Answering Approximate Queries over Autonomous Web Databases

Xiangfu Meng
College of Information Science and Engineering, Northeastern University
Shenyang, China, 110004
marxi@126.com

Z. M. Ma
College of Information Science and Engineering, Northeastern University
Shenyang, China, 110004
mazongmin@ise.neu.edu.cn

Li Yan
School of Software
Northeastern University
Shenyang, China, 110004
yanlilyjiaji@163.com

ABSTRACT
To deal with the problem of empty or too little answers returned from a Web database in response to a user query, this paper proposes a novel approach to provide relevant and ranked query results. Based on the user original query, we speculate how much the user cares about each specified attribute and assign a corresponding weight to it. This original query is then rewritten as an approximate query by relaxing the query criteria range. The relaxation order of all specified attributes and the relaxed degree on each specified attribute are varied with the attribute weights. For the approximate query results, we generate users' contextual preferences from database workload and use them to create a priori orders of tuples in an off-line preprocessing step. Only a few representative orders are saved, each corresponding to a set of contexts. Then, these orders and associated contexts are used at query time to expeditiously provide ranked answers. Results of a preliminary user study demonstrate that our query relaxation and results ranking methods can capture the user's preferences effectively. The efficiency and effectiveness of our approach is also demonstrated by experimental result.

Categories and Subject Descriptors
H.2.3 [**Database Management**]: Languages - *Query languages*; H.2.4 [**Database Management**]: Systems - *Query processing*

General Terms
Algorithms, Performance, Experimentation, Human Factors.

Keywords
Web database, query relaxation, query results ranking, top-k.

1. INTRODUCTION
Nowadays, there is more and more interest in using the World Wide Web, especially, for searching and retrieving information over Web database[1] that are available "on-line". Exploiting Web-based information sources is non-trivial because the user has no direct access to the data. Users in general accomplish their search using Boolean queries and an item from the database simply either matches or it does not. In such context, users may be confronted with the following two problems:

1. Empty answers: When the query is too selective, the answer may be empty or too little. In that case, it is desirable to have the option of relaxing the original query for presenting more relevant items that can meet user's needs and preferences closely.

2. Many answers: When the query is not too selective, too many tuples may be in the answer. In such a case, it will be desirable to have the option of order the matches automatically that ranks more "globally important" answer tuples higher and returning only the best matches.

In the first case, several approaches have been proposed to deal with this issue [4, 16, 19]. The basic idea of these approaches is based on reducing the constraints on the original query in order to expand the scope of the query. However, most existing work does not consider the user preferences when relaxing the original query. But in real applications the efficiency of query relaxation is affected greatly by the user preferences.

In this paper, we tackle the empty answers problem for Web database by proposing an automatic relaxing and ranking approach, AQRR (approximate query[2] & results ranking), which can relax the original query and rank the approximate query results from a Web database in a domain and user independent way. We will use the illustrative examples below to motivate and provide an overview of our approach.

Example 1. Consider a used car selling Web database CarDB (Make, Model, Price, Color, Engine, Year, Mileage). Each tuple in CarDB represents a used car for sale. Based on the used car database the user may issue the following query:

Q: -CarDB (Model = Camry and Price < 10000)

On receiving the query, CarDB will provide a list of a few *Camrys* that is priced below \$10000 (since there are very few *Camrys* priced below \$10000). In such a case, the traditional query relaxation approaches will expand each conditions of the original query with the same relaxation degree to provide the relevant answers. However, in the real world the user who submitted this query may care more about *Price* than *Model* (because there are few used cars priced below \$10000 and then we can speculate that the user concerns more about the *Price*) and the relaxation degree of the query criteria on *Price* should be relaxed smaller than that on *Model*. Hence, we need to surmise

[1] We use the term "Web database" to refer to a non-local autonomous database that is accessible only via a Web (form) based interface.

[2] A user query that requires a close but not necessarily exact match is an approximate query.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW'09, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

the user's preferences on different attributes when relaxing the scope of the original query.

In this paper, our solution for relaxing a given query Q is to generate an approximate query $\tilde{Q}$ by reducing the constraints of the original query. The underlying motivation is that the tuples most similar to the original query will have differences only in the least important attribute specified by the query and the relaxation degree for each specified attribute should be different according to the importance of the attribute to the specific user. The first attribute to be relaxed must be the least important specified attribute and has the maximum relaxation degree. The intuition of our attribute weight measuring approach is that the importance of specified attribute for the user can be reflected by the distribution of the specified attribute value in the database.

However, after relaxing the original query, another problem faced by users will be that there are usually many answers returned for an approximate query. To resolve the many answers problem, two types of solutions have been proposed. The first type [5, 9] categorizes the query results into a navigational tree, and the second type [3], [6], [12] ranks the results. The success of both approaches depends on the utilization of user preferences. But these approaches do not consider the contexts in which the preferences appear. However, in real applications the preferences are often associated to specified contexts, i.e. contextual preferences, which take the form $i_1 \succ i_2$, d | X, meaning that item i_1 is preferred to item i_2 with the interest degree d in the context of X.

Example 2. Consider the used car relation CarDB mentioned above. Assume that we have the following contextual preferences:

Model=Camry $\succ$ Model=Accord, 0.6 | Price = 20000
Model=Accord $\succ$ Model=Camry, 0.8 | Price = 25000

These preferences illustrate that the ranking of the tuples of a relation is subjective. In the context of a used car that is priced around $20,000, people may prefer *Camry* to *Accord* with interest degree 0.6. In contrast, in the context of a used car that is priced around $25,000, people may prefer *Accord* to *Camry* with interest degree 0.8. This example shows that one can not rank objects independently of the context in which they appear.

In this paper, we propose a contextual preference model of the form $\{i_1 \succ i_2$, d | $X\}$, meaning that item i_1 is preferred to item i_2 with the interest degree d in the context of X. Based on contextual preferences, we incorporate the set of contextual preferences $\mathcal{P}$ into the query results ranking mechanism. Specifically, for a given approximate query, the contextual preferences that are related to it are taken into account to provide ranked top-k results. We collect the contextual preferences from the database workload by using association rules. These contextual preferences are used to create a priori orders of tuples in an offline processing step. Only a few representative orders are saved, each corresponding to a set of contexts. For an incoming approximate query $\tilde{Q}$, we first evaluate the similarity between $\tilde{Q}$ and contexts, and then quickly provide the ranked results that agree with the orders as much as possible.

Our contributions are summarized as follows:

- We propose a query relaxation method to solve the empty answers problem and provide relevant answers for a user query over the autonomous Web database. This method considers both the data distribution and user preferences when relaxing the original query.
- We propose a ranking method for the approximate query results. This method uses pre-computation, clustering and top-k algorithm to deal with the large result tuples.

The rest of this paper is organized as follows. Section 2 reviews some related work. Section 3 proposes our query relaxation method. Section 4 gives the definition of contextual preferences and outlines the ranking method of approximate query results while the algorithmic solutions for it are discussed in Section 5. The experiment results are presented in Section 6. The paper is concluded in Section 7.

2. RELATED WORK

Several researches have been proposed to deal with the empty answers problem. These researches can be classified into two main categories. The first one is based on fuzzy set theory such as [4] and [15], which relaxes the query criteria by using membership functions, domain knowledge and α-cut operation of fuzzy number. The second category focuses on the development of cooperative database systems such as [16, 19] which handle the query relaxation based on distance notion, data distribution, etc. However, it should be noted that the approaches based on fuzzy sets are highly dependent on the domain knowledge and it is mainly useful in expanding the numerical query criteria range while the cooperative database system usually requires the user feedback. Furthermore, both of these two types of approaches seldom consider the user preferences when relaxing the query and ranking the query results. Compared with the above work, our approach is fully automatic and does not require the domain knowledge. Our approach also takes the user preferences into consideration when relaxing the query and ranking the answers.

