U_i, Q_j
STEP ONE:
for $p = 1$ to M **do**
  %% Using Equation 2 %%
  Calculate Query Condition Similarity (Q_j, Q_p)
end for
%% Based on descending order of similarity with Q_j %%
Sort(Q_1, Q_2, ... Q_M)
Select Q_{Kset} i.e., top-K queries from the above sorted set
STEP TWO:
for $r = 1$ to N **do**
  %% Using Equation 7 %%
  Calculate User Similarity (U_i, U_r) over Q_{Kset}
end for
%% Based on descending order of similarity with U_i %%
Sort(U_1, U_2, ... U_N) to yield U_{set}
STEP THREE:
for Each $Q_s \in Q_{Kset}$ **do**
  for Each $U_t \in U_{set}$ **do**
    Rank(U_t,Q_s) =
    Rank($U_t \in U_{set}$) + Rank($Q_s \in Q_{Kset}$)
  end for
end for
$\mathcal{F}_{xy}$ = Get-RankingFunction()

4.2.4 Summary of User Similarity Models

Intuitively, *query-dependent* estimation of user similarity is likely to yield better ranking as compared to the *query-independent* model. Also, barring the *workload-based top-K* model, ranking may not *always* be possible in other *query-dependent* models. In order to overcome this, the framework can always dynamically choose the top-K queries at runtime in the following order: *strict*, *user-based*, and *workload-based*. The *strict top-k* gives the best results as can be understood intuitively and has been ascertained experimentally. The alternatives extend the availability of ranking functions with reduced accuracy. Additionally, as our experiments show, the choice of *K* in *top-K* is not as critical as the selection of *K* in *K-means* algorithm. Thus, the *top-K* model seems to be the best alternative in a query-dependent environment.

We would also like to point out that since this framework is based on the notion of similarity, we could have used the similarity notion used in recommender systems that is typically based on user profiles. However, our premise was that the profile-based systems provide static or context-independent similarity (i.e., a users behavior does not change across all items/objects) whereas we believe that ranking of results (unlike recommending) varies substantially for the same user based on the query type (i.e., is context-dependent) and attributes specified (as elaborated by *Example-2* in Section 1). This work can be treated as a generalization that uses information beyond profiles and seems appropriate for the Web database context.

4.3 The Composite Similarity Model

In order to derive a user's (U_i) ranking function for a query (Q_j), we have proposed two independent approaches based on *user* and *query* similarity. However, given the scale of Web users and queries, and the sparseness of the workload, applying only one model may not be the best choice at all times.

Considering *Example-1* and *Workload-B* (Table 2), we want to identify a ranking function to rank Q_1's results for U_1. Using only the *query-similarity* model, $\mathcal{F}_{13}$ will be selected since Q_3 is most similar to Q_1. In contrast, applying only *user-similarity* model will yield $\mathcal{F}_{21}$ as U_2 is most similar to U_1. It would be meaningful to rank these functions ($\mathcal{F}_{13}$ and $\mathcal{F}_{21}$) to choose the most appropriate one. Furthermore, in a more practical setting, the workload is likely to have a ranking function for a similar query (to Q_1) derived for a similar user (to U_1). For instance, the likelihood of $\mathcal{F}_{22}$ existing in the workload would be higher than the occurrance of either $\mathcal{F}_{13}$ or $\mathcal{F}_{21}$. Hence, it would be meaningful to combine the two measures into a single *Similarity Model*.

The goal of this composite model is to determine a ranking function ($\mathcal{F}_{xy}$) derived for the most similar query (Q_y) to Q_j given by the most similar user (U_x) to U_i to rank Q_j's results. The process for finding such an appropriate ranking function is given by the Algorithm 1.

The input to the algorithm is a user (U_i) and a query (Q_j) along with the workload matrix (W) containing ranking functions. The algorithm begins by determining the *query-condition similarity* (STEP ONE) between Q_j and every query in the workload. It then sorts all these queries (in descending order) based on their similarity with Q_j and selects the set (Q_{Kset}) of the top-K most similar queries to Q_j that satisfy the conditions for the *top-K user similarity* model. Based on these selected queries, the algorithm determines the *user-similarity* (STEP TWO) between U_i and every user in the workload. All the users are then sorted (again, in descending order) based on their similarity to U_i. We then generate a list of all the user-query pairs (by combining the elements from the two sorted sets), and linearise these pairs by assigning a rank (which is the sum of query and use similarity ranks) to each pair (STEP THREE). For instance, if U_x and Q_y occur as the x^{th} and y^{th} elements in the respective ordering with the input pair, the pair (U_x, Q_y) are assigned an aggregate rank. In this case, a rank of "$x + y$" will be assigned. The "Get-RankingFunction" method then selects the pair (U_x, Q_y) that has the lowest combined rank and contains a ranking function ($\mathcal{F}_{xy}$) in the workload. Then, in order to rank the results (N_j),

the corresponding attribute weights and value weights obtained for $\mathcal{F}_{xy}$ will be individually applied to each tuple in N_j (using Equation 1), from which a general ordering of all tuples will be achieved.

