Addressing Diverse User Preferences: A Framework for Query Results Navigation

Zhiyuan Chen
University of Maryland, Baltimore County
zhchen@umbc.edu

Tao Li
Florida International University
taoli@cs.fiu.edu

Abstract

Database queries are often exploratory and users often find their queries return too many answers, many of them irrelevant. Existing approaches include categorization, ranking, and query refinement. The success of all these approaches depends on the utilization of user preferences. However, most existing work assumes that all users have the same user preferences, but in real life different users often have different preferences. In this paper, we propose a framework that addresses diverse user preferences in query results navigation.

1 Introduction

Database queries are often exploratory and users often find their queries return too many answers, many of them irrelevant. Three approaches have been proposed to solve this problem. The first approach [4] categorizes query results into a *navigational tree*. The second approach [2, 5, 1] ranks the results. The third approach refines queries based on user feedbacks [8]. The success of all three approaches depends on the utilization of user preferences. However, most existing work assumes that all users have the same user preferences, but in real life different users often have different preferences. In this paper, we propose a framework that addresses diverse user preferences in query results navigation. This framework uses a tree-based method and a cluster-based method to help user navigation. The tree-based method was originally proposed in [6]. Next we describe these methods in Section 2 and Section 3. Section 4 describes some future research directions.

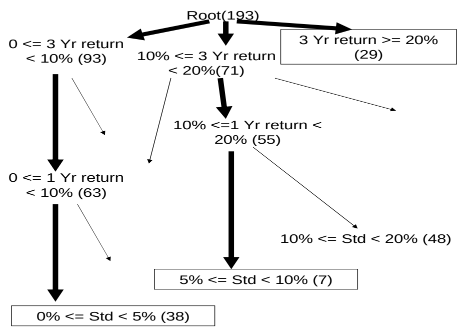
2 Tree-based Method

We first describe a motivating example in Section 2.1. We then describe the tree based method in Section 2.2.

2.1 A Motivating Example

Example 1. Consider a mutual fund selection website. Figure 1 shows a fraction of a navigational tree generated using a method proposed in [4] over 193 mutual funds returned by a query with the condition fund_name like '%Vanguard%'. Each tree node specifies the range or equality conditions on an attribute, and the number in the

Copyright 2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering



Root(193)
0 <= 3 Yr return < 10% (93)
10% <= 3 Yr return < 20%(71)
3 Yr return >= 20% (29)
10% <=1 Yr return < 20% (55)
0 <= 1 Yr return < 10% (63)
10% <= Std < 20% (48)
5% <= Std < 10% (7)
0% <= Std < 5% (38)

Figure 1: Tree generated by the existing method [4].

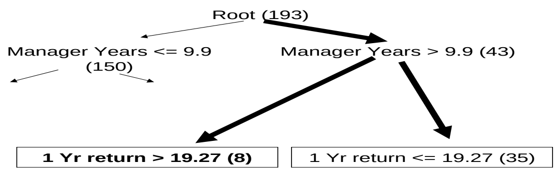


Figure 2: Tree generated by our tree-based method.

parentheses is the number of data records satisfying all conditions from the root to the current node. Users can use this tree to select the funds that are interesting to them. For example, a user interested in funds with very high returns may select those funds with "3 Yr return" over 20% (the right most node in the first level). Now only 29 records instead of 193 records need to be examined in the query results.

Consider four users U1, U2, U3, and U4. U1 and U2 prefer funds with high returns and are willing to take higher risks, U3 prefers funds with low risks and is willing to sacrifice some returns, and U4 prefers both high return funds and low risk funds (these two types of funds typically do not overlap). The existing method assumes that all users have the same preferences. Thus it places attributes "3 Year return" and "1 Year return" at the first two levels of the tree because more users are concerned with returns than risks. However, the attribute characterizing the risks of a fund is "standard deviation", and is placed at multiple nodes in the third level of the tree. Suppose U3 and U4 want to visit low risk funds at the two bottom-left nodes (bounded with rectangles) in Figure 1, they need to visit many nodes (including nodes on the paths from the root plus the sibling nodes because users need to check the labels of the siblings to decide which path to follow.).

2.2 Tree-based Navigation Method

User preferences are often difficult to obtain because users do not want to spend extra efforts to specify their preferences. Thus there are two major challenges to address the diversity issue of user preferences: (1) how to summarize diverse user preferences from the behavior of all users already in the system, and (2) how to decide the subset of user preferences associated with a specific user. As most existing work [4, 2, 5] did, we use query history to infer the behavior of all users in the system.