There is also some work on ranked retrieval from a database. In [20] and [23], user relevance feedback is employed to learn the similarity between a result tuple and the query. In [14] and [17], the SQL query language is extended to allow the user to specify the ranking function according to their preference for the attributes. In [3] and [8], the importance scores of tuples in a relation are extracted automatically by analyzing the workloads, which can reveal what users are looking for and what they consider as important. In [10] and [14], a quantitative and a qualitative preference model were proposed, respectively. In the first, preferences are specified indirectly using scoring functions that associate a numeric score with every tuple of the query answer. While in the second, preferences among tuples are specified directly using binary preference relations. Recently, several works started to consider the contextual preferences for ranking database query results, such as [1], [21] and [22]. We make use of some of these ideas, but enhance the contextual preferences with the interest degrees and focus on how the preferences associated with different contexts and interest degrees have impact on the query results.

The work that is most similar to ours is the AIMQ in [16], which addresses the problem of answering imprecise queries over autonomous Web databases by using approximate functional dependencies and approximate keys. Our approach differs from that in [16] in the following aspects:

1. AIMQ learns the attribute importance based on pre-extracted data and it can only determine the attribute importance sequence without the specific weights. The attribute importance sequence is also invariant to the different user queries. In contrast, our approach speculates how much the user cares about each specified

attribute according to the user's query and thus the attribute importance can be tailored to the user preferences.

2. AIMQ only takes the similarities between an imprecise query and answer tuples into consideration for ranking the relevant answer tuples while it ignores the impact of user's preferences on ranking. On the contrary, AQRR considers both the similarities of the query and answer tuples and the user preferences.

3. APPROXIMATE QUERY

3.1 Problem Definition

Definition 1. (approximate query) Consider an autonomous Web database R with categorical and numerical attributes $A = \{A_1, A_2, \dots, A_m\}$ and a query Q over R with a conjunctive selection condition of the form $Q = \wedge_{i\in\{1,\dots,k\}}(A_i\,\theta\, a_i)$, where $k \le m$ and $\theta \in \{>, <, =, \neq, \geq, \leq, \text{between}\}$. Note that, if θ is the operator *between* and a_i is an interval which is represented by $[a_{i1}, a_{i2}]$, $A_i\,\theta\, a_i$ has the form of "A_i *between* a_{i1} AND a_{i2}". Each A_i in the query condition is an attribute from A and a_i is a value (or interval) in its domain. By relaxing Q, an approximate query $\tilde{Q}$ which is used to find all tuples of R that show similarity to Q above a threshold $T_{\text{sim}}\in(0, 1)$ is obtained. Specifically,

$$\tilde{Q}(R) = \{t \mid t\in R, \text{Similarity}(Q, t) > T_{\text{sim}}\}.$$

3.2 Attribute Weight Assignment

In the real world, different users have different preferences, thus the attribute importance is usually different for different types of users. The query criteria user specified can reflect the user's preferences on the attribute. For instance, for a query with condition "Year = 2007 and Price < 10000", the specified attribute Year is less important for user (there may be many used cars have the date of shipment in 2007) than the attribute Price (relatively fewer used cars priced below \$10,000). Hence, we will assign the weight for each specified attribute according to the distribution of its value specified by the user in the database.

3.2.1 Importance of Specified Categorical Attributes

The well-known IDF method has been used extensively in IR to suggest that commonly occurring words convey less information about user's needs than rarely occurring words, and thus should be weighted less. $IDF(w)$ of a word w is defined as $\log(n/F(w))$ where n is the number of documents, and $F(w)$ is the number of document in which w appears. If the database only had categorical attributes, each tuple can be treated as a small document. Thus, we can mimic these techniques for our problem.

For a point query "$A_i = v$", we define $IDF_i(v)$ as $\log(n/F_i(v))$ which represents the importance of attribute value v in the database, where n is the number of tuples in the database and $F_i(v)$ is the frequency of tuples in the database of $A_i = v$. As mentioned above, the importance of specified categorical attribute value is treated as the importance of its corresponding attribute.

3.2.2 Importance of Specified Numerical Attributes

For evaluating the importance of numerical attribute values, it is inappropriate to adopt the definition of traditional IDF as above mentioned because of their binary nature (where if u and v are arbitrarily close to each other yet distinct). Moreover, the "IDF" of a numeric value should depend on nearby values.

In this paper, we adopt the definition given in [3] to measure the similarity of numeric attribute values. Let $\{v_1, v_2, \dots, v_n\}$ be the values of attribute A that occur in the database. For specified attribute value v in the query, it defined $IDF(v)$ as shown in Equation (1), where h is the bandwidth parameter.

$$IDF(v) = \log\left(\frac{n}{\sum_{i}^{n} e^{-\frac{1}{2}\left(\frac{v_i - v}{h}\right)^2}}\right) \quad (1)$$

A popular estimate for the bandwidth is $h = 1.06\sigma n^{-1/5}$, where σ is the standard deviation of $\{v_1, v_2, \dots, v_n\}$. Intuitively, the denominator in Equation (1) represents the sum of "contributions" to v from every the other point v_i in the database. These contributions are modeled as (scaled) Gaussian distributions, so that the further v is from v_i, the smaller is the contribution from v_i. For example, "Price = 10000" is sparse in its domain and other values are far from it, thus the value "10000" will get a large IDF. The importance of specified numerical attribute value is also treated as the importance of its corresponding attribute.

Moreover, if query condition is generalized as "A_i *IN* Q_i", where Q_i is a set of values for categorical attributes, or a range $[lb, ub]$ for numeric attributes, We define the maximum $\log(n/F_i(v))$ of each different value v in Q_i. The generalized importance measuring function is shown in Equation (2).

$$IDF_i(v) = \max_{v\in Q} IDF_i(v) \quad (2)$$

By normalized processing, the weight w_i of attribute A_i specified by the query can be calculated by

$$w_i = \frac{IDF_i(v)}{\sum_{i=1}^{k} IDF_i(v)} \quad (3)$$

in which, k is the number of attributes specified by the query.

3.3 Attribute Values Similarity Assessment

3.3.1 Similarity of Categorical Attribute Values

We discuss an approach for deriving the similarity coefficient between two categorical attribute values. It is an adaptation of the method proposed in [16]. The similarity between two values binding a categorical attribute is measured as the percentage of common AV-pairs that are associated to them. Given a categorical value, all the AV-pairs associated to the value can be seen as the features describing the value. The similarity between two values can be estimated by the commonality in the features (AV-pairs) describing them. For example, given tuple t ={Toyota, Camry, 15k, 2008}, the AV-pair Model=Camry is associated to the AV-pairs Make = Toyota, Price = 15000 and Year = 2008.

An AV-pair can be visualized as a selection query that binds only a single attribute. By issuing an AV-pair query (such as "Model = Camry") over the extracted database, a set of tuples all containing the AV-pair can be identified. The answer set containing each AV-pair as a structure is called the *supertuple*. The supertuple contains a *Set* of keywords for each attribute in the relation not bound by the AV-pair. Table 1 shows the supertuple for "Model = Camry" over the relation CarDB as a 2-column tabular structure.

