Query Result Ranking over E-commerce Web Databases

Weifeng Su
Dept. of Computer Science & Engineering
HKUST
Hong Kong
weifeng@cse.ust.hk

Jiying Wang
Computer Science Dept.
City University
Hong Kong
wangjy@cityu.edu.hk

Qiong Huang
Dept. of Computer Science & Engineering
HKUST
Hong Kong
bestivy@cse.ust.hk

Fred Lochovsky
Dept. of Computer Science & Engineering
HKUST
Hong Kong
fred@cse.ust.hk

ABSTRACT
To deal with the problem of too many results returned from an E-commerce Web database in response to a user query, this paper proposes a novel approach to rank the query results. Based on the user query, we speculate how much the user cares about each attribute and assign a corresponding weight to it. Then, for each tuple in the query result, each attribute value is assigned a score according to its "desirableness" to the user. These attribute value scores are combined according to the attribute weights to get a final ranking score for each tuple. Tuples with the top ranking scores are presented to the user first. Our ranking method is domain independent and requires no user feedback. Experimental results demonstrate that this ranking method can effectively capture a user's preferences.

Categories and Subject Descriptors
H.3.5 [**INFORMATION STORAGE AND RETRIEVAL**]: Online Information Services - *Web-based services*

General Terms
Algorithms, Design, Experimentation, Human Factors.

Keywords
E-commerce, Query result ranking, Attribute weight assignment

1. INTRODUCTION
With the rapid expansion of the World Wide Web, more and more Web databases are available. At the same time, the size of existing Web databases is growing rapidly. One common problem faced by Web users is that there is usually *too many query results* returned for a submitted query. For example, when a user submits a query to autos.yahoo.com to search for a used car within 50 miles of New York with a price between $5,000 and $10,000, 10,483 records are returned. In order to find "the best deal", the user has to go through this long list and compare the cars to each other, which is a tedious and time-consuming task.

Most Web databases rank their query results in ascending or descending order according to a single attribute (e.g., sorted by date, sorted by price, etc.). However, many users probably consider multiple attributes simultaneously when judging the relevance or desirableness of a result. While some extensions to SQL allow the user to specify attribute weights according to their importance to him/her [21], [26], this approach is cumbersome and most likely hard to do for most users since they have no clear idea how to set appropriate weights for different attributes. Furthermore, the user-setting-weight approach is not applicable for categorical attributes.

In this paper, we tackle the many-query-result problem for Web databases by proposing an automatic ranking method, QRRE (*Query Result Ranking for E-commerce*), which can rank the query results from an E-commerce Web database without any user feedback. We focus specifically on E-commerce Web databases because they comprise a large part of today's online databases. In addition, most E-commerce customers are ordinary users who may not know how to precisely express their interests by formulating database queries. The carDB Web database is used in the following examples to illustrate the intuitions on which QRRE is based.

Example 1: *Consider a used Car-selling Web database D with a single table carDB in which the car instances are stored as tuples with attributes: Make, Model, Year, Price, Mileage and Location. Each tuple t_i in D represents a used car for sale.*

Given a tuple t_i in the query result T_q for a query q that is submitted by a buyer, we assign a ranking score to t_i, based on its attribute values, which indicates its desirableness, from an E-commerce viewpoint, to the buyer. For instance, it is obvious that a luxury, new and cheap car is globally popular and desired in the used car market. However, sometimes the desired attribute values conflict with each other. For example, a new luxury car with low mileage is unlikely to be cheap. Hence, we need to decide which attributes are more important for a buyer. Some buyer may care more about the model of a car, while some other buyer may be more concerned about its price. For each attribute, we use a weight to denote its importance to the user.

In this work, we assume that the attributes about which a user cares most are present in the query he/she submits, from which the attribute importance can be inferred. We define *specified attributes* to be attributes that are specified in a query and *unspecified attributes* to be attributes that are not specified in a query. Furthermore, we also consider that a subset of the unspecified attributes, namely, those attributes that are closely correlated to the query, is also important.

Example 2: *Given a query with condition "Year > 2005", the query condition suggests that the user wants a relatively new car. Intuitively, besides the Year attribute, the user is more concerned about the Mileage than he/she is concerned about the Make and Location, considering that a relatively new car usually has low mileage.*

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

Given an unspecified attribute A_i, the correlation between A_i and the user query q is evaluated by the difference between the distribution of A_i's values over the query results and their distribution over the whole database D. The bigger the difference, the more A_i correlates to the specified attribute value(s). Consequently, we assign a bigger attribute weight to A_i. Example 3 explains our intuition.

Example 3: *Suppose a used car database D contains car instances for which the Year has values 1975 and onwards and D returns a subset d containing the tuples that satisfy the query with condition "Year > 2005". Intuitively, Mileage values of the tuples in d distribute in a small and dense range with a relatively low average, while the Mileage values of tuples in D distribute in a large range with a relatively high average. The distribution difference shows a close correlation between the unspecified attribute, namely, Mileage, and the query "Year > 2005".*

Besides the attribute weight, we also assign a preference score to each attribute value, including the values of both specified and unspecified attributes. In the E-commerce context, we first assume that an expensive product is less preferred than a cheap product if other attribute values are equal. Hence, we assign a small preference score for a high Price value and a large preference score for a low Price value. We further assume that a non-Price attribute value with high desirableness, such as a luxury car or a new car, correlates positively with a high Price value. Thus, a luxury car is more expensive than a standard car and a new car is usually more expensive than an old car. Based on this assumption, we convert a value a_i of a non-Price attribute A_i to a Price value p'_1 where p'_1 is the average price of the product for $A_i = a_i$ in the database. Consequently, the preference score for a_i will be large if p'_1 is large because a large Price value denotes a high desirableness for the user. Finally, the attribute weight and the value preference score are combined to get the final ranking score for each tuple. The tuples with the largest ranking scores are presented to the user first.

The contributions of this paper include the following:

1. We present a novel approach to rank the tuples in the query results returned by E-commerce Web databases.
2. We propose a new attribute importance learning approach, which is domain independent and query adaptive.
3. We also propose a new attribute-value preference score assignment approach for E-commerce Web databases.

In the entire ranking process, no user feedback is required.