Algorithm 1 only displays a higher-level view of finding a desired function using the *Similarity Model*. However, the process of estimating *query similarities*, *user similarities* as well as *matrix traversal* can be costly with respect to time. Applying adequate indexing techniques where estimating *query similarity* can be reduced to a simple lookup of similarities between attribute value-pairs, pre-computation of *user similarity* to maintain an indexed list of similar users for every user, and maintaining appropriate data structures to model the matrix traversal as a mere lookup operation are some of the preliminary approaches that we adopted to establish a workable ranking framework. We discuss some of the preliminary results of efficiently using our framework in Section 5. Although there is great scope for devising techniques to make the system scalable and efficient, the focus of this work was to propose techniques to establish good ranking quality.

5 Experimental Evaluation

We have evaluated each proposed model (*query-similarity* and *user-similarity*) in isolation, and then compared both these models with the *combined* model for *quality/accuracy*. We also evaluated the *efficiency* of our ranking framework.

Ideally, we would have preferred to compare our approach against existing ranking schemes in databases. However, what has been addressed in literature is the use of exclusive profiles for user-based ranking (the techniques for the same do not distinguish between queries) or the analysis of the database in terms of frequencies of attribute values for query-dependent ranking (which does not differentiate between users). In the context of Web databases like Google Base, the data is obtained on-the-fly from a collection of data sources; thus, obtaining the entire database for determining the individual ranking functions, for comparing with query-dependent ranking techniques, is difficult. Even if we obtain ranking functions for different queries, all users will see the same ranking order for a given query. Thus, comparing such static ordering of tuples against our approach (that determines distinct ranking of tuples for each user and query separately) would not be a meaningful/fair comparison. Similarly, we felt that the comparing static user profiles (that ignore the different preferences of the same user for different queries) to our proposed definition of user similarity, for user-dependent ranking will not be fair. Hence we have tried to compare the proposed user, query, and combined similarities to indicate the effectiveness of each model with respect to the other two models.

Further, the context of recommendation systems is different from the one considered in this paper; hence, the direct application of these techniques for comparing against our framework was not feasible. We would also like to point out that unlike information retrieval, there are no standard benchmarks available and hence, we had to rely on controlled user studies for evaluating our framework.

Q_1	"Make=Honda AND Location=Dallas,TX AND Price<10,000$"
Q_2	"Make=Toyota AND Location=Miami,FL AND Price<8,000$"
...	...
Q_{10}	"Location=Chicago,IL AND Mileage<100,500 AND Year>2004"
...	...

TABLE 11
Sample Queries from the *Vehicle* database

Q_1	"Location=Dallas,TX AND Beds=3 AND To=Buy"
Q_2	"Location=Denton, TX AND Beds=2 AND Bath=2 AND To=Buy"
...	...
Q_{16}	"Location=Dallas, TX AND Type=Townhouse AND To = Rent"
...	...

TABLE 12
Sample Queries from the *Real Estate* database

5.1 Setup

We used two real Web databases provided by Google Base. The first is a *vehicle* database comprising of 8 categorical/discretized attributes (Make, Vehicle-Type, Mileage, Price, Color, etc.). Although Price and Mileage are numeric, they are discretized apriori by Google into meaningful ranges. In addition, for every attribute, the domain of values (e.g., 'Chevrolet', 'Honda', 'Toyota', 'Volkswagen', ... for the Make attribute) is provided. The second is a *real estate* database with 12 categorical/discretized attributes (Location, Price, House Area, Bedrooms, Bathrooms, etc.). Google provides APIs for querying its databases, and returns a maximum of 5000 results for every API query. Our experiments were performed on a Windows XP Machine with a 2.6 GHz Pentium 4 processor and 4 GB RAM. All algorithms were implemented in Java.

5.2 Workload Generation and User Studies

Our framework utilizes a workload comprising of users, queries and ranking functions (derived over a reasonable set of user queries). Currently, user and query statistics are not publicly available for any existing Web databases. Furthermore, establishing a real workload of users and queries for a Web database would require significant support from portals supporting Web databases such as Yahoo or Google – a task beyond the scope of this paper. Hence, to experimentally validate the quality of our framework, we had to rely on controlled user studies for generating the workload. For each database, we initially generated a pool of 60 random queries (comprising of conditions based on randomly selected attributes and their values), and manually selected 20 representative queries that are likely to be formulated by real users. Tables 11 and 12 show three such queries over each database.

We then conducted two separate surveys (one for each database) where every user was shown 20 queries (one-at-a-time) and asked to input, for each query, a ranking function by assigning weights to the schema attributes (on a scale of 1 to 10). For aiding the user in expressing these preferences, we also displayed the set of results returned for each query. In reality, collecting functions explicitly from users for forming the workload is far from ideal; however, the focus of this paper was on using, instead of establishing, a workload for performing similarity-based ranking. Although generating a larger set of queries would have been ideal for testing the framework, asking users to interact with more than 20 queries

would have been difficult. We would also like to indicate that as is the case with most user studies in the domain of databases, obtaining a large number of users and queries to participate in the corresponding survey is difficult and hence, these numbers are typically very small (as seen by the user surveys conducted for database ranking in [11] [27] [20]).