Table 1. Supertuple for Model = "Camry"

Make	Toyota: 112
Price	5000-10000: 15, 10000-30000: 40,…
Mileage	10000-20000: 12, 20000-40000: 25,…
Color	Black: 46, Silver: 15, …
Year	2008: 17, 2007: 37,…

The similarity between two AV-pairs can be measured as the similarity shown by their supertuples. The supertuples contain sets of keywords for each attribute in the relation and the *Jaccard Coefficient* is used to determine the similarity between two supertuples. Thus, in this paper the similarity coefficient between two categorical values is then calculated as a sum of the *Set* similarity on each attribute,

$$\text{VSim}(C_1, C_2) = \sum_{i=1}^{m} \text{J}(C_1.A_i, C_2.A_i) \tag{4}$$

where C_1, C_2 are supertuples with m attributes, A is the *Set* corresponding to the i-th attribute, J(,) is the Jaccard Coefficient and is computed as $\text{J(A, B)} = |\text{A}\cap\text{B}|/|\text{A}\cup\text{B}|$.

3.3.2 Similarity of Numerical Attribute Values

Because of the continuity of numerical data, we propose an approach to estimate the similarity coefficient between two numerical values. Let $\{v_1, v_2, \ldots, v_n\}$ be the values of numerical attribute A occurring in the database. Then the similarity coefficient NSim(v, q) between v and q can be defined by Equation (5), where h is the same as mentioned in Equation (1).

$$\text{NSim}(v, q) = \frac{1}{1+\left(\dfrac{v-q}{h}\right)^2} \tag{5}$$

For a numerical condition $A_i = q$ of Q, let ψ_i be a sub-threshold for A_i, according to Equation (5), we can then get the relaxation range of numerical attribute A_i as follows:

$$[q - h\sqrt{\frac{1-\psi_i}{\psi_i}}\ ,\ q + h\sqrt{\frac{1-\psi_i}{\psi_i}}\ ] \tag{6}$$

3.4 Query Relaxation

The sub-threshold for each specified attribute should be computed according to the attribute weights and the threshold for the query. Given a conjunctive selection query Q to be executed over database table R, we assume w_i is the weight of specified attribute A_i, k is the number of specified attributes, T_{sim} is the given threshold by the user. Then, the sub-threshold ψ_i for each specified attribute in Q can be calculated as follows:

$$\psi_{i\,(i=1,\ldots,k)} = \begin{cases} \dfrac{w_1}{\psi_1} =,\ldots,= \dfrac{w_k}{\psi_k} \\ \text{T}_{\text{sim}} = \sum_{i=1}^{k} w_i \cdot \psi_i \end{cases} \tag{7}$$

The query rewriting algorithm (Algorithm 1) is shown as follows. For each condition C_i in the original query Q, by extracting values of its corresponding attribute A_i having similarity above the sub-threshold ψ_i and adding them into its query range, we can get the relaxed condition $\tilde{C}_i$. By join all the relaxed conditions, the approximate query $\tilde{Q}$ is formed.

Algorithm 1 The query rewriting algorithm

Input: Original query $Q = \{C_1, \ldots, C_k\}$, sub-threshold $\{\psi_1, \ldots, \psi_k\}$.
Output: An approximate query $\tilde{Q} = \{\tilde{C}_1, \ldots, \tilde{C}_k\}$.

```
1.  For $i = 1, \ldots, k$
2.    $\tilde{C}_i \leftarrow C_i$
3.    If $A_i$ is a categorical attribute
4.      $\forall v \in \text{Dom}(A_i)$
5.      If VSim($a_i$, $v$) > $\psi_i$
6.        Add $v$ into the query range of $\tilde{C}_i$
7.      End If
8.    End If
9.    If $A_i$ is a numerical attribute
10.     Replace query range of $\tilde{C}_i$ with $[q-h\sqrt{\frac{1-\psi_i}{\psi_i}}, q+h\sqrt{\frac{1-\psi_i}{\psi_i}}]$
11.   End If
12.   $\tilde{Q} = \tilde{Q} \cup \tilde{C}_i$
13. End For
14. Return $\tilde{Q}$
```

4. APPROXIMATE QUERY RESULTS RANKING

This section firstly gives a definition of the contextual preferences, and then discusses the method of preferences processing. Finally, the approximate query results ranking problem is defined and its solution is presented based on contextual preferences.

4.1 Contextual Preferences

4.1.1 Definition of Contextual Preferences

Definition 2 (contextual preferences) Contextual preferences are of the form $\{A_i = a_{i1} \succ A_i = a_{i2}, \text{d} \mid X\}$, where X is $\bigwedge_{j\in l}(A_j\ \theta\ a_j)$, with $a_{i1}, a_{i2} \in \text{Dom}(A_i)$, $l \subseteq \{1,\ldots, k\}$, $\theta \in \{>, <, =, \neq, \geq, \leq, \text{between}\}$ and $a_j \in \text{Dom}(A_j)$, d is the interest degree of preference (i.e., compared to a_{i2}, the interest degree of a_{i1} is d, where $0.5 \leq \text{d} \leq 1$, and it can be learned from the database workload). The left-hand side of a preference specifies the *choice* and the *interest degree* while the right-hand side is the *context*.

For collecting contextual preferences, we automatically generate preferences by using association-rules mining [2] on the database workload-log of past users queries. The workload is represented as a set of "tuples", where each tuple represents a query and is a vector containing the corresponding values of the specified attributes [8]. The rationale behind this automatic generation of preferences is the following. We say that $\{A_1 = a \succ A_1 = b, \text{d} \mid \text{X}\}$ if $\text{conf}(X \rightarrow a) > \text{conf}(X \rightarrow b)$, where $\text{conf}(X \rightarrow a)$ is the confidence of the association rule $X \rightarrow a$ in the workload, i.e.,

$$\text{conf}(X \rightarrow a) = \frac{\text{frequency}(X \wedge a)}{\text{frequency}(X)}$$

$$\text{d} = \frac{\text{conf}(X \rightarrow a)}{\text{conf}(X \rightarrow a) + \text{conf}(X \rightarrow b)} \tag{8}$$

where, $\text{conf}(X \rightarrow a)$ (resp. $\text{conf}(X \rightarrow b)$) is the frequency of the value a (resp. b) occurring together with context X in the workload, then we can obtain the value of d by using Equation (8). The intuition is that when an attribute value a occurs together with context X in the workload more often than the attribute value b, then this implies that a is also preferred to b over X.

4.1.2 Contextual Preferences Processing

Example 3. Consider the used car relation CarDB of Table 2 and the following preferences for tuples in this relation.

Table 2. CarDB table

Model	Color	Engine	Make	Price	Year
Accord	Silver	2.4L	Honda	30999	2008
Accord	Blue	3.5L	Honda	31999	2007
CR-V	Black	3.0L	Honda	32500	2007
Camry	Blue	3.5L	Toyota	22999	2007
Matrix	Gray	3.3L	Toyota	23999	2007

p_1 = {Model = Accord $\succ$ Model = CR-V, 0.7 | Make = Honda $\wedge$ Price between 30000 and 33000}
p_2 = {Color = Sliver $\succ$ Color = Black, 0.6 | Make = Honda $\wedge$ Price between 30000 and 33000}
p_3 = {Engine = 3.0L $\succ$ Engine = 2.4L, 0.9 | Make = Honda $\wedge$ Price between 30000 and 33000}
p_4 = {Model = Camry $\succ$ Model = Matrix, 0.8 | Make = Toyota $\wedge$ Price between 22000 and 25000}

Under the assumption above, preference p_1 suggests that in the context of "Make=Honda $\wedge$ Price between 30000 and 33000", tuples t_1 and t_2 are preferred to tuple t_3 with the interest degree 0.7. In the same context, from p_2, tuple t_1 is preferred to t_3 with the interest degree 0.6, and from p_3 tuple t_3 is preferred to t_1 with the interest degree 0.9. Finally, from p_4, tuple t_4 is preferred to tuple t_5 in the context of "Make = Toyota $\wedge$ Price between 22000 and 25000" with the interest degree 0.8.