The rest of the paper is organized as follows. Section 2 reviews some related work. Section 3 gives a formal definition of the many-query-result problem and presents an overview of QRRE. Section 4 proposes our attribute importance learning approach while Section 5 presents our attribute preference score assignment approach. Experimental results are reported in Section 6. The paper is concluded in Section 7.

2. RELATED WORK

Query result ranking has been investigated in information retrieval for a long time. Cosine Similarity with TF-IDF weighting of the vector space model [2] and [26], the probabilistic ranking model [30] and [31] and the statistical language model [12] have been successfully used for ranking purposes. In addition, [10], [11], [14] and [15] explore the integration of database and information retrieval techniques to rank tuples with text attributes. [1], [5] and [17] propose some keyword-query based retrieval techniques for databases. However, most of these techniques focus on text attributes and it is very difficult to apply these techniques to rank tuples with categorical or numerical attributes.

Some recent research addresses the problem of relational query result ranking. In [9], [26], [28] and [33], user relevance feedback is employed to learn the similarity between a result record and the query, which is used to rank the query results in relational multimedia databases. In [21] and [26], the SQL query language is extended to allow the user to specify the ranking function according to their preference for the attributes. In [18] and [19], users are required to build profiles so that the query result is ranked according to their profile. Compared with the above work, our approach is fully automatic and does not require user feedback or other human involvement.

In [1] and [12], two ranking methods have been proposed that take advantage of the links (i.e., associations) among records, such as the citation information between papers. Unfortunately, linking information among records does not exist for most domains.

The work that is most similar to ours is the probabilistic information retrieval (PIR) model in [8], which addresses the many-query-result problem in a probabilistic framework. In PIR, the ranking score is composed of two factors: global score, which captures the global importance of unspecified values, and conditional score, which captures the strength of the dependencies between specified and unspecified attribute values. The two scores are combined using a probabilistic approach. Our approach differs from that in [8] in the following aspects:

1. PIR only focuses on point queries, such as "$A_i = a_i$". Hence, both a query with condition "Mileage < 5000" and a query with condition "Mileage < 2500" may have to be converted to a query with condition "Mileage = small" to be a valid query in PIR, which is not reasonable for many cases. In contrast, QRRE can handle both point and range queries.
2. PIR focuses on the unspecified attributes during query result ranking while QRRE deals with both specified and unspecified attributes. For example, suppose a car with price less than $10,000 is labeled as a "cheap" car. For a query "Price < 10000", PIR will only consider the value difference for non-Price attributes among tuples and ignore the price difference, which is usually important for a poor buyer. On the contrary, QRRE will consider the value difference for all attributes.
3. A workload containing past user queries is required by PIR in order to learn the dependency between the specified and unspecified attribute values, which is unavailable for new online databases, while QRRE does not require such a workload.

The experimental results in Section 6 show that QRRE produces a better quality ranking than does PIR.

The attribute-importance learning problem was studied in [23] and [24], in which attribute importance is learned according to the attribute dependencies. In [23], a Bayesian network is built to discover the dependencies among attributes. The root attribute is the most important while the leaf attributes are less important.

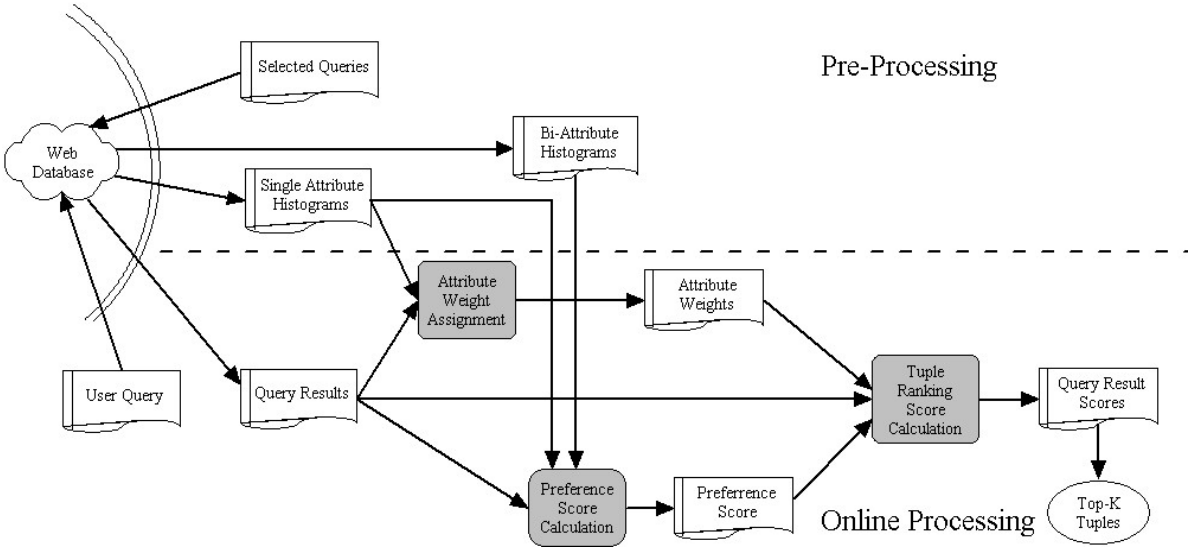

Figure 1: Architecture of a system employing Query Result Ranking for E-commerce (QRRE).

In [24], an attribute dependency graph is built to discover the attribute dependencies. Both of these methods learn the attribute importance based on some pre-extracted data and their result is invariant to the user queries. Furthermore, both methods can only determine the attribute importance sequence. They are incapable of giving a specific value to show how important each attribute is. In contrast, the attribute-importance learning method presented in this paper can be adapted to the user's query and thus can be tailored to take into account the desire of different users, since each attribute is assigned a weight that denotes its importance for the user. To our knowledge, this is the first work that generates attribute weights that are adaptive to the query the user submitted.

3. QUERY RESULT RANKING

In this section, we first define the many-query-result problem and then present an overview of QRRE.