Each *explicit* ranking provided by a user for a particular query was then stored in the associated workload $\mathbf{W}$. The *vehicle* database survey was taken by 55 users whereas the *real estate* database survey was taken by 75 users (graduate students and faculty members) who provided ranking functions for all the queries displayed to them. Thus, we generated a workload of 1100 ranking functions for the *vehicle* database and 1500 functions for the *real estate* database.

It is evident that more the number of functions in the workload, better will be the quality of ranking achieved (since more similar queries and users would be found for whom functions exist in the workload). This hypothesis was further validated when we achieved a better ranking quality when 50% of the workload was filled (i.e., 50% of the functions were masked out) as compared to the one achieved when the workload contains 25% and 10% of the total ranking function. However, for most Web databases, the workloads would generally be very sparse since obtaining ranking functions across a large number of user-query pairs would be practically difficult. In order to show the effectiveness of our model for such scenarios, in this paper, we present the results when ranking functions exist for only 10% workload (i.e., 90% of the functions are masked out). Hence, for the rest of the section, we consider the workload for the *vehicle* database consisting only 110 (10% of 1100) ranking functions, and the *real estate* database comprising 150 ranking functions.

5.3 Quality Evaluation

Query Similarity: Based on the two proposed models of *query similarity* (Section 4.1.1 and 4.1.2), in the absence of a function $\mathcal{F}_{ij}$ for a user-query pair (U_i, Q_j), the most similar query (Q_c and Q_r using the query-condition and the query-result model respectively) asked by U_i, for which a function ($\mathcal{F}_{ic}$ and $\mathcal{F}_{ir}$ respectively) exists in the workload, is selected and the corresponding function is used to rank Q_j's results.

We test the quality of both query similarity models as follows: We rank Q_j's results (N_j) using $\mathcal{F}_{ic}$ and $\mathcal{F}_{ir}$ respectively, and obtain two sets of ranked results (R' and R''). We then use the original (masked) function $\mathcal{F}_{ij}$ to rank N_j and obtain the set (R). Since R represents the true ranking order provided by U_i for Q_j, we determine the quality of this model by computing the *Spearman rank correlation coefficient* (Equation 8) between R and R', and between R and R''. If the coefficients obtained are high (nearing 1.0), it validates our hypothesis (that for similar queries, the same user displays similar ranking preferences). Furthermore, if the coefficient between R and R' is greater than the one between R and R'', our understanding that *query-condition* model performs better than the *query-result* model is validated.

We performed the above process for each user asking every query. Figure 4 shows, for both databases, the *average query-condition* similarity (as well as the *average query-result* similarity) obtained across every query. The horizontal axis represents the queries; whereas the vertical axis represents the average value of the resulting *Spearman coefficient*. As the graph shows, over both the domains, the *query-condition* model outperforms the *query-result* model. The graphs indicate that the comparative loss of quality (highest value of Spearman coefficient being 0.95 for query 5) is due to the restricted number of queries in the workload. Although finding a similar query (for which a ranking function is available) for a workload comprising of 20 queries and only 10% of ranking functions is difficult, the results are very encouraging. Based on the results of these experiments, we believe that the *query similarity* model would perform at an acceptable level of quality even for large, sparse workloads.

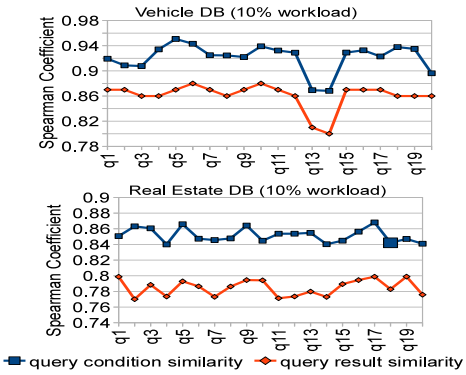

Fig. 4. Ranking Quality of Query Similarity Models

We further tested this model for comparing the quality produced by applying an *aggregate* function (i.e., selecting the top-K similar queries and combining their respective functions) instead of using the function of the most similar query. We varied the values of K from 2, 4, 5 and 10. For a value of $K = 5$, the *query-condition-similarity* model produced an overall average value of 0.86 for the Spearman coefficient (versus the 0.91 obtained if a single function of the most similar query is used) for the Vehicle database. Similarly, a value of 0.83 was obtained for the *query-result-similarity* model (versus the 0.86 obtained for a single function). A similar trend was observed for the Real Estate database as well. In the interest of space, we do not show the graphs for these results. The reader is directed to [29] for the complete set of detailed results of these experiments.

User Similarity: We validate the *user similarity* model as follows: Using the original function given by a user U_i for Q_j, we obtain a ranked set of results R. Then we determine, for – *query-independent*, *clustered*, and the three *top-K* models, the corresponding user U_l most similar to U_i having function $\mathcal{F}_{lj}$ in the workload. Using the corresponding function for each case, we get the ranked set of results and the Spearman coefficient between this set and R. In our experiments, we set a value of $K = 5$ for the *K-means* algorithm. A smaller value of $K = 2$ was chosen for the *top-K* models as our workload (in the number of queries) is small.