For any single preference p and any pair of tuples (t_i, t_j), p either prefers t_i to t_j (denoted by $t_i \succ_p t_j$) or t_j to t_i (denoted by $t_j \succ_p t_i$) or it is inapplicable with respect to t_i and t_j (denoted by $t_i \sim_p t_j$). Thus, every preference p defines a *preference degree* ($\mathrm{d_{pref}}$) over any pair of tuples t, t' that evaluates as follows:

$$\mathrm{d_{pref}}(t, t', p) = \begin{cases} \mathrm{d} & , \; \textit{if } t \succ_p t' \\ 1\text{-}\mathrm{d} & , \; \textit{if } t' \succ_p t \\ \perp & , \; \textit{if } t \sim_p t' \end{cases}$$

where, d ($0.5 \le \mathrm{d} \le 1$) is the interest degree of the preference p.

For any two contexts $X_1 = \wedge_{j\in l_1}(A_j\theta a_j)$ and $X_2 = \wedge_{j\in l_2}(A_j\theta b_j)$ with l_1, $l_2 \subseteq \{1, 2, \dots, k\}$, we say that they are equal if and only if $l_1 = l_2 = l$ and θa_j is the same as θb_j for all $j \in l$. We say that two preferences $\{A_i = a_{i1} \succ A_i = a_{i2}, \mathrm{d}_1 \mid X_1\}$ and $\{A_j = a_{j1} \succ A_j = a_{j2}, \mathrm{d}_2 \mid X_2\}$ belong to the same *preference class* if $X_1 = X_2 = X$. P_X is used to denote the set of all preferences in the same class defined by context X. For example, preferences p_1, p_2 and p_3 from Example 3 belong to the same class defined by "Make=Honda $\wedge$ Price between 30000 and 33000", while p_4 belongs to a different class.

The set of preferences P_X in a class, partitions the tuples in the relation in two sets, the set of *indifferent* tuples and the set of *asserted* tuples. A tuple is indifferent with respect to a context if no explicit preferences that involve it have been expressed within the context. All tuples that are not indifferent are *asserted*. This paper only considers the *asserted* tuples.

Let's consider the preferences of the same class and defines the effective preference ($\mathrm{P_{eff\text{-}p}}$) for ordered pairs of tuples (t, t'): for any ordered pair of asserted tuples (t, t') such that there exists a $p \in P_X$ for which $\mathrm{d_{pref}}(t, t', p) + \mathrm{d_{pref}}(t', t, p) = 1$, we then have:

$$\mathrm{P_{eff-p}}(t,t',P_X) = \frac{\sum_{p\in P_X} \mathrm{d_{pref}}(t,t',p)}{\sum_{p\in P_X} (\mathrm{d_{pref}}(t,t',p) + \mathrm{d_{pref}}(t',t,p))}.$$

If for a pair of asserted tuples (t, t') there does not exist a $p \in P_X$ for which $\mathrm{d_{pref}}(t, t', p) + \mathrm{d_{pref}}(t', t, p) = 1$, then

$$\mathrm{P_{eff-p}}(t,t',P_X) = \mathrm{P_{eff-p}}(t',t,P_X) = \frac{1}{2}.$$

If for a pair of tuples (t, t'), t or t' (or both) are indifferent with respect to context X, then: $\mathrm{P_{eff-p}}(t,t',P_X) = \perp$.

Let's consider the preference class $P_{\text{Make = Honda} \wedge \text{Price between 30000 and 33000}}$ from Example 3 to illustrate the definition mentioned above. For this preference class (denote by P_X for brevity), the effective preferences for the tuples are as follows:

(1) $\mathrm{P_{eff\text{-}p}}(t_1, t_2, P_X) = \mathrm{P_{eff\text{-}p}}(t_2, t_1, P_X) = 1/2$.

(2) $\mathrm{P_{eff\text{-}p}}(t_1, t_3, P_X) = (0.7+0.6+0.1)/3 = 7/15$, $\mathrm{P_{eff\text{-}p}}(t_3, t_1, P_X) = (0.3+0.4+0.9)/3 = 8/15$.

(3) $\mathrm{P_{eff\text{-}p}}(t_2, t_3, P_X) = 7/10$, $\mathrm{P_{eff\text{-}p}}(t_3, t_2, P_X) = 3/10$.

(4) $\mathrm{P_{eff\text{-}p}}(., ., P_X) = \perp$.

For a given class of preferences P_X and a relation R, the X-preference graph $G_X(V_X, E_X)$ is defined as: the set of nodes is the set of all asserted tuples in R. For every ordered pair of nodes (t, t') there exists a directed edge $e(t, t') \in E_X$, with weight:

$w_X(t \to t') = \mathrm{p_{eff\text{-}p}}(t, t', P_X)$, and $w_X(t \to t') + w_X(t' \to t) = 1$.

Figure 1 shows the preference graph for $P_{\text{Make = Honda} \wedge \text{Price between 30000 and 33000}}$ for the used car relation and preferences p_1, p_2 and p_3.

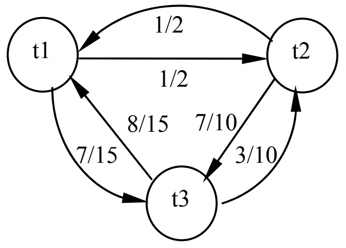

Figure 1. Preference graph for context "Make = Honda $\wedge$ Price between 30000 and 33000"

4.2 The Approximate Query Results Ranking Problem

Consider a conjunctive selection query Q over relation $R = \{t_1,\dots, t_n\}$ with schema $(A_1,\dots,A_m)$. Let $\tilde{Q}$ is an approximate query generated based on Q. The approximate query $\tilde{Q}$ is of the form $\tilde{Q} = \sigma \wedge_{j\in\{1,\dots,k\}} \tilde{C}_j$,where $k \le m$. Let $\tilde{Q}(R) \subseteq R$ is the subset of tuples in R that are in the answer of $\tilde{Q}$. The goal is to address the approximate query results ranking problem defined as follows:

Problem 1. (approximate query results ranking problem): Assume a set of preferences $\mathcal{P} = \{ P_{X_1}, \ldots, P_{X_m} \}$ and an approximate query $\tilde{Q}$. The approximate query results ranking problem asks for an order of τ of $\tilde{Q}(R)$ such that

$$\tau = \arg\max_{\tau'} \sum_{i=1}^{m} \mathrm{sim}(\tilde{Q}, X_i)\mathrm{Agree}(\tau', P_{X_i})$$

where, $\mathrm{Agree}(\tau, P_X) = \sum_{(t,t'):\tau(t)<\tau(t')} \mathrm{P}_{\text{eff-p}}(t, t', P_X)$

The objective of the problem is to find the order τ over the set of tuples in $\tilde{Q}(R)$ that agrees as much as possible with the input preferences. Additionally, the degree of agreement with a class of preferences is weighted by the similarity between the contexts of those preferences and the approximate query $\tilde{Q}$.

We adopt cosine similarity to quantify the similarity between an approximate query $\tilde{Q}$ and a context X. Firstly, we form the set representations of a context and an approximate query. Consider the set $\mathcal{D}$ of all distinct ⟨attribute, value⟩ pairs appearing in the R, that is, $\mathcal{D} = \{\langle A_i, a\rangle \mid \forall i \in \{1,\ldots,k\}$ and $\forall a \in \mathrm{Dom}(A_i)\}$. Since $\mathrm{Dom}(A_i)$ is the active domain of attribute A_i the cardinality of this set is finite. Let $\mathrm{N} = |\mathcal{D}|$ and let O_R be an arbitrary but fixed order on the pairs appearing in $\mathcal{D}$. $\mathcal{D}[i]$ refers to the i-th element of $\mathcal{D}$ based on the ordering O_R. A vector representation of a context $X = \bigwedge_j (A_j = a_j)$ is a binary vector V_X of size N. The i-th element of the vector corresponds to pair $\mathcal{D}[i]$. If $\mathcal{D}[i]$ satisfies one of the conjunctions of X then $V_X[i] = 1$, otherwise it is 0.