3.1 Problem Formulation

Consider an autonomous Web database D with attributes $A=\{A_1, A_2, \ldots, A_m\}$ and a selection query q over D with a conjunctive selection condition that may include point queries, such as "$A_i = a_i$", or range queries, such as "$a_{i1} < A_i < a_{i2}$". Let $T=\{t_1, t_2, \ldots, t_n\}$ be the set of result tuples returned by D for the query q. In many cases, if q is not a selective query, it will produce a large number of query results (i.e., a large T). The goal is to develop a ranking function to rank the tuples in T that captures the user's preference, is domain-independent and does not require any user feedback.

3.2 QRRE

Initially, we focus on E-commerce Web databases because E-commerce Web databases comprise a large proportion of the databases on the Web. We further assume that each E-commerce Web database has a Price attribute, which we always assume to be A_1. The Price attribute A_1 plays an intermediate role for all attributes during the attribute preference score assignment.

Example 4: *Consider the tuples in Table 1 that represent an example query result set T. It can be seen that most tuples have their own advantages when compared with other tuples. For example, t_1 is a relatively new car while t_2 is a luxury car and t_3 is the cheapest among all cars. Hence, depending on a user's preferences, different rankings may be needed for different users. Assuming that a user would prefer to pay the smallest amount for a car and that all other attribute values are equal, then the only certainty is that t_4 should always be ranked after t_3 because its mileage is higher than t_3 while it is more expensive than t_3.*

Table 1. Examples of used car tuples.

	Year	Make	Model	Mileage	Price	Location
t_1	2005	Toyota	Corolla	16995	26700	Seattle
t_2	2002	Mercedes-Benz	G500	47900	39825	Seattle
t_3	2002	Nissan	350Z	26850	17448	Seattle
t_4	2002	Nissan	350Z	26985	18128	Seattle

According to *Example 4*, two problems need to be solved when we assign a ranking score for a tuple $t_i=\{t_{i1}, t_{i2}, \ldots, t_{im}\}$ in the query result T:

1. How can we surmise how much a user cares about an attribute A_j and how should we assign a suitable weight w_j for the attribute(s) A_j to reflect its (their) importance to the user?
2. How do we assign a preference score v_{ij} for an attribute value t_{ij}?

For example, when assigning the score for the attribute value "Year = 2005" in t_1, should the score be larger than the score assigned for attribute value "Year = 2002" in t_2 and how much larger is reasonable? The first problem will be discussed in Section 4. The second problem will be discussed in Section 5.

Having assigned a preference score v_{ij} ($1 \le j \le m$) to each attribute-value of t_i and a weight w_j to the attribute A_j, the value preference scores v_{ij} are summed to obtain the ranking score s_i for t_i to reflect the attribute importance for the user. That is:

$$s_i = \sum_{j=1}^{m} w_j v_{ij}$$

The overall architecture of a system employing QRRE is shown in Figure 1. Such a system includes two components: pre-processing component and online processing component. The pre-processing component collects statistics about the Web database D using a set

of selected queries. Two kinds of histograms are built in the pre-processing step: single-attribute histograms and bi-attribute histograms. A single-attribute histogram is built for each attribute A_j. A bi-attribute histogram is built for each non-Price attribute (i.e., A_j in which i>1) using the Price attribute A_1.

The online-processing component ranks the query results given the user query q. After getting the query results T from the Web database D for q, a weight is assigned for each attribute by comparing its data distribution in D and in the query results T. At the same time, the preference score for each attribute value in the query result is determined using the information from the bi-attribute histograms. The attribute weights and preference scores are combined to calculate the ranking score for each tuple in the query result. The tuples' ranking scores are sorted and the top K tuples with the largest ranking scores are presented to the user first.

4. ATTRIBUTE WEIGHT ASSIGNMENT

In the real world, different users have different preferences. Some people prefer luxury cars while some people care more about price than anything else. Hence, we need to surmise the user's preference when we make recommendations to the user as shown by Example 4 in Section 3. The difficulty of this problem lies in trying to determine what a user's preference is (i.e., which attributes are more important) when no user feedback is provided. To address this problem, we start from the query the user submitted. We assume that the user's preference is reflected in the submitted query and, hence, we use the query as a hint for assigning weights to attributes. The following example provides the intuition for our attribute weight assignment approach.

Example 5: *Consider the query q with condition "Year > 2005", which denotes that the user prefers a relatively new car. It is obvious that the specified attribute Year is important for the user. However, all the tuples in the query result T satisfy the query condition. Hence, we need to look beyond the specified attribute and speculate further about what the user's preferences may be from the specified attribute. Since the user is interested in cars that are made after 2005, we may speculate that the user cares about the Mileage of the car. Considering the distribution of Mileage values in the database, cars whose model year is greater than 2005 usually have a lower mileage when compared to all other cars. In contrast, attribute Location is less important for the user and its distribution in cars whose model year is greater than 2005 may be similar to the distribution in the entire database.*

According to this intuition, an attribute A_j that correlates closely with the query will be assigned a large weight and vice verse. Furthermore, as Example 3 in Section 1 shows, the correlation of A_j and the query can be measured by the data distribution difference of A_j in D and in T.

It should be noted that the specified attribute is not always important, especially when the condition for the specified attribute is not selective. For example, for a query with condition "Year > 1995 and Make = BMW", the specified attribute Year is not important because almost all tuples in the database satisfy the condition "Year > 1995" and the Year distribution in D and in T is similar.

A natural measure of the distribution difference of A_j in D and in T is the Kullback-Leibler distance or Kullback-Leibler (KL) divergence [13]. Suppose that A_j is a categorical attribute with value set $\{a_{j1}, a_{j2}, \ldots, a_{jk}\}$. Then the KL-divergence of A_j from D to T is:

$$D_{KL}(D \| T) = \sum_{l=1}^{k} prob(A_j = a_{jl} \mid D) \log \frac{prob(A_j = a_{jl} \mid D)}{prob(A_j = a_{jl} \mid T)} \quad (1)$$

in which $prob(A_j=a_{jl} \mid D)$ refers to the probability that $A_j = a_{jl}$ in D and $prob(A_j=a_{jl} \mid T)$ refers to the probability that $A_j = a_{jl}$ in T. If A_j is a numerical attribute, its value range is first discretized into a few value sets, where each set refers to a category, and then the KL-divergence of A_j is calculated as in (1).