Figure 5 shows the average ranking quality individually achieved for the *vehicle* as well as *real estate* database, across all queries for all users taking the survey. Our results clearly

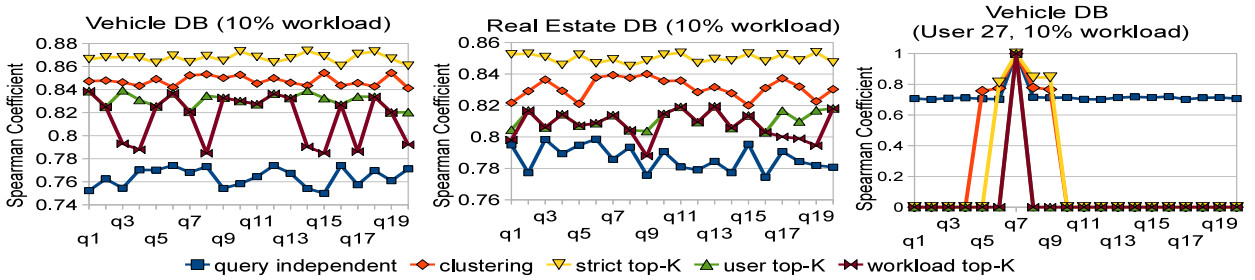

Fig. 5. Ranking Quality of User Similarity Models

show that the *strict top-K* model performs consistently better than the rest. However, as Figure 5 shows, the *strict top-K* (as well as the *clustered* and *user top-K*) fail to find functions for several queries (shown for a randomly selected user U_{27}). Figure 6 further confirms this fact by comparing the different models, in terms of their ability to determine ranking functions, across the entire workload. In spite of this, the accuracy of the *strict top-K model* is superior to all other models when a ranking function can be identified. Only when a ranking function cannot be found, using *strict top-K* does not make sense. We have proposed a suit of models (such as *clustered*, *user top-k*, *workload-based top-k* as well as *query-independent*) precisely for this reason and together will cover all the scenarios encountered in a sparse workload to provide a meaningful ranking function.

Similar to the *Query Similarity* model, using an aggregate function (i.e., derived by combining functions of top-K most similar users) did not provide any improvement in the quality of ranking than the one achieved by using a single function of the most similar user. Given the lack of space, we do not provide the details of these results (can be found in [29]).

Combined Similarity: We evaluated the quality of the *combined similarity model* (Algorithm 1). Figure 7 show the average quality of the combined model for both databases. We observe that the composite model performs better than the individual models as the sparseness of the workload decreases (i.e., the number of ranking functions increases). This matches the intuition that with more ranking functions in the workload, one is likely to find both individual and composite function with better similarity value.

For instance, in the vehicle database, the composite model achieved an average Spearman coefficient (across all users asking all queries) of 0.89 versus the 0.85 achieved by the user similarity model and 0.82 achieved by the query similarity model when 90% of the functions were masked (i.e., out of the 5000 results for the query, the combined similarity model correctly ranked 4450 tuples versus the

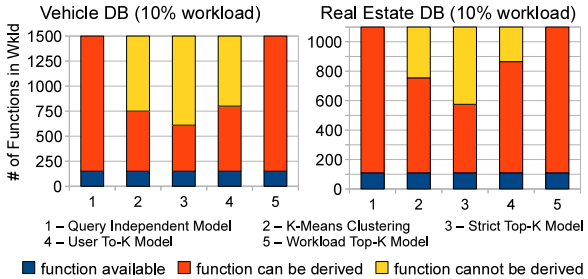

Fig. 6. User Similarity: Ranking Functions Derived

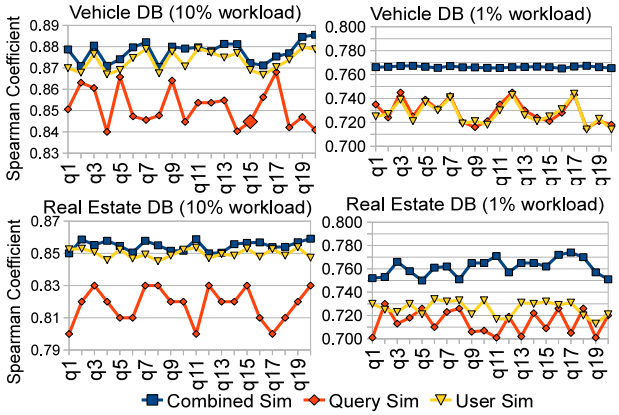

Fig. 7. Ranking Quality of Combined Similarity Model

user and query similarity model that correctly ranked 4250 and 41000 tuples respectively). However, when 99% of the functions were masked, the composite model achieved a 0.77 Spearman coefficient versus the 0.72 and 0.71 achieved by the user and query similarity models respectively (i.e., the combined model correctly ranked 3850 tuples whereas the user and query similarity models correctly ranked 3600 and 3550 tuples respectively). A similar trend is seen for the real estate database as well. Furthermore, user similarity shows a better quality than the query similarity. Since the averages are presented, one reason for the user similarity to be better is that in most cases a similar user is found due to the larger number of users than queries in our experimental setting.