The vector representation of an approximate query $\tilde{Q}$ is a vector $V_{\tilde{Q}}$ of size N. The i-th element of the vector corresponds to pair $\mathcal{D}[i]$. If $\mathcal{D}[i]$ satisfies one of the conditions of $\tilde{Q}$, then $V_{\tilde{Q}}[i]$ can be computed as follows:

$$V_{\tilde{Q}}[i] = w_j \times \begin{cases} \mathrm{VSim}(\mathcal{D}[\mathrm{i}].\mathrm{value}, C_j.\mathrm{value}), & \text{if Domain}(C_j.\text{attribute})\text{=categorical} \\ \mathrm{NSim}(\mathcal{D}[\mathrm{i}].\mathrm{value}, C_j.\mathrm{value}), & \text{if Domain}(C_j.\text{attribute})\text{=numerical} \end{cases}$$

where, w_j is the weight of attribute specified by the condition C_j of original query Q. Otherwise, the $V_{\tilde{Q}}[i] = 0$.

Now we can define the similarity between context X and approximate query $\tilde{Q}$ using their vector representations as follows:

$$\mathrm{sim}(\tilde{Q}, X) = \cos(V_{\tilde{Q}}, V_X) = \frac{\sum_{i=1}^{|\mathcal{D}|} V_{\tilde{Q}}[i] \cdot V_X[i]}{\sqrt{\sum_{i=1}^{|\mathcal{D}|} V_{\tilde{Q}}[i]^2}\sqrt{\sum_{i=1}^{|\mathcal{D}|} V_X[i]^2}}$$

So, we can additionally define the similarity between an approximate query $\tilde{Q}$ and a set of context χ as follows:

$$\mathrm{sim}(\tilde{Q}, \chi) = \sum_{X \in \chi} \mathrm{sim}(\tilde{Q}, X)$$

After this, the approximate query results ranking problem is fully defined. According to [11], this problem is NP-hard.

4.3 Approach

Given Problem 1 is NP-hard, we have to think of approximation algorithms for solving it. In this paper, we adopt the idea of solution presented in [1] and propose the improved algorithms for solving it. The solution consists of three processing steps: create orders of tuples, find representative orders and rank the top-k answer tuples. The first and second steps are processed during offline time and the third step is processed during online time.

Step 1 (create orders): For each preference class P_{X_i} create a order τ_i of the tuples in R such that

$$\tau_i = \arg\max_{\tau_i'} \mathrm{Agree}(\tau_i', P_{X_i}) \tag{9}$$

The output of this step is a set of m ⟨context, order⟩ pairs of the form $\langle X_i, \tau_i \rangle$, where X_i is the context for the preferences in P_{X_i} and τ_i is the order of the tuples in R that (approximately) satisfies Equation (9). According to the output order of tuples, each tuple t has a score that is associated with the position of t in each order τ_i. The score of tuple t in τ_i that corresponds to X_i is: $s(t \mid X_i) = n - \tau_i(t) + 1$, where τ_i (t) represents the position of tuple t in order τ_i.

Step 2 (find representative orders): In order to reduce the number of ⟨context, order⟩ pairs, we need to find representative orders $\bar{\tau}_1, \ldots, \bar{\tau}_l$ for m initial pairs $\langle X_i, \bar{\tau}_i \rangle$, where, $l < m$. These orders partition the space of the m initial ⟨context, order⟩ pairs into l groups. Each group i is characterized by order $\bar{\tau}_i$ and a disjunction of contexts $\bar{X}_i \subseteq \{X_1, \ldots, X_m\}$ such that for each $X_j \in \bar{X}_i$ order $\bar{\tau}_i$ is a *representative order* for the initial order τ_j. Finding representative orders can be considered as an orders clustering problem. The score of tuple t in $\langle \bar{X}_i, \bar{\tau}_i \rangle$ is given by:

$$s(t \mid \bar{X}_i) = n - \bar{\tau}_i(t) + 1 \tag{10}$$

where $\bar{\tau}_i$ (t) represents the position of tuple t in order $\bar{\tau}_i$.

Given the offline computations, the only online task is to appropriately combine the priori formed rankings of the tuples to return a ranked answer to the given approximate query.

Step 3 (**rank the top-k answer tuples**) For an approximate query $\tilde{Q}$ over relation R, using the output of step 2, compute the set $\tilde{Q}_k(R) \subseteq \tilde{Q}(R) \subseteq R$ with $|\tilde{Q}_k(R)| = k$, such that $\forall t \in \tilde{Q}_k(R)$ and $t' \in \{R - \tilde{Q}_k(R)\}$ it holds that score(t, $\tilde{Q}$) > score(t', $\tilde{Q}$), with score(t, $\tilde{Q}$) = $\sum_{\bar{X}_i} \mathrm{sim}(\tilde{Q}, \bar{X}_i) \cdot s(t \mid \bar{X}_i)$.

5. ALGORITHM

In this section, we discuss the complexity of the problem in each step and present the algorithm for solving it.

5.1 Orders Creating Algorithm

Maximum Acyclic Sub-graph problem is known to be NP-Hard. We describe it in brief as follows: for an input directed weighted graph G find the maximum-weight sub-graph of G that is acyclic. From this, the orders creating problem is as hard as the Maximum Acyclic Sub-graph problem, and the connection between the Maximum Acyclic Sub-graph and the orders creating problem becomes intuitively clear via the X-preference graph.

We next give an algorithm for the orders creating problem. The Greedy algorithm for creating orders (Algorithm 2) is an adaptation of the algorithm proposed in [1] and [11]. The algorithm is operated on the X-preference graph. At every step a greedy selection is made.

Algorithm 2 The Greedy algorithm for creating orders

Input: Relation $R = \{t_1,\ldots, t_n\}$, a set of preferences from a single class P_X .

Output: A pair $\langle X, \tau \rangle$ where is an order of the tuples in R such that as many of the preferences in P_X are satisfied.

```
1. S = {$t_1,\ldots,t_n$}
2. rank = 0
3. For all $i \in \{1,\ldots, n\}$ Do
4.     $p(t_i) = \sum_{j=1}^{n} w_X(t_i \to t_j)$
5. End For
6. While S ≠ ∅ Do
7.     rank + = 1
8.     $t_v = \arg\max_{t_u \in S} p(t_u)$
9.     $\tau(t_v)$ = rank
10.    S = S – {$t_v$}
11.    For all $t \in$ S Do
12.        $p(t) = p(t) - w_X(t \to t_v)$
13.    End For
14. End While
```

5.2 Orders Clustering Algorithm

5.2.1 Orders Clustering Problem

In order to quantify how well an order τ of the tuples in R is represented by another order ρ we need to define a distance measure between orders over the same set of tuples. The well-known *Euclidean distance* is employed in our paper:

$$\mathrm{d_E}(\rho,\tau) = (\sum_{i=1}^{n}(\tau_i - \rho_i)^2)^{\frac{1}{2}}$$

In *Euclidean distance*, $\mathrm{d_E}$ satisfies the triangle inequality. That is, if τ_1 , τ_2 and τ_3 are any three permutations over a set of n objects. Then, the following inequality is true:

$$\mathrm{d_E}(\tau_1, \tau_2) + \mathrm{d_E}(\tau_2, \tau_3) \le \mathrm{d_E}(\tau_1, \tau_3)$$

Based on the Euclidean distance, we can reformulate the orders clustering problem in step 2: assume an input consisting of m context-order pairs $\langle X_i, \tau_i \rangle$ and let T_m be the set of the m orders over the tuples of relation R: $T_m = \{\tau_1,\ldots, \tau_m\}$. Find a set of $l < m$ orders $T_l = \{\bar{\tau}_1,\ldots, \bar{\tau}_l\}$ such that:

$$\mathrm{cost}(T_l) = \sum_{\tau \in T_m} \mathrm{d}(\tau, T_l) \qquad (11)$$

is minimized. The distance of a single order τ from a set of orders T is defined as $\mathrm{d}(\tau,T) = \min_{\rho \in T} \mathrm{d}(\tau,\rho)$.