One well-known solution to the second challenge is to define a user profile for each user and use the profile to decide his/her preferences. However, in real life user profiles may not be available because users do not want to or can not specify their preferences (if they can, then they can form the appropriate query and there is no need for either ranking or categorization). One could try to derive a profile from the query history of a certain user, but this method will not work if the user is new to the system, which is exactly when the user needs help the most.

Clustering query history: We propose a two-step approach to address both challenges for the categorization case. The first step occurs offline. It analyzes query history of all users already in the system and generates a set of non overlapping clusters over the data, each corresponding to one type of user preferences. Each cluster has an associated probability of users being interested in that cluster. We assume that an individual user's preference can be represented as a subset of these clusters, and each cluster will appear in the subset with its associated probability. The system stores these clusters (by adding a class label to each data record) and probabilities for each cluster.

We propose two preprocessing steps to prune unimportant queries and merge similar queries. The algorithm is described in Figure 3. At step 1 and 2, the algorithm prunes unimportant queries and merges similar queries into a single query. At step 3 to 5, the clusters are generated.

```
Input:  query results $D_Q$, data $D$, and query history $H = \{(Q_1, U_1, F_1), \dots$
$(Q_k, U_k, F_K)\}$ where $Q_i$ is a query, $U_i$ is session ID, and $F_i$ is the weight of $Q_i$.
1.  $H'$=Prune($H$, $D$).
2.  $\{QC_1, \dots, QC_k\} = Merge(H')$ where $QC_i$ is a merged query.
3.  For each record $r_i \in D_Q$, identify $S_i = \{QC_p | \exists Q_j \in QC_p$
     such that $r_i$ is returned by $Q_j\}$.
4.  Group records in $D_Q$ by $S_i$.
5.  Output each group as a cluster $C_j$, assign a class label
     for each cluster, and compute probability $P_j = \frac{\sum_{Q_i \in S_j} F_i}{\sum_{Q_p \in H'} F_p}$.
```

Figure 3: The algorithm to generate preference-based clusters.

In Example 1, the first step of our approach generates three clusters, one for high return funds, one for low risk funds, and the third for the remaining funds (i.e., those no users will be interested in).

Building a navigational tree: The second step occurs online when a specific user asks a query. It first intersects the set of clusters generated in the first step with the answers of the query. It then automatically constructs a navigational tree over these intersected clusters on the fly. This tree is then presented to the user. The user first navigates this tree to select the subset of clusters matching his/her needs. The user can then browse, rank, or categorize the results in the selected clusters.

Note that here we do not intend to select the most interesting records for the user. Instead, it is up to the user to do so. We just provide a navigational tree that "best describes" the differences of different clusters such that the user can easily locate the subset of clusters matching his/her needs. This step also does not assume the availability of a user profile or a meaningful query.

The diversity issue is addressed in two aspects. First, the first step of our approach captures diverse user preferences by identifying all types of user preferences in the format of clusters. Second, the second step uses a navigational tree to let a specific user select the subset of clusters (or types of user preferences) matching his needs. Let k be the number of clusters, the user can potentially select up to 2^k different preferences (subsets). This gives users more flexibility than the profiling approach because in the latter case each user is forced to use only one profile.

The navigational tree is generated in a similar way as constructing a decision tree over the query results. The major difference is the splitting criteria because we want to minimize the cost of navigating this tree. The standard splitting criteria for decision tree does not consider the difference between visiting a leaf with many records and visiting a leaf with very few records, as long as both leaves have similar distribution of classes. We try to maximize the following splitting criteria:

$$\frac{IGain(t, t_1, t_2)/E(t)}{(\sum_{j=1,2} N(t_j)(\sum_{C_i \cap t_j \neq \emptyset} P_i))/(N(t)\sum_{C_l \cap t \neq \emptyset} P_l)}. \tag{1}$$

Here t is the current tree node that will be split into child nodes t_1 and t_2. $IGain(t, t_1, t_2)$ is the information gain of splitting t into t_1 and t_2. $E(t)$ is the entropy of t. $N(t_j)$ is the size of child t_j. P_i is the probability of cluster C_i, which is computed in the clustering step.