5.4 Efficiency Evaluation

The goal of this study was to determine whether our framework can be incorporated into a real-world application. We generated a workload comprising of 1 million queries and 1 million users, and randomly masked out ranking functions such that only 0.001% of the workload was filled. We then generated 20 additional queries and selected 50 random users from the workload. We measure the efficiency of our system in terms of the average time, taken across all users, to perform ranking over the results of these queries (using Algorithm 1).

If we use main memory for storing the workload and not use any pre-computation and indexing for retrieval, determining similarities (STEPS ONE & TWO) are computational bottlenecks as compared to the latter. In order to reduce the time for estimating query similarities, we can pre-compute pairwise similarities between all values of every attribute in the schema. Furthermore, in order to reduce the time to lookup every query in the workload and then evaluate its similarity with the input query, we use a value-based hashing technique [11] to store

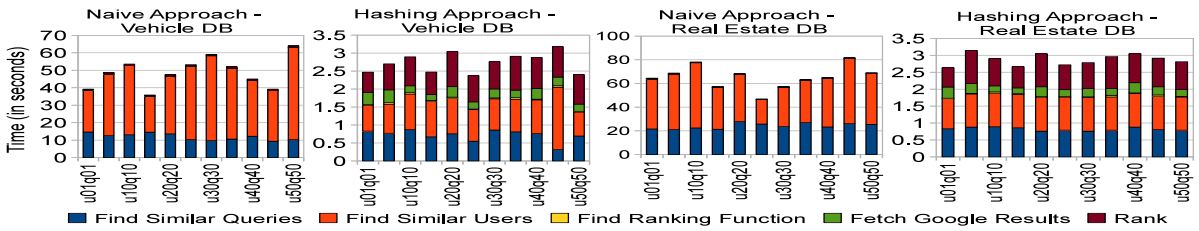

Fig. 8. Performance Comparison: Naive vs Hash-based Precomputation

all the queries in the workload. Likewise, all users are stored using a similar technique where the values corresponding to a user refer to various properties of the user profile.

Figure 8 show the performance of our framework using naive (where all steps are done in main memory at query time) and pre-computed/hash-based approaches (where Steps One and Two are pre-computed and the values are stored using hashing) for both databases. As is expected the naive approach is an order of magnitude slower than the other approach. For the vehicle database, Google Base provides unranked results in approximately 1.5 seconds. Our hash-based pre-computated system takes an average of 2.84 seconds (including Google Base response time) to rank and return the results. Our system adds, on the average, 1.3 seconds to return results using query- and user-dependent ranking. Likewise, for the *real estate* database, our system takes an average of 2.91 seconds (as compared to 2 seconds for Google Base) to show a ranked set of results adding less than a second for the improved ranking.

Although our system adds approximately a second over Google to display the results, the user is getting a customized, ranked result as compared to a "one size fits all" ranking. We believe that if, the ranking produced is desirable to the users, the extra overhead added would still be tolerable as compared to displaying unranked results. Finally, as we indicated earlier, the performance can be improved further.

6 Workload of Ranking Functions

Our proposed framework uses a workload of ranking functions derived across several user-query pairs. Since a ranking function symbolizes a users' specific preferences towards individual query results, obtaining such a function (especially for queries returning large number of results) is not a trivial task in the context of Web databases. Furthermore, since obtaining ranking functions from users on the Web is difficult (given the effort and time the user needs to spend in providing his/her preferences) determining the exact set of ranking functions to be derived for establishing the workload is important.

6.1 Alternatives for Obtaining a Ranking Function

Given a user U, a query Q and its results N, the ranking function ($\mathcal{F}_{\mathcal{UQ}}$) corresponds to the preference associated to each tuple (in N) by U. A straightforward alternative for deriving $\mathcal{F}_{\mathcal{UQ}}$ would be to ask U to manually specify the preferences as part of Q; a technique adapted in relational databases [30] [21] [23] [33]. However, Web users are typically unaware of the query language, the data model as well as the working of a ranking mechanism; thus, rendering this solution unsuitable for Web databases.

Since Web applications allow users to select results of their choice by an interaction (clicking, cursor placement, etc.) with the Web page, a solution to obtain $\mathcal{F}_{\mathcal{UQ}}$ would be to analyze U's interaction over N; an approach similar to *relevance feedback* [16] in document retrieval systems. However, mandatorily asking U to iteratively mark a set of tuples as relevant (and non-relevant) i.e., obtaining *explicit* feedback, may be difficult since Web users are typically reluctant to indulge in a lengthy interactions. Although providing various incentives to users (as done by routine surveys conducted by portals such as Yahoo, Amazon, etc.) is possible, an explicit feedback approach appears impractical in Web databases' context.