We call the orders in set T_l representative orders and associate with each representative order $\bar{\tau}_i$ a set of contexts

$$\bar{X}_j = \{X_i \mid \bar{\tau}_j = \arg\min_{j'} \mathrm{d}(\tau_i, \bar{\tau}_{j'})\} .$$

As we know, the *k*-median problem is to be NP-hard and the orders clustering problem can be treated as the *k*-median problem [7]. We next propose an algorithm for dealing with the orders clustering problem.

5.2.2 Algorithm

Observing the solution of the orders clustering, we can find that every representative order connects with some other orders of T_m and these connections are like star structures. Here, we call a connection as a Star. Then we can re-define the orders clustering problem as follows: Let U be the set of all Stars, i.e., $U = \{\langle \tau_i, \bar{T}_i \rangle \mid \tau_i \in T_m, \bar{T}_i \subseteq T_m\}$. The cost of each *Star* s = $\langle \tau_i, \bar{T}_i \rangle \in U$ can be denoted as: $\mathrm{c_s} = \sum_{\tau_j \in \bar{T}_i} \mathrm{d_E}(\tau_i, \tau_j)$. Let $\mathrm{r_s} = \mathrm{c_s} / |\bar{T}_i|$ be the performance-price ratio. Our objective is to find a set of *Star S*, such that $S \subseteq U$, which minimizes the cost and enables that there are l representative orders in S and any initial order $\tau_j \in T_m$ appears at least one time at *Star* s $\in S$.

For solving this problem, our approach consists of two steps, which are a pre-processing step and a processing step. In the pre-processing step, we build a sequential permutation $l_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$ over T_m for each order $\tau_i \in T_m$, where the orders in l_i are arranged non-decreasing according to their cost corresponding to τ_i, that is, $\mathrm{d_E}(\tau_i, \tau_{i1}) \le \mathrm{d_E}(\tau_i, \tau_{i2}) \le \ldots \le \mathrm{d_E}(\tau_i, \tau_{im})$. Such permutations can help the algorithm to find the near-globally optimal solution. Note that, the time complexity of the pre-processing phrase is $O(m^2 \log m)$, where $|T_m| = m$.

The task of processing step is to find the l representative orders by using the Greedy refinement algorithm (Algorithm 3) based on the Stars formed in pre-processing step. The time complexity of the processing phrase is $O(ml)$.

Algorithm 3 The Greedy-RF algorithm for clustering orders

Input: $T_m = \{\tau_1,\ldots, \tau_m\}$, $U = \{\langle \tau_i, \bar{T}_i \rangle \mid \tau_i \in T_m, \bar{T}_i \subseteq T_m\}$, l

Output: A set of representative l orders $T_l = \{\langle \bar{\tau}_1, \bar{T}_1 \rangle \ldots \langle \bar{\tau}_l, \bar{T}_l \rangle\}$

```
1. Let $B$ = {} be a buffer that can hold $m$ $\langle \tau_i, \bar{T}_i \rangle$ pairs
2. While $T_m \neq \emptyset$ and $l > 0$ Do
3.     $B \leftarrow \emptyset$
4.     For each $\tau_i \in T_m$ Do
5.         Pick $s_i = \langle \tau_i, \bar{T}_i \rangle$ with minimum $r_s$ from $U_i = \{\langle \tau_i, \bar{T}_i \rangle \mid \bar{T}_i \subseteq T_m, |\bar{T}_i| = [2, |T_m| - l + 1]\}$
6.         $B \leftarrow B + \{s_i\}$
7.     End For
8.     Pick $s = \langle \tau_i, \bar{T}_i \rangle$ with minimum $r_s$ from $B$
9.     $T_m \leftarrow T_m - \bar{T}_i - \{\tau_i\}$, $T_l \leftarrow T_l + s$, $l \leftarrow l - 1$
10. End While
11. Return $T$
```

5.3 Top-k Ranking Algorithm

We describe a solution of ranking top-k answers problem, which employs the computations made in the offline steps, to provide ranked top-k answers for an approximate query. As discussed above, there are l different orders of all the tuples of relation R. Each order $\bar{\tau}_i$ is associated with a set of contexts $\bar{X}_i \subseteq \{X_1,\ldots,X_m\}$ forming l pairs $\langle \bar{X}_i, \bar{\tau}_i \rangle$. Each tuple $t \in R$ in each such pair i is associated with the $s(t \mid \bar{X}_i)$ as defined in Equation (10). We adapt Fagin's *Threshold Algorithm* [13] to retrieve the top-k answers. The top-k algorithm (Algorithm 4) works as follows.

Algorithm 4 The top-k ranking algorithm

```
Input: Representative orders $T_l = \{\bar{\tau}_1, \ldots, \bar{\tau}_l\}$, associated
       contexts $\bar{X}_i \subseteq \{X_1, \ldots, X_m\}$, an approximate query $\tilde{Q}$.
Output: top-k answer tuples.
1. Let $B = \{\}$ be a buffer that can hold $k$ tuples ordered by score
2. Let $L$ be an array of size $l$ storing the last score from each order
3. Repeat
4.   For all $i \in \{1, \ldots, l\}$ Do
5.     Retrieve next tuple $t$ from $\bar{\tau}_i$
6.     Compute $\text{score}(t, \tilde{Q}) = \text{sim}(\tilde{Q}, \bar{X}_i) \cdot s(t \mid \bar{X}_i)$ as the score of $t$
7.     Update $L$ with score of $t$ in $\bar{\tau}_i$
8.     If $t \in \tilde{Q}(R)$
9.       Get score of $t$ from other orders $\{\bar{\tau}_i \mid \bar{\tau}_j \in T_l$ and $j \neq i\}$
         via random access
10.      score($t$, $\tilde{Q}$) $\leftarrow$ summing up of all the retrieved scores
11.      Insert $\langle t, \text{score}(t, \tilde{Q})\rangle$ in the correct position in $B$
12.    End If
13.  End For
14. Until $B[K].\text{score} \geq \sum_{i=1}^{l} L[i]$
15. Return $B$
```

6. EXPERIMENTS

6.1 Experimental Setup

For our evaluation, we set up a used car database CarDB (Make, Model, Year, Color, Engine, Price, Mileage) containing 100,000 tuples extracted from Yahoo! Autos. The attributes Make, Model, Year, Color and Engine are categorical attributes and the attributes Price and Mileage are numerical attributes. We used Microsoft SQL Server 2005 RDBMS on a P4 3.2-GHz PC with 1 GB of RAM for our experiments. We implemented all algorithms in C# and connected to the RDBMS through ADO.

We employed the approach discussed in Section 4.1 to generate preferences from the workload of internet used car database. Using a confidence level of 0.2, we obtained 529 different classes of preferences, which are used in our experiments.