The split should reduce the impurity in each leaf such that each leaf should contain records mostly from the same cluster. This is measured by the the nominator, which is the information gain normalized by the entropy of t. The split should also reduce the cost of visiting the leaf records. This is measured by the denominator, which is the cost of visiting leaf records after split, normalized by the cost before split.

For example, a navigational tree generated by our tree-based method is shown in Figure 2. Attribute "manager years" is selected in the first level of the tree. The intuition is that funds managed by very senior managers often have either very high returns, or very low risks because otherwise those managers may get fired for their

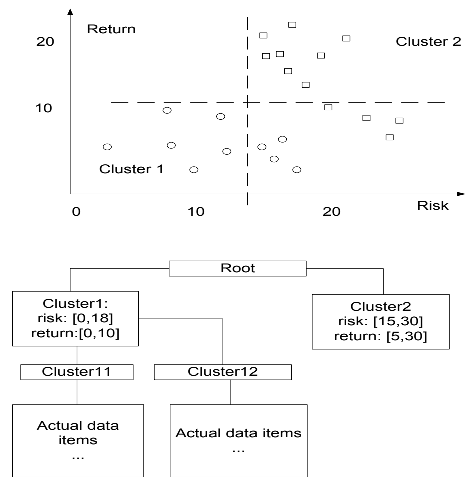


Figure 4: An example for the cluster-based method.

```
ExpandOneNode($A$, $Q$, $c$, $A_u$)
1.  sort $A - A_u$ in descending order
    of weights.
2.  $A_c$ = first $k$ attributes in $A - A_u$ that
    appear in $Q$.
3.  for each pair of attributes $(A_i, A_j)$ in $A_c$
4.      Generate clusters for records in $c$
        using $(A_i, A_j)$
5.      Compute cost(c) using equation (4)
6.  endfor
7.  select the pair of attributes
    $(A_{min_1}, A_{min_2})$ with the lowest cost(c)
8.  $A_u$ = $A_u \cup \{A_{min_1}, A_{min_2}\}$
9.  return $A_{min_1}$, $A_{min_2}$, and the set of
    clusters generated.
```

Figure 5: Algorithm to expand one node.

poor performance. “manager years” is not selected by the existing categorization method [4] because the most frequently used attribute is “3 Yr return” in the query history. Next “1 year return” is selected to separate the remaining two clusters. Now “high return” funds and “low risk” funds can be found at two leaf nodes with bounding rectangles. All four users only need to visit 4 tree nodes (2 in first level and 2 in the second level) excluding the root, while using the tree in Figure 1, U3 and U4 need to visit 12 nodes (3 in the first level, 5 in the second level, and 4 in the third level).

The results of experimental evaluation of the tree-based method can be found in [6]. The proposed method has been compared to a method without clustering query history (i.e., not considering diverse user preferences) and a method using standard decision tree splitting criteria (i.e., not considering the navigational cost). The results show that our method beats both alternatives in terms of navigational cost and user satisfaction.

3 Cluster-based Method

One major problem of the tree based approach is that it considers one attribute at a time. However, this is often inappropriate when the data contains correlated attributes. For example, the top half of Figure 4 shows the results of mutual funds where “risk” and “return“ are correlated. As a result, the results can be divided into two clusters. However, the boundary of clusters can not be easily specified using a categorization tree. For instance, if we consider dividing the data along “risk”=15, part of cluster 1 will be on the same side as cluster 2. Similarly, it is difficult to partition along “return”. Instead, we propose a cluster-based method to address this problem.

The proposed method uses a hierarchical navigational structure where each node in the hierarchy consists of a small number of clusters that have a concise description and can be easily visualized. For example, in the bottom half of Figure 4, we present to user a hierarchical structure where the first level contains two clusters: cluster 1 and cluster 2. Each cluster can be further divided into smaller clusters. The leaf level contains actual data records. To help users navigate the tree, each node also contains description of the cluster associated with this node. The description may include a representative record (e.g., using the centroid of the cluster), the ranges of some important attributes in the cluster, or a graphic visualization of clusters.

We propose a method to learn the distance function for clustering based on the correlation of query workload and data. We also propose a cost-based Greedy algorithm that constructs a hierarchy. Section 3.1 describes the learning algorithm. Section 3.2 describes the algorithm to construct the hierarchy. Section 3.3 reports the experimental evaluation of the cluster-based method.