4.1 Histogram Construction

To calculate the KL-divergence in equation (1) we need to obtain the distribution of attribute values over D. The difficulty here is that we are dealing with an autonomous database and we do not have full access to all the data. In [24], the attribute value distribution over a collection of data crawled from D is used to estimate the actual attribute value distribution over D. However, it is highly likely that the distribution of the crawled data can be different from that of D because the crawled data may be biased to the submitted queries.

In this paper, we propose a probing-and-count based method to build a histogram for an attribute over a Web database[1]. We assume that the number of query results is available in D for a given query. After submitting a set of selected queries to D, we can extract the number of query results, instead of the actual query results, to get the attribute value distribution of A_i. An equi-depth histogram [27] is used to represent the attribute value distribution, from which we will get the probability required in Equation (1). The key problem in our histogram construction for A_i is how to generate a set of suitable queries to probe D.

Figure 2 shows the algorithm for building a histogram for attribute A_i. For each attribute A_i, a histogram is built in the preprocessing stage. We assume that one attribute value of A_i is enough to be a query for D. If A_i is a categorical attribute, each category of A_i is used as a query to get its occurrence count (Lines 2-3). If A_i is a numerical attribute, an equal-depth histogram is built for A_i. We first decide the occurrence frequency threshold t for each bucket by dividing $|D|$, namely, the number of tuples in D, with the minimum bucket number n that will be created for a numerical attribute A_i. In our experiments, n is empirically set to be 20. Then we probe D using a query with condition on A_i such that low$\leq A_i <$up and get c, the number of instances in that range (Line 8). If c is smaller than t, a bucket is added for it in H_{Di} (Line 10) and another query probe is prepared (Line 11). Otherwise, we update the query probe condition on A_i by reducing the size of the bucket (Line 13) and a new iteration begins. The iteration continues until each value in the value range is in a bucket. It is obvious that there are some improvements that can be made to the algorithm to accelerate the histogram construction. The improvements are not described here because histogram construction is not the major focus of this paper. Considering that only a single-attribute histogram is constructed, the process should complete quickly.

[1] Although both our histogram construction method and the histogram construction methods in [1] and [5] are probing-based, they have different goals. The goal in [1] and [5] is to build a histogram that precisely describes the regions on which the queries concentrate, while our purpose is to build a histogram that summarizes the data distribution of D as precisely as possible with a number of query probes.

Input: Attribute A_i and its value range

Web database D with the total number of tuples $|D|$

Minimum bucket number n

Output: A single-attribute histogram H_{Di} for A_i

Method:

1. If A_i is a categorical attribute
2. For each category a_{ij} of A_i, probe D using a query with condition "$A_i=a_{ij}$" and get its occurrence count c
3. Add a bucket (a_{ij}, c) into H_{Di}
4. If A_i is a numerical value attribute with value range [a_{low}, a_{up})
5. $t = |D| / n$
6. low = a_{low}, up = a_{up}
7. Do
8. probe D with a query with condition "low $\leq A_i <$up" and get its occurrence count c
9. if $c \leq t$
10. Add a bucket (*low, up*, c) into H_{Di}
11. low = up, up = a_{up}
12. else
13. up = low + (up - low) / 2
14. While low < a_{up}
15. Return H_{Di}

Figure 2: Probing-based histogram construction algorithm.

A histogram H_{Ti} also needs to be built for A_i over T (the result set) to get its probability distribution over T. For each bucket of H_{Di}, a bucket with the same bucket boundary is built in H_{Ti} while its frequency is counted in T.

4.2 Attribute Weight Setting

After getting the histogram of A_i over D and T, the histograms are converted to a probability distribution by dividing the frequency in each bucket of the histogram by the bucket frequency sum of the histogram. That is, the probability distribution of A_i for D, P_{Di}, is

$$p_{Di} = \frac{|c_{Dk}|}{|D|}$$

in which c_{Dk} is the frequency of the k^{th} bucket in H_{Di}. The probability distribution of A_i for T, P_{Ti}, is

$$p_{Ti} = \frac{|c_{Tk}|}{|T|}$$

in which c_{Tk} is the frequency of the k^{th} bucket in H_{Ti}.

Next, for the i^{th} attribute A_i, we assign its importance w_i as

$$w_i = \frac{KL(P_{Di}, P_{Ti})}{\sum_{j=1}^{m} KL(P_{Dj}, P_{Tj})}$$

The attribute weight assignment is performed not only on the unspecified attributes, but also on the specified attributes. If a specified attribute is a point condition, its attribute weight will be the same for all tuples in the query result. If a specified attribute is a range condition, its attribute weight will be different for the tuples in the query result. Example 6 illustrates this point.

Example 6: *Consider a query q with condition "Make = 2004 and Price < 10000". In q, since the specified attribute Make is a point attribute, the attribute weight assigned to it is useless because all the query results have the same value for Make. On the other hand, since the attribute Price is a range attribute, the price of different tuples is an important factor to consider during ranking.*

4.3 Examples of Attribute Weight Assignment

In our experiments, we found that the attribute weight assignment was intuitive and reasonable for the given queries. Table 2 shows the attribute weight assigned to different attributes corresponding to different queries in our experiments for the carDB. Given a query with condition "Mileage < 20000", which means that the user prefers a new car, as expected the attribute "Mileage" is assigned a large weight because it is a specified attribute and the attribute "Year" is assigned a large weight too. The attribute "Model" is assigned a large weight because a new car usually has a model that appears recently. In contrast, Consider the query with condition "Make = BMW & Mileage < 100000". The sub-condition "Mileage < 100000" possesses a very weak selective capability because almost all tuples in the database satisfy it. The buyer is actually just concerned about the Make and the Model of the car. As expected, the attribute Make and Model are assigned large weights, while Year and Mileage are no longer assigned large weights.

Table 2: Attribute weight assignments for two queries.

	Mileage < 20000	Make = BMW & Mileage < 100000
Year	0.222	0.015
Make	0.017	0.408
Model	0.181	0.408
Price	0.045	0.120
Mileage	0.533	0.04
Location	0.0003	0.002

5. ATTRIBUTE PREFERENCE SCORE ASSIGNMENT

In addition to the attributes themselves, different values of an attribute may have different attractions for the buyer. For example, a car with a low price is obviously more attractive than a more expensive one if other attribute values are the same. Similarly, a car with low mileage is also obviously more desirable. Given an attribute value, the goal of the attribute preference score assignment module is to assign a preference score to it that reflects its desirableness for the buyer. To facilitate the combination of scores of different attribute values, all scores assigned for different attribute values are in [0, 1].