6.2 Relaxation and Ranking Experiments

To verify the efficiency of the query relaxation and results ranking methods of AQRR, we requested 5 subjects to behave as different kinds of buyers, such as rich people, clerks, students, women, etc. and each subject was asked to submit three queries for CarDB according to their preference. Each query had on average 2.8 specified attributes for CarDB.

Since it is not practical to ask the people to find all relevant answers in the database and rank the whole query results for a given approximate query, we adopt the following strategy. For each test query Q_i, a set H_i of 30 tuples, which likely to contain a good mix of relevant and irrelevant tuples to the query, is generated (We did this by mixing the Top-10 results of each ranking algorithm of AQRR, AIMQ and RANDOM, removing ties, and adding a few randomly selected tuples. These ranking algorithms will be described in the following.). Finally, we presented the queries along with their corresponding H_i's to each user in our study. Each subject's responsibility was to mark the tuples in H_i relevant to the query Q_i and mark the Top 10 tuples that they preferred most. We then measure the efficiency of query relaxation and results ranking methods of AQRR.

6.2.1 *Query Relaxation Experiment*

We use the Recall metrics to evaluate the efficiency of our query relaxation method. Recall is the ratio of the number of relevant tuples retrieved to the total number of relevant tuples. To evaluate the Recall of answers we provide, we also set up another approximate query answering system that uses the AIMQ algorithm proposed in [16], since AIMQ addresses the similar problem (answering imprecise queries over autonomous database) as AQRR does. AIMQ makes use of the AFDs (approximate function dependencies) and approximate keys to decide the attribute relaxation order, and then AIMQ relaxes the original query according to the relaxation order and the threshold for the query. The Recall of answers for AQRR and AIMQ is shown in Figure 2 (where T_{sim}=0.7 for both AQRR and AIMQ).

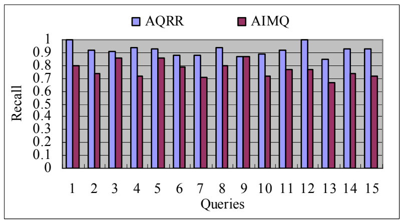

Figure 2. Recall of answers for AQRR and AIMQ

It can be seen that the Recall of AQRR consistently higher than AIMQ. The average Recall of AQRR is 0.92 while the AIMQ is 0.77. This is because AIMQ learns the attribute importance based on some pre-extracted data and the importance of attributes are invariant to the different user queries. It is also incapable of giving a specific weight to show how important each attribute is. In contrast, AQRR speculates how much the user cares about each specified attribute according to the user's query and assigns a specific weight to each specified attribute. Thus, the attribute importance can be tailored to the user preferences. Additionally, the similarities between different attribute values are reasonable. Hence, the result tuples for the approximate query can meet the user's needs and preferences more closely.

6.2.2 *Ranking Experiment*

This experiment aims at evaluating the ranking precision of AQRR. Besides AQRR described above, we implemented RANDOM and AIMQ algorithms, to compare with AQRR.

RANDOM ranking model: In the RANDOM ranking model, the tuples in the query results are presented to the user in a random order. The RANDOM model provides a base line to show how well AQRR can capture the user behaviour over a random method.

AIMQ ranking model: AIMQ measures the similarity between an imprecise query Q and an answer tuple t as

$$\text{Sim}(Q,t) = \sum_{i=1}^{n} W_{imp}(A_i) \times \begin{cases} \text{VSim}(Q.A_i, t.A_i), & \text{if Domain}(A_i)\text{=Categorical} \\ 1 - \dfrac{Q.A_i, t.A_i}{Q.A_i}, & \text{if Domain}(A_i)\text{=Numerical} \end{cases}$$

where n = Count(boundattributes(Q)), W_{imp}($\sum_{i=1}^{n} W_{imp} = 1$) is the importance weight of each attribute, and VSim measures the similarity between the categorical values. The similarities between the Q and answer tuples are used to rank the relevant query results.

For formally comparing the ranking precision of the various ranking functions, we used a standard collaborative filtering metric *R* proposed in paper [3] to measure ranking quality (Equation (12)). In the equation, r_i is the subject's preference for the *i*th tuple in the ranked list returned by the ranking function (1 if it is marked relevant, and 0 otherwise).

$$R = \sum_i \frac{r_i}{2^{\left(\frac{i-1}{9}\right)}} \tag{12}$$

In order to make sure that the same results without the same order will be retrieved by using each ranking algorithm, we use the same relaxed queries (where T_{sim} = 0.7 for each original query) as the input for RANDOM, AIMQ and AQRR. Figure 3 shows the ranking precision of different ranking algorithms for each query.

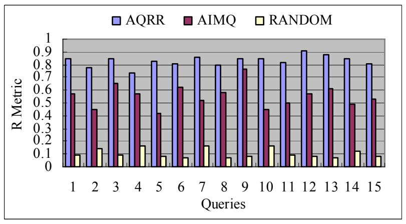

Figure 3. Precision for different ranking methods for CarDB

It can be seen that AQRR greatly outperform AIMQ and RANDOM. The average ranking precision of AQRR and AIMQ were 0.83 and 0.55, respectively. The reason is that AIMQ can only rank the relevant answers of the relaxed query by using the similarities while the exact answers are ranked randomly. In contrast, AQRR can rank both the exact answers and relevant answers according to the user's needs and preferences. Moreover, the ranking method of AQRR takes both the similarities between tuples and the query and the user's preferences into consideration, so that the ranking results can capture the user's needs and preferences more efficiently. Also, our method computes the similarity of numerical attribute values factoring in the distribution of attribute values while AIMQ only computed the distance of two numerical attribute values.

6.3 Orders Creating&Clustering Experiments

6.3.1 Orders Creating Experiment

This experiment aims at evaluating the accuracy of the orders of tuples created by Greedy algorithm based on the contextual preferences with interest degrees (henceforth referred to as CPDG algorithm). In [1], Agrawal presents a contextual preference model of the form "$A_1 = a_1 \succ A_1 = a_2 \mid X$", while our contextual preference model has the form "$A_1 = a_1 \succ A_1 = a_2$, d $\mid X$". Obviously, the critical difference between the two preference models is that our contextual preference model involves the interest degree of the preference while the contextual preference model in [1] does not. We also build system by employing the Greedy algorithm to create orders based on the contextual preferences without interest degrees (henceforth referred to as CPG algorithm). Next, we conduct an experiment to compare the accuracy of orders created by CPDG and CPG, respectively. We collect 50 tuples asserted by 100 preferences in the same preference class. We then use the metric *P* (Equation (13)) to evaluate the accuracy of orders

$$P = \text{Order (A)} \cap \Gamma \,/\, |\,T\,| \tag{13}$$

where A = {CPDG, CPG}, Γ represents the correct order of test tuples ranked according to the input preferences, Order(A) represents the order of tuples created by algorithm A. Order(A) ∩ Γ represents the number of tuples locating in the same position in the orders of Order(A) and Γ, respectively; $|T|$ is the total number of test tuples. Figure 4 shows the accuracy of the orders created by CPDG and CPG, resp.

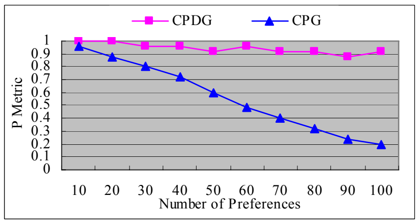

Figure 4. Accuracy of orders for CPDG and CPG

It can be seen that the accuracy of the order created by CPDG is steadily better than the order created by CPG. Additionally, the accuracy of the order created by CPG degrades as the amount of preferences increases. This is because the contextual preference model does not consider the interest degree of preference so that it can not reconcile the contradictions of too many preferences.