In contrast, since most users voluntarily choose a few tuples for further investigation (again by clicking, cursor placement, etc.), applying *implicit* feedback (where these selected tuples can be analyzed without U's knowledge) to obtain $\mathcal{F}_{\mathcal{UQ}}$ seems a more practical alternative. However, determining the number of tuples to be selected for generating an accurate function is difficult. This problem is further compounded by the fact that most users typically select very few tuples from the displayed results. An alternative to this problem would be to display an appropriately chosen sample, instead of the entire set of tuples, and determine $\mathcal{F}_{\mathcal{UQ}}$ based on the tuples selected from this sample (and extrapolate it over the entire set of results). Although determining an appropriate sample and validating the corresponding function obtained is difficult, we believe that there is scope to further investigate this problem and derive relevant solutions. In this paper, we present a preliminary solution, in the form of a learning model, to obtain the requisite function $\mathcal{F}_{\mathcal{UQ}}$ using *implicit* techniques.

6.1.1 Learning Model For Deducing a Ranking Function

For a query Q given by user U, we have at our disposal the set of query results N generated by the database. Let R ($\subset N$) be the set of tuples selected *implicitly* by U. For an attribute A_i, the relationship between its values in N and R can capture the significance (or weight) associated by U for A_i. As discussed in Section 3, our ranking function is a linear combination of attribute- and value-weights. We propose a learning technique called the *Probabilistic Data Distribution Difference* method for estimating these weights.

Learning Attribute-Weights: From *Example-1*, we know that U_1 is interested in 'red' colored 'Honda' vehicles. Let us focus on the preferences for the "Color" and "Mileage"attributes, and consider the individual probability distributions (shown in Figure 9) of their respective values in sets N and R. Since the set R will contain only 'red' colored vehicles, there is a significant difference between the distributions (for "Color") in R and N. In contrast, assuming that U_1 is not interested in

any particular mileage, the difference in the distributions for "Mileage" over sets R (which will contain cars with different mileages) and N is small.

Based on this observation, we can hypothesize that – for an attribute A_i, if the difference in the probability distributions between N and R is *large*, it indicates that A_i is important to the user, and hence, will be assigned a *higher* weight, and vice versa. The attribute weights are estimated formally, using the popular *Bhattacharya distance* (D_B) measure. If the probability distributions for the values of attribute A_i in sets N and R are N_p and R_p respectively, the *attribute-weight* (W_{A_i}) of A_i is given as:

$$W_{A_i} = D_B\left(N_p, R_p\right) = -ln\left(BC(N_p, R_p)\right) \tag{9}$$

where *BC* (Bhattacharya coefficient) for categorical and numerical attributes is given in Equations 10 and 11 as:

$$BC(N_p, R_p) = \sum_{x \in X} \sqrt{N_x, R_x} \tag{10}$$

$$BC(N_p, R_p) = \int \sqrt{N_x, R_x}\, dx \tag{11}$$

Determining Value-Weights: In order to rank the results of a query (using Equation 1), it is necessary that the score associated with every tuple is on a canonical scale. Although the attribute-scores estimated using the *Bhattacharya Distance* are between [0.0,1.0], the values in the tuples may have different types and ranges. Hence, we need to normalize them to obtain the necessary value-weights.

Since we have assumed that numerical attribute values are discretized using an appropriate scheme (Section 4.1.1), we normalize only categorical attributes (e.g., "Make", "Color", etc.) using a *frequency-based* approach. For the query Q by U, let a be the value of a categorical attribute A_i. The *value-weight* (a_{weight}) is given (by Equation 12) as the ratio of the frequency of a in R (a_r) with its frequency in N (a_n).

$$a_{weight} = \frac{a_r/|R|}{a_n/|N|} \tag{12}$$

6.1.2 Learning Model Evaluation

We test the quality of our proposed learning method (*Probabilistic Data Distribution Difference*) in deriving *attribute-weights* for a user query. In an ideal scenario, using the feedback (i.e., tuples selected) provided by an user over a query's results, a ranking function would be deduced using the learning model. The quality of this function, however, can only be evaluated the next time the same user asks the same query, and would be estimated in terms of the percentage of the top-k results generated (by this function) that match the user's preferences. In an experimental setting, asking a user to select tuples (from a large set) once, and then validate the function by asking the same query again would be difficult.