Instead of requiring human involvement for attribute value assignment, given a normal E-commerce context, we make the following two intuitive assumptions:

1. *Price assumption:* A product with a lower price is always more desired by buyers than a product with a higher price if the other attributes of the two products have the same values. For example, if all other attribute values are the same, a cheaper car is preferred over a more expensive car.
2. *Non-Price assumption:* A non-Price attribute value with higher desirableness for the user corresponds to a higher price. For example, a new car, which most buyers prefer, is usually more

expensive than an old car. Likewise, a luxury car is usually more expensive than an ordinary car.

With the above two assumptions, we divide the attributes into two sets: Price attribute set, which only includes the attribute Price, and non-Price attribute set, which includes all attributes except Price. The two sets of attributes are handled in different ways.

According to the Price assumption, we assign a large score for a low price and a small score for a high price. To avoid requiring human involvement to assign a suitable score for a Price value, the Price distribution in D is used to assign the scores. Given a Price value t, a score v_t is assigned to it as the percentage of tuples whose Price value is bigger than t_i in D:

$$v_t = \frac{S_t}{|D|}$$

in which S_t denotes the number of tuples whose Price value is bigger than t. In our experiments, the histogram for the attribute Price A_1, whose construction method is described in Section 4, is used for the Price preference score assignment.

Figure 3 shows the algorithm used to assign a score v for a Price value t using the Price histogram. Given the Price histogram H_{D1}, the frequency sum is fist calculated (Line 1). Then we count the number S_t of tuples whose Price value is bigger than t. For each bucket in H_{D1}, if the lower boundary of the bucket is bigger than t, it means that all the tuples for this bucket have a Price value bigger than t and the frequency of this bucket is added to S_t (Line 4). If t is within the boundary of the bucket, we assume that the Price has a uniform distribution in the bucket and a fraction of the frequency in this bucket is added to S_t (Line 5). If the upper boundary of the bucket is smaller than t, it means that all the tuples for this bucket have a Price value lower than t and the bucket is ignored. Finally the ratio is generated by dividing S_t with the frequency sum.

For a value a_i of a non-Price attribute A_i, the difficulty of assigning it a score is two fold:

1. How to make the attribute preference score assignment adaptive for different attributes? Our goal is to have an intuitive assignment for each attribute without human involvement. The difficulty is that different attributes can have totally different attribute values.
2. How to establish the correspondence between different attributes? For example, how can we know that the desirableness of "Year = 2005" is the same as the desirableness of "Mileage = 5000" for most users?

Input: Price histogram $H_{D1}=\{(c_1,low_1,up_1), ..., (c_m, low_m, up_m)\}$

Price value t

Output: Price score v

Method:

1. $sum = \sum_i c_i$
2. $S_t = 0$
3. For $i = 1..m$
4. if $(low_m > t)$ $S_t = S_t + c_i$
5. if $(low_m < t < up_m)$ $S_t = S_t + c_i * (t - low_m)/(up_m - low_m)$
6. $v = S_t / sum$
7. return v

Figure 3: Price value score assignment algorithm.

We solve the problem in two steps. First, based on the non-Price assumption, we can convert a non-Price value a_i to a Price attribute value t_i:

- If A_i is a categorical attribute, t_i is the average price for all tuples in D such that $A_i = a_i$.
- If A_i is a numerical attribute, v_i is the average price for all tuples in D such that $a_i - d < A_i < a_i + d$ where d is used to prevent too few tuples or no tuple being collected if we just simply set $A_i = a_i$.

In our experiments, a bi-attribute histogram (A_1, A_i) is used when a_i is converted to a Price value. The bi-attribute histograms are built in the pre-processing step in a way similar to the histogram construction described in Section 4.

Second, after converting all non-Price attribute values to Price values, we use a uniform mechanism to assign them a preference score. We assign a large score for a large Price value according to the non-Price assumption. That is, given a converted Price value t_i, a preference score v_i is assigned to it as the percentage of Price values that is smaller than t_i in D. The algorithm for the converted Price preference score assignment can be easily adapted from the algorithm in Figure 3.

5.1 Examples of Attribute Preference Score Assignment

Table 3 shows the average Price and assigned score for different Make values for the carDB database used in our experiments. It can be seen that the prices for different car makes fit our intuition well. Luxury cars are evaluated to have a higher price than standard cars and, consequently, are assigned a larger preference score. We found that the attribute preference assignments for other attributes in carDB are intuitive too.

Table 3: Make-Price-Score correspondence.

Make	Average Price	Score
Mitsubishi	12899	0.183
Volkswagen	16001	0.372
Honda	16175	0.373
Toyota	16585	0.387
Acura	20875	0.599
BMW	33596	0.893
Benz	37930	0.923

6. EXPERIMENTS

In this section, we describe our experiments, report the QRRE experimental results and compare them with some related work. We first introduce the databases we used and the related work for comparison. Then we informally give some examples of query result ranking to provide some intuition for our experiments. Next, a more formal evaluation of the ranking results is presented. Finally, the running time statistics are presented.

6.1 Experimental Setup

To evaluate how well different ranking approaches capture a user's preference, five postgraduate students were invited to participate in the experiments and behave as buyers from the E-commerce databases.

6.1.1 Databases

For our evaluation, we set up two databases from two domains in E-commerce. The first database is a used car database carDB(Make, Model, Year, Price, Mileage, Location) containing 100,000 tuples extracted from Yahoo! Autos. The attributes Make, Model, Year and Location are categorical attributes and the attributes Price and Mileage are numerical attributes. The second database is a real estate database houseDB(City, Location, Bedrooms, Bathrooms, Sq Ft, Price) containing 20,000 tuples extracted from Yahoo! Real Estate. The attributes City, Location, Bedrooms and Bathrooms are categorical attributes and the attributes Sq Ft and Price are numerical attributes. To simulate the Web databases for our experiments we used MySQL on a P4 3.2-GHz PC with 1GB of RAM . We implemented all algorithms in JAVA and connected to the RDBMS by DAO.