6.3.2 Orders Clustering Experiment

This experiment aims at testing the quality of the algorithm for the orders clustering. For this experiment we assume there are no indifferent tuples. Every dataset is characterized by 4 parameters: *n*, *m*, *l*, *noise*. Here *n* is the number of tuples in each of the orders, *m* is the number of input orders, and *l* is the number of true underlying clusters. We generate *l* random orders by sampling uniformly at random the space of all possible permutations of *n* elements. These initial orders form the centers around which we build each one of the clusters. The task of the algorithms is to rediscover the clustering model used for the data generation. Given a cluster center, each order from the same cluster is generated by adding to the center a specified amount of *noise* of *swaps*. The *swap* means that tuples from the initial order are picked and their positions in the order are exchanged. The amount of noise is the number of swaps we make.

We experiment with datasets generated for the following parameters: *n* = 300, *m* = 600, *l* = {8, 16}, *noise* = {2, 4, 8,…,128} for swaps. Figure 5 shows the performance of the algorithms as a function of the amount of noise. The *y* axis is ratio: F(A)/F(INP), for A = {Greedy, Furthest, Greedy-RF}, where the Greedy-RF algorithm is proposed in Algorithm 3, while the Greedy and Furthest algorithm are presented in [1]. We compare them here since they all aim at solving the orders clustering problem. The F(A) is the total cost of the solution provided by algorithm A when Euclidean distance is used as a distance measure between orders. The F(INP) corresponds to the cost of the clustering structure (Equation(11)) used in the data generation process.

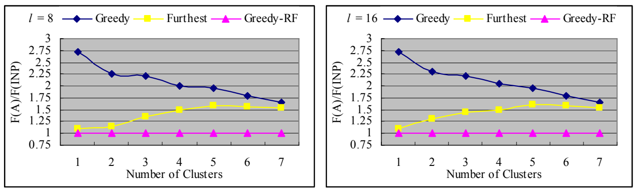

Figure 5. Algorithms' performance for orders clustering

From Figure 5 we can see that: Greedy-RF algorithm performs greatly better than Greedy algorithm and slightly better than Furthest algorithm. The reason is that: the Greedy-RF is executed on the orders which were arranged according to their cost in pre-processing step and makes twice greedy selection in processing step, so that it can obtain the near-globally optimization solution.

6.4 Performance Report

Figure 6 shows the online execution time of the queries over CarDB as a function of the number of representative orders. It can be seen that the execution time of AQRR grows almost linearly with the number of representative orders. This is because most of the computations have been accomplished at pre-processing time.

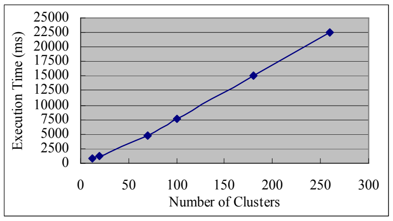

Figure 6. Execution times for different numbers of representative orders.

7. CONCLUSIONS

In this paper we first motivated the need for supporting approximate queries & results ranking over databases in a domain-independent way. Then we presented AQRR, a domain independent approach for answering approximate queries over autonomous Web databases. Starting from the user query, AQRR assigned the weights of specified attributes according to the distribution of values of specified attributes in the database. Then, according to the similarities of different attribute values, AQRR relaxed the original query by adding the most similar categorical values or nearby numerical values into the query criteria range. For ranking the approximate query results, AQRR took advantage of the user's contextual preferences to pre-compute a few representative orders of tuples and used them to quickly provide the ranked query results. The experiments on real dataset identified that the query relaxation method of AQRR can find more relevant tuples for the user. Rather, the top-k ranked answers can be returned fast and achieve high accuracy as well. It would be interesting to investigate how to minimize the updating cost when the database and preferences are varied.

8. ACKNOWLEDGMENTS

Work is supported by the National Natural Science Foundation of China (60873010) and Program for New Century Excellent Talents in University (NCET-05-0288).

9. REFERENCES

[1] Agrawal, R. and Rantzau, R. Context-sensitive ranking. In Proceedings of the SIGMOD Conference, 383-394, 2006.

[2] Agrawal, R., Imielinski, T., and Swami, A. N. Mining association rules between sets of items in large databases. In Proceedings of the SIGMOD Conference, 207-216. 1993.

[3] Agrawal, S., Chaudhuri, S., Das, G., and Gionis, A. Automated ranking of database query results. ACM Trans. Database Syst., 28(2): 140-174, 2003.

[4] Bosc, P., Hadjali, A., and Pivert, O. Empty versus overabundant answers to flexible relational queries. Fuzzy Sets and Systems, 159, 1450-1467, 2007.

[5] Chen, Z. Y. and Li, T. Addressing diverse user preferences in SQL-Query-Result navigation. In Proceedings of the SIGMOD Conference, 641-652, 2007.

[6] Chakrabarti, K., Ganti, V., Han, J., and Xin, D. Ranking objects based on relationships. In Proceedings of the SIGMOD Conference, 371–382, 2006.

[7] Chrobak, M., Keynon, C., and Young, N. The reverse greedy algorithm for the metric k-median problem. Information Processing Letters 97, 68-72, 2005.

[8] Chaudhuri, S., Das, G., and Hristidis, V. Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst., 31(3):1134–1168, 2006.

[9] Chakrabarti, K., Chaudhuri, S., and Hwang, S. Automatic categorization of query results. In Proceedings of the SIGMOD Conference, 755–766, 2004.

[10] Chomicki, J. Preference formulas in relational queries. ACM Trans. Database Syst., 28(4): 427-466, 2003.

[11] Cohen, W. W., Schapire, R. E. Learning to order things. Journal of Artificial Intelligence Research, 10, 243–270, 1999.

[12] Das, G., Hristidis, V., Kapoor, N., and Sudarshan, S. Ordering the attributes of query results. In Proceedings of the SIGMOD Conference, 395-406, 2006.

[13] Fagin, R., Lotem, A., and Naor, M. Optimal aggregation algorithms for middleware. In Proceedings of the PODS Conference, 102-113, 2001.

[14] Kieβling, W. Foundations of preferences in database systems. In Proceedings of the VLDB Conference, 311-322, 2002.

[15] Ma, Z. M. and Yan, L. Generalization of strategies for fuzzy query translation in classical relational databases. Information and Software Technology, 49(2):172-180, 2007.

[16] Nambiar, U. and Kambhampati, S. Answering imprecise queries over web databases. In Proceedings of the ICDE Conference, 45-54, 2006.

[17] Ortega, B. M., Chakrabarti, K., and Mehrotra, S. An approach to integrating query refinement in SQL. In Proceedings of the EDBT Conference, 15-33, 2002.

[18] Ortega, B. M. Integrating similarity based retrieval and query refinement in databases. Ph.D Dissertation, UIUC, 2003.

[19] Muslea, I. Machine learning for online query relaxation. In Proceedings of the KDD Conference, 246-255, 2004.

[20] Rui, Y., Huang, T. S., and Merhotra, S. Content-based image retrieval with relevance feedback in MARS. In Proceedings of the ICIP Conference, 815-818, 1997.

[21] Stefanidis, K. and Pitoura, E. Adding context to preferences. In Proceedings of the ICDE Conference, 846-855, 2007.

[22] Su, W., Wang, J., Huang, Q., and Lochovsky, F. Query result ranking over e-commerce web databases. In Proceedings of the CIKM Conference, 575-584, 2006.

[23] Wu, L., Faloutsos, C., Sycara, K., and Payne, T. FALCON: feedback adaptive loop for content-based retrieval. In Proceedings of the VLDB Conference, 297-306, 2000.