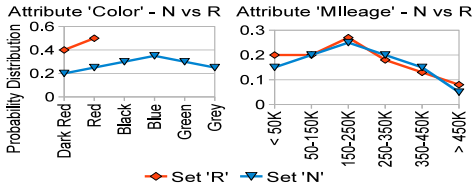

Fig. 9. Probability Distribution of sets N and R

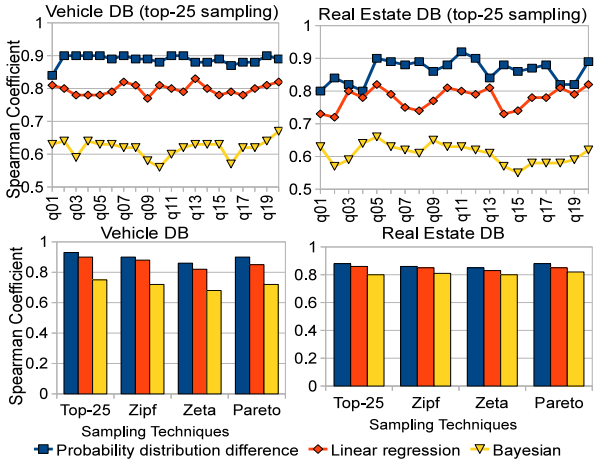

Fig. 10. Ranking Quality of Learning Models

Consequently, we test the quality of the proposed learning model as follows: From the workload in Section 5, we have at our disposal ranking functions provided by users over 20 distinct queries on two databases. Consider query Q_1 (with results N_1) for which a user U_1 has provided a ranking function ($\mathcal{F}_{11}$). Using this function, we rank N_1 to obtain N_{r1}, and select a set of top-K tuples (from this ranked set) as the set (R_1) chosen by U_1. Using our learning model, we derive a ranking function ($\mathcal{F}'_{11}$) comparing N_1 and R_1. We then use $\mathcal{F}'_{11}$ to rank Q_1's results and obtain N'_{r1}. The quality of the learning model is then estimated as the *Spearman rank correlation coefficient* (Equation 8) between N_{r1} and N'_{r1}. Higher the value of this coefficient, better is the quality of the ranking function (and the model), and vice-versa.

In our experiments, we chose a value of $K = 25$ i.e., we choose the *top-25* tuples generated by the user's original ranking function as the set R since a Web user, in general, selects a very small percentage of the top-K tuples shown to him/her. The model was also tested with R varying from 10 to 50 tuples and since, the results are similar for different values of R, we omit the details in the interest of space. We validate the ranking quality of our proposed learning model to the quality achieved by using two established and widely-used learning models – *linear regression* and *naive Bayesian classifier*. Figure 10 compares the average ranking quality achieved by the models in deriving ranking functions for each individual query for all the users in both databases.

In order to prove the generic effectiveness of our model, the R is now chosen using different sampling techniques (instead of top-25). It is natural that based on users' preferences, higher ranked tuples (from N_{r1}) should have a higher probability of being sampled than the lower ranked ones. Hence, the sampling technique we choose selects the required 25 tuples using following power-law distributions: *Zipf*, *Zeta*, and *Pareto*. Figure 10 compares the quality of our learning model (shown by vertical bar in the graph) with the other models across both databases using different sampling schemes. The results clearly indicates that our proposed method performs on par with the popular learning models. These results also validate our claim of *learning* using *feedback* as an alternative to obtaining ranking functions for generating workloads.

6.2 Challenges in Establishing a Workload

At the time of answering a user query, if a ranking function for this or a similar pair is not available, the Similarity model will be forced to use a ranking function corresponding to a user and/or query which may not be very similar to the input pair, thus, affecting the final quality of ranking achieved by the function. Therefore, in order to achieve an acceptable quality of ranking, the workload should be established in a way such that for any user asking a given query, there exists a ranking function corresponding to at least one similar pair.

Consequently, for the workload $\mathbf{W}$ comprising of M queries and N users, the goal is to determine a set $\mathcal{S}$ of user-query pairs such that, for each user-query pair in $\mathbf{W}$, there exists at least one user-query pair in $\mathcal{S}$ that occurs in the list of pairs most similar to the former. In order to determine such a set, two important questions must be answered – i) what is the minimum size of this set $\mathcal{S}$?, and ii) which user-query pairs (or cells) should be selected from $\mathbf{W}$ to represent this set $\mathcal{S}$?.

The first question is critical as obtaining ranking functions for Web database queries requires the user to spend a considerable amount of time and effort. Likewise, the second question is crucial since the quality of ranking obtained depends largely on the functions derived for an appropriate set of user-query pairs. We are currently working on addressing these challenges for generating an appropriate workload [28].

7 Conclusion

In this paper, we proposed a *user*- and *query*-dependent solution for ranking query results for Web databases. We formally defined the similarity models (*user*, *query* and *combined*) and presented experimental results over two Web databases to corroborate our analysis. We demonstrated the practicality of our implementation for real-life databases. Further, we discussed the problem of establishing a workload, and presented a learning method for inferring individual ranking functions.

Our work brings forth several additional challenges. In the context of Web databases, an important challenge is the design and maintenance of an appropriate workload that satisfies properties of similarity-based ranking. Determining techniques for inferring ranking functions over Web databases is an interesting challenge as well. Another interesting problem would be to combine the notion of user similarity proposed in our work with existing user profiles to analyze if ranking quality can be improved further. Accommodating range queries, usage of functional dependencies and attribute correlations needs to be examined. Applicability of this model for other domains and applications also needs to be explored.

References

[1] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[2] S. Amer-Yahia, A. Galland, J. Stoyanovich, and C. Yu. From del.icio.us to x.qui.site: recommendations in social tagging sites. In *SIGMOD Conference*, pages 1323–1326, 2008.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

[4] M. Balabanovic and Y. Shoham. Content-based collaborative recommendation. *ACM Communications*, 40(3):66–72, 1997.

[5] J. Basilico and T. Hofmann. A joint framework for collaborative and content filtering. In *SIGIR*, pages 550–551, 2004.

[6] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.

[7] M. K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.

[8] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Intnl. Conf. on Machine Learning (ICML)*, pages 46–54, 1998.