6.1.2 Implemented Algorithms

Besides QRRE described above, we implemented two other ranking methods, which are described briefly below, to compare with QRRE.

RANDOM ranking model: In the RANDOM ranking model, the tuples in the query result are presented to the user in a random order. The RANDOM model provides a baseline to show how well QRRE can capture the user behavior over a random method.

Probabilistic Information Retrieval (PIR) ranking model: A probabilistic information retrieval (PIR) technique, which has been successfully used in the Information Retrieval field, is used in [8] for ranking query results. This technique addresses the same problem as does QRRE. In PIR, given a tuple *t*, its ranking score is given by the following equation:

$$Score(t) = \prod_{y \in Y} \frac{p(y \mid W)}{p(y \mid D)} \prod_{y \in Y} \prod_{x \in X} \frac{p(x \mid y, W)}{p(x \mid y, D)}$$

in which X is the specified attributes, Y is the unspecified attributes, W is a past query workload and p denotes the probability.

As mentioned in Section 2, PIR work focuses on point queries without considering range queries. Therefore, when applying the PIR ranking model, the numerical attributes Price and Mileage in carDB and Sq Ft and Price in houseDB are discretized into meaningful ranges as categories, which in reality requires a domain expert.

In PIR, a workload is required to obtain the conditional probability used to measure the correlation between specified attribute values present in the query and the unspecified attributes. In our experiments, we requested 5 subjects to behave as different kinds of buyers, such as rich people, clerks, students, women, etc. and post queries against the databases. We collected 200 queries for each database and these queries are used as the workload W for the PIR model.

6.2 Examples of Query Result Ranking

When we examine the query result rankings, we find that the ranking results of both QRRE and PIR are much more reasonable and intuitive than that of RANDOM. However, there are some interesting examples that show that the QRRE rankings are superior to those of PIR. We found that the ranking result of QRRE is more reasonable than that of PIR in several ways:

- QRRE can discover an assumption that is implicitly held by a buyer. For example, for a query with condition "Mileage < 5000". QRRE ranks cars with Year = 2006 as the top recommendation. Intuitively, this is because a 2006 model year car usually has lower mileage and this is what the user is looking for. However, PIR is unable to identify the importance of Year because most users assume that Mileage itself is enough to represent their preference and, consequently, the relationship between Year and Mileage is not reflected in the workload.
- In PIR, given a numerical attribute, its value range needs to be discretized into meaningful categories and the values within a category are assumed to be the same during ranking. For example, if we assign a car with "Mileage < 10000" to be a category "Mileage = small", then PIR will treat "Mileage = 2000" to be the same as "Mileage = 9000", which is obviously unreasonable. In contrast, QRRE will identify that "Mileage = 2000" is more desirable than "Mileage = 9000".
- QRRE considers the value difference of the specified attributes of the tuples in the query result, while PIR ignores the difference. For example, for a query with condition "Make = Mercedes-Benz and Model = ML500 and Year > 2003", QRRE usually ranks the cars that are made in this year first. However, PIR does not take the Year difference in the query result records into consideration during ranking.

Likewise, QRRE often produces a ranking better than does PIR for houseDB. The actual evaluation in the following section confirms these observations.

6.3 Ranking Evaluation

We now present a more formal evaluation of the query result ranking quality. A survey is conducted to show how well each ranking algorithm captures the user's preference. We evaluate the query results in two ways: average precision and user preference ranking.

6.3.1 Average Precision

In this experiment, each subject was asked to submit three queries for carDB and one query for houseDB according to their preference. Each query had on average 2.2 specified attributes for carDB and 2.4 specified attributes for houseDB. We found that all the attributes of carDB and houseDB were specified in the collected queries at least once. On average, for carDB, each query had a query result of 686 tuples, with the maximum being 4,213 tuples and the minimum 116 tuples. It can be seen that the many-query-result problem is a common problem in reality. Each query for houseDB has a query result of 166 tuples on average.

Since it is not practical to ask the subjects to rank the whole query result for a query, we adopt the following strategy to compare the performance of different ranking approaches. For each implemented ranking algorithm, we collected the first 10 tuples that it recommended. Hence, thirty tuples are collected in total. If there is overlap among the recommended tuples from different algorithms, we extract more tuples using the RANDOM algorithm so that thirty unique tuples are collected in total. Next, for each of the fifteen queries, each subject was asked to rank the top 10 tuples as the relevant tuples that they preferred most from the thirty unique tuples collected for each query. During ranking, they were asked to behave like real buyers to rank the records according to their preferences.

Table 4: Average precision for different ranking methods for carDB.

	QRRE	PIR	RANDOM
q1	0.72	0.52	0.08
q2	0.62	0.62	0.06
q3	0.72	0.22	0.06
q4	0.52	0.64	0.04
q5	0.84	0.78	0.06
q6	0.68	0.36	0.04
q7	0.92	0.46	0.02
q8	0.88	0.64	0.06
q9	0.78	0.62	0.04
q10	0.74	0.64	0.04
q11	0.56	0.66	0.06
q12	0.86	0.76	0.08
q13	0.84	0.36	0.02
q14	0.58	0.38	0.04
q15	0.76	0.66	0.06
Average	0.735	0.555	0.048

We use the Precision/Recall metrics to evaluate how well the user's preference is captured by the different ranking algorithms. Precision is the ratio obtained by dividing the number of retrieved tuples that are relevant by the total number of retrieved tuples. Recall is the ratio obtained by dividing the number of relevant tuples by the number of tuples that are retrieved. In our experiments, both the relevant tuples and the retrieved tuples are 10, which make the Precision and Recall to be equal. Table 4 shows the average precision of the different ranking methods for each query. It can be seen that both QRRE and PIR consistently have a higher precision than RANDOM. For 11 queries out of 15, the precision of QRRE is higher than that of PIR. The precision of QRRE is equal to that of PIR for two queries and is lower than that of PIR for the remaining two queries. QRRE's average precision is 0.18 higher than that of PIR. QRRE has a precision higher than 0.5 for each query while PIR has a precision as low as 0.22 for q3. It should be noted that there is some overlap between the top-10 ranked results of QRRE and top-10 ranked results of PIR for most queries. Figure 4 and Figure 5 show the average precision of the three ranking methods graphically for both carDB and houseDB.

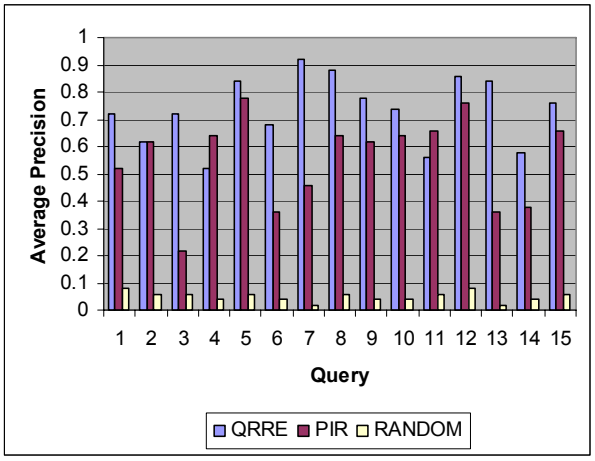

Figure 4: Average prevision for different ranking methods for carDB.

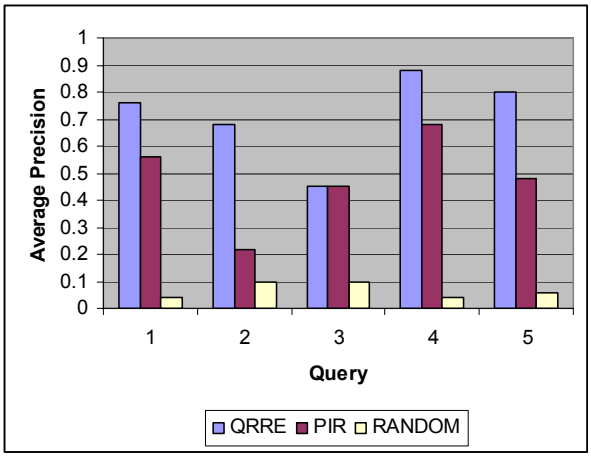

Figure 5: Average precision for different ranking methods for houseDB.

6.3.2 User Preference Ranking

In this experiment, 10 queries were collected from the 5 subjects for carDB and 5 queries were collected for houseDB. After getting the query results, they were ranked using the three ranking methods. The ranking results were then provided to the subjects in order to let them select which result they liked best.

Table 5 and Table 6 show the user preference ranking (UPR) of the different ranking methods for each query for carDB and houseDB, respectively. It can be seen that again both QRRE and PIR greatly outperform RANDOM. In most cases, the subjects preferred the ranking results of QRRE to that of PIR.

Table 5: UPR for different ranking methods for carDB.

	QRRE	PIR	RANDOM
q1	0.8	0.2	0
q2	1	0	0
q3	0.6	0.4	0
q4	0.4	0.6	0
q5	0.4	0.6	0
q6	0.8	0.2	0
q7	1	0	0
q8	0.6	0.2	0.2
q9	0.8	0.2	0
q10	0.8	0.2	0
Average	0.72	0.26	0.02

Table 6: UPR for different ranking methods for houseDB.

	QRRE	PIR	RANDOM
q1	0.4	0.4	0.2
q2	0.8	0.2	0
q3	1	0	0
q4	0.4	0.6	0
q5	0.6	0.4	0
Average	0.66	0.32	0.02

While these preliminary experiments indicate that QRRE is promising and better than the existing work, a much larger scale user study is necessary to conclusively establish this finding.

6.4 Performance Report

Using QRRE, histograms need to be constructed before the query results can be ranked. The histogram construction time depends on the number of buckets and the time to query the web to get the number of occurrences for each bucket. However, in most cases the histogram usually does not change very much over time and so needs to be constructed only once in a given time period.

The query result ranking in the online processing part includes four modules: the attribute weight assignment module, the attribute-value preference score assignment module, the ranking score calculation module and the ranking score sorting module. Each of the first three modules has a time complexity of O(n) after constructing the histogram, where n is the number of query results, and the ranking score sorting module has a time complexity of O($n$log(n)). Hence, the overall time complexity for the online processing stage is O($n$log(n)).

Figure 6 shows the online execution time of the queries over carDB as a function of the number of tuples in the query result. It can be seen that the execution time of QRRE grows almost linearly with the number of tuples in the query result. This is because ranking score sorting is fairly quick even for a large amount of data and thus most of the running time is spent in the first three modules.

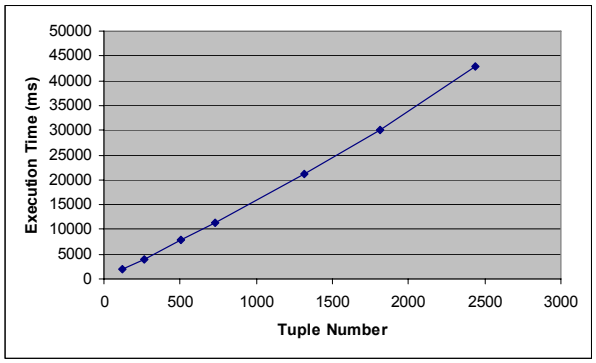

Figure 6: Execution times for different numbers of query results for carDB.

7. CONCLUSION

In this paper, a novel automated ranking approach for the many-query-result problem in E-commerce is proposed. Starting from the user query, we assume that the specified attributes are important for the user. We also assume that the attributes that are highly correlated with the query also are important to the user. We assign a weight for each attribute according to its importance to the user. Then, for each value in each tuple of the query result, a preference score is assigned according to its desirableness in the E-commerce context, where users are assumed to more prefer products with lower prices. All preference scores are combined according to the attribute weight assigned to each attribute. No domain knowledge or user feedback is required in the whole process. Preliminary experimental results indicate that QRRE captures the user preference fairly well and better than existing works.

We acknowledge the following shortcoming of our approach, which will be the focus for our future research. First, we do not deal with string attributes, such as book titles or the comments for a house, contained in many Web databases. It would be extremely useful to find a method to incorporate string attributes into QRRE. Second, QRRE has only been evaluated on small-scale datasets. We realize that a large, comprehensive benchmark should be built to extensively evaluate a query result ranking system, both for QRRE and for future research. Finally, QRRE has been specifically tailored for E-commerce Web databases. It would be interesting to extend QRRE to also deal with non-E-commerce Web databases.