[9] K. C.-C. Chang, B. He, C. Li, M. Patil, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, 2004.

[10] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[11] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 31(3):1134–1168, 2006.

[12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *International conference on World Wide Web (WWW)*, pages 613–622, New York, NY, USA, 2001. ACM.

[13] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *VLDB*, pages 696–707, 1990.

[14] S. Gauch and M. S. et. al. User profiles for personalized information access. In *Adaptive Web*, pages 54–89, 2007.

[15] Google. Google base. http://www.google.com/base.

[16] B. He. Relevance feedback. In *Encyclopedia of Database Systems*, pages 2378–2379, 2009.

[17] T. Hofmann. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *SIGIR*, pages 259–266, 2003.

[18] S.-W. Hwang. *Supporting Ranking For Data Retrieval*. PhD thesis, University of Illinois, Urbana Champaign, 2005.

[19] T. Kanungo and D. Mount. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions of Pattern Analysis in Machine Intelligence*, 24(7):881–892, 2002.

[20] G. Koutrika and Y. E. Ioannidis. Constrained optimalities in query personalization. In *SIGMOD Conference*, pages 73–84, 2005.

[21] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, pages 131–142, 2005.

[22] X. Luo, X. Wei, and J. Zhang. Guided game-based learning using fuzzy cognitive maps. *IEEE Transactions on Learning Technologies*, PP(99):1–1, 2010.

[23] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions of Database Systems*, 29(2):319–362, 2004.

[24] A. Penev and R. K. Wong. Finding similar pages in a social tagging repository. In *WWW*, pages 1091–1092, 2008.

[25] Y. Rui, T. S. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars. In *IEEE Internationa Conference on Image Processing*, pages 815–818, 1997.

[26] X. Shi and C. C. Yang. Mining related queries from web search engine query logs using an improved association rule mining model. *J. Am. Soc. Inf. Sci. Technol.*, 58:1871–1883, October 2007.

[27] W. Su, J. Wang, Q. Huang, and F. Lochovsky. Query result ranking over e-commerce web databases. In *Conference on Information and Knowledge Management (CIKM)*, pages 575–584, 2006.

[28] A. Telang, S. Chakravarthy, and C. Li. Establishing a workload for ranking in web databases. Technical report, UT Arlington, http://cse.uta.edu/research/Publications/Downloads/CSE-2010-3.pdf, 2010.

[29] A. Telang, C. Li, and S. Chakravarthy. One size does not fit all: Towards user- and query-dependent ranking for web databases. Technical report, UT Arlington, http://cse.uta.edu/research/Publications/Downloads/CSE-2009-6.pdf, 2009.

[30] K. Werner. Foundations of preferences in database systems. In *VLDB*, pages 311–322. VLDB Endowment, 2002.

[31] L. Wu and C. F. et. al. Falcon: Feedback adaptive loop for content-based retrieval. In *VLDB*, pages 297–306, 2000.

[32] Z. Xu, X. Luo, and W. Lu. Association link network: An incremental semantic data model on organizing web resources. In *Intl. Conf. on Parallel and Distributed Systems (ICPADS)*, pages 793–798, 2010.

[33] H. Yu, S.-w. Hwang, and K. C.-C. Chang. Enabling soft queries for data retrieval. *Information Systems*, 32(4):560–574, 2007.

[34] H. Yu, Y. Kim, and S. won Hwang. Rv-svm: An efficient method for learning ranking svm. In *PAKDD*, pages 426–438, 2009.

[35] T. C. Zhou, H. Ma, M. R. Lyu, and I. King. Userrec: A user recommendation framework in social tagging systems. In *AAAI*, 2010.

Aditya Telang Aditya Telang is a PhD Candidate in the Department of Computer Science and Engineering at the University of Texas at Arlington. His research interests are ranking, querying, data mining and information integration over Web databases. He received his Masters degree in Computer Science from the State University of New York (SUNY) at Buffalo in 2004. He received his Bachelors degree in Computer Science from Fr. Conceicao Rodrigues College of Engineering, Mumbai University, India.

Chengkai Li Chengkai Li is an Assistant Professor in the Department of Computer Science and Engineering at the University of Texas at Arlington. His research interests are in the areas of databases, Web data management, data mining, and information retrieval. He works on ranking and top-k queries, Web search/mining/integration, database exploration, query processing and optimization, social networks and user-generated content, OLAP and data warehousing. His current focus is on both bringing novel retrieval and mining facilities into database systems and providing query functionality on top of Web data and information. He received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign, and a M.E. and a B.S. degree in Computer Science from Nanjing University.

Sharma Chakravarthy Sharma Chakravarthy is Professor of Computer Science and Engineering Department at The University of Texas at Arlington (UTA). He is the founder of the Information Technology laboratory (IT Lab) at UTA. His research has spanned semantic and multiple query optimization, complex event processing, and web databases. He is the co-author of the book: Stream Data Processing: A Quality of Service Perspective (2009). His current research includes information integration, web databases, recommendation systems, integration of stream and complex event processing, and knowledge discovery. He has published over 150 papers in refereed international journals and conference proceedings.