ACKNOWLEDGMENTS

This research was supported by the Research Grants Council of Hong Kong under grant HKUST6172/04E.

REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. "Self-tuning Histograms: Building Histograms Without Looking at Data," *Proc. of the ACM SIGMOD Conf.*, 181-192, 1999.

[2] S. Agrawal, S. Chaudhuri and G. Das. "DBXplorer: A System for Keyword Based Search over Relational Databases," *Proc. of 18th Intl. Conf. on Data Engineering*, 5-16, 2002.

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

[4] A. Balmin, V. Hristidis and Y. Papakonstantinou. "ObjectRank: Authority-Based Keyword Search in Databases," *Proc. of the 30th Intl. Conf. on Very Large Databases*, 564-575, 2004.

[5] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti and S. Sudarshan. "Keyword Searching and Browsing in Databases using BANKS," *Proc. of 18th Intl. Conf. on Data Engineering*, 431-440, 2002.

[6] N. Bruno, S. Chaudhuri and L. Gravano. "STHoles: A Multidimensional Workload-aware Histogram," *Proc. of the ACM SIGMOD Conf.*, 211-222, 2001.

[7] K. Chakrabarti, S. Chaudhuri and S. Hwang. "Automatic Categorization of Query Results," *Proc. of the ACM SIGMOD Conf.*, 755-766, 2004.

[8] S. Chaudhuri, G. Das, V. Hristidis and G. Weikum. "Probabilistic Ranking of Database Query Results," *Proc. of the Intl. Conf. on Very Large Databases*, 888-899, 2004.

[9] K. Chakrabarti, K. Porkaew and S. Mehrotra. "Efficient Query Refinement in Multimedia Databases," *Proc. of 16th Intl. Conf. on Data Engineering*, 196, 2000.

[10] W. Cohen. "Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity," *Proc. of the ACM SIGMOD Conf.*, 201-212, 1998.

[11] W. Cohen. "Providing Database-like Access to the Web Using Queries Based on Textual Similarity," *Proc. of the ACM SIGMOD Conf.*, 558-560,1998.

[12] W.B. Croft and J. Lafferty. *Language Modeling for Information Retrieval*. Kluwer 2003.

[13] R.O. Duda, P.E. Hart and D.G. Stork, *Pattern Classification*. John Wiley & Sons, USA, 2001.

[14] N. Fuhr. "A Probabilistic Framework for Vague Queries and Imprecise Information in Databases," *Proc. of the 16th Intl. Conf. on Very Large Databases*, 696-707, 1990.

[15] N. Fuhr. "A Probabilistic Relational Model for the Integration of IR and Databases," *Proc. of the ACM SIGIR Conf*., 309-317, 1993.

[16] F. Geerts, H. Mannila and E. Terzi. "Relational Link-based Ranking," *Proc. of the 30th Intl. Conf. on Very Large Databases*, 552-563, 2004.

[17] V. Hristidis and Y. Papakonstantinou. "DISCOVER: Keyword Search in Relational Databases," *Proc. of the 28th Intl. Conf. on Very Large Databases*, 670-681, 2002.

[18] G. Koutrika and Y.E. Ioannidis. "Personalization of Queries in Database Systems," *Proc. of 20th Intl. Conf. on Data Engineering*, 597-608, 2004.

[19] G. Koutrika and Y.E. Ioannidis. "Constrained Optimalities in Query Personalization," *Proc. of the ACM SIGMOD Conf*., 73-84, 2005.

[20] Y.E. Ioannidis. "The History of Histograms (abridged)," *Proc. of the 29th Intl. Conf. on Very Large Databases*, 19-30, 2003.

[21] W. Kießling. "Foundations of Preferences in Database Systems," *Proc. of the 28th Intl. Conf. on Very Large Databases*, 311-322, 2002.

[22] R. Kooi. *The Optimization of Queries in Relational Databases*. PhD Thesis, Case Western Reserve University, 1980.

[23] I. Muslea and T. Lee. "Online Query Relaxation via Bayesian Causal Structures Discovery," *Proc. of the AAAI Conf*., 831-836, 2005.

[24] U. Nambiar and S. Kambhampati. "Answering Imprecise Queries over Autonomous Web Databases," *Proc. of 22nd Intl. Conf. on Data Engineering*, 45, 2006.

[25] Z. Nazeri, E. Bloedorn and P. Ostwald. "Experiences in Mining Aviation Safety Data," *Proc. of the ACM SIGMOD Conf*., 562-566, 2001.

[26] M. Ortega-Binderberger, K. Chakrabarti and S. Mehrotra. "An Approach to Integrating Query Refinement in SQL," *Proc. Intl. Conf. on Extending Data Base Technology*, 15-33, 2002.

[27] G. Piatetsky-Sharpiro and C. Connell. "Accurate Estimation of the Number of Tuples Satisfying a Condition," *Proc. of the ACM SIGMOD Conf.,* 256--276, 1984.

[28] Y. Rui, T.S. Huang and S. Merhotra. "Content-Based Image Retrieval with Relevance Feedback in MARS," *Proc. IEEE Intl. Conf. on Image Processing*, 815-818, 1997.

[29] G. Salton, A. Wong and C.S. Yang. "A Vector Space Model for Information Retrieval," *Communications of the ACM* **18**(11), 613-620, 1975.

[30] K. Sparck Jones, S. Walker and S.E. Robertson. "A Probabilistic Model of Information Retrieval: Development and Comparative Experiments - Part 1," *Inf. Process. Management* **36**(6), 779-808, 2000.

[31] K. Sparck Jones, S. Walker and S.E. Robertson. "A Probabilistic Model of Information Retrieval: Development and Comparative Experiments - Part 2," *Inf. Process. Manage*ment **36**(6), 809-840, 2000.

[32] E.M. Voorhees. "The TREC-8 Question Answering Track Report," *Proc. of the 8th Text Retrieval Conf*, 1999.

[33] L. Wu, C. Faloutsos, K. Sycara and T. Payne. "FALCON: Feedback Adaptive Loop for Content-Based Retrieval," *Proc. of the 26th Intl. Conf. on Very Large Databases*, 297-306, 2000.