3.1 Learning the Clustering Distance Function

In this section, we propose the algorithm to learn the distance function for clustering. In this paper we use weighted Minkowski distance function as follows.

$$d_w(A,B) = (\sum_{i=1}^{m} w_i |a_i - b_i|^p)^{\frac{1}{p}}. \tag{2}$$

Here A and B are two records, a_i or b_i is the i-th attribute of A or B, and w_i is the weight on the i-th attribute.

Our goal in the paper is to construct a cluster-based navigational hierarchy and we utilize the user preferences (in the form of query workloads) to dynamically assign weights to attributes. To automatically determine the weights for attributes, we use the metric learning approaches [9], which learn appropriate distance functions based on the correlations between attributes and workloads. We assume that when data records are similar, they tend to share similar preferences by many users, i.e., retrieved together by many queries.

Let n be the number of data records and m be the number of attributes. We can view the data as an $n \times m$ matrix R where the i-th row R_i is the i-th data record and $R_{ik}, 1 \leq k \leq m$, represents the k-th attribute of the i-th data record. Let q be the number of queries, the workload can also be represented as a $n \times q$ matrix Q where the i-th row Q_i indicates the user preferences for the i-th data records. Note that $Q_i = (Q_{i1}, \cdots, Q_{iq})$ where $Q_{i,v}, 1 \leq v \leq q$, is one if the i-th data record is selected by the v-th query and zero otherwise. In other words, Q is a binary matrix where each entry is either 1 or 0.

We use $d_R(i,j;\mathbf{w}) = \sqrt{\sum_{k=1}^{m} w_k (R_{i,k} - R_{j,k})^2}$ to denote the distance measurement among all pairs of records, based on the values of these records and parameterized by weights $\mathbf{w} = \{w_1, \cdots, w_m\}$. We assume values of numerical attributes have been normalized to [0,1]. If an attribute is categorical, we use as many binary attributes as the number of original attribute values. The value of a binary attribute corresponding to $\langle attribute_1, value_1 \rangle$ would be 1 if $attribute_1$ had $value_1$, and 0 otherwise. The binary attributes generated from the same categorical attribute have the same weight.

We use $d_Q(i,j) = \sqrt{\sum_{k=1}^{q} (Q_{i,k} - Q_{j,k})^2}$ to denote the distance measurement among all pairs of records, based on user preferences. Given that the user preferences are subjective judgments, a good weighting scheme for attributes should lead to a distance measurement that is consistent with the one based on user preferences. Thus to learn appropriate weights $\mathbf{w}$ for the attributes, we enforce the consistency between distance computation $d_R(i,j;\mathbf{w})$ and $d_Q(i,j)$, which leads to the following optimization problem:

$$\mathbf{w}^* = \arg\min \sum_{i \neq j} (d_R(i,j;\mathbf{w}) - d_R(i,j))^2 s.t. w_i \geq 0, w_i \in \mathbf{w} \tag{3}$$

We use the quadratic programming technique [3] to solve the above optimization problem. Given that user preferences can be noisy and may not accurately reflect the interests of individual users, we also introduce a regularization term as L_1 norm of weights, into the objective function in Equation (3) to eliminate small weights that are caused by accidental matches in workload. We also reduce the cost of quadratic programming by a condensation step. This step creates a large number of clusters (say 100) using an equal-weight distance function. Records in each cluster are condensed to the centroid of the cluster. Each cluster is then associated with the union of the sets of queries associated with each record in the cluster. We then run the quadratic programming algorithm on these centroid and their corresponding queries.

3.2 Hierarchy Construction Algorithm

In this section we propose a cost-based algorithm to construct the hierarchy. We first describe a cost estimation method, then propose an algorithm to select clustering attributes and generate clusters for a single node, and finally describe the complete algorithm.

Cost Estimation: We use the same model as in [4] to estimate the navigational cost in our hierarchy construction algorithm. Given a node v, let c be cluster associated with v and $c_i, 1 \leq i \leq m$ be the clusters associated with its children. Let $P(c_i)$ be the probability that users will visit c_i once they reach c, and $cost(c)$ be the cost for users to navigate the subtree rooted at v. Let K be the cost of checking the description of a cluster. The cost for the subtree rooted at v is given by the following formula:

$$Cost(c) = |c| * (1 - \sum_{i=1}^{m} P(c_i)) + \sum_{i=1}^{m} P(c_i) * (m * K + \sum_{i=1}^{m} P(c_i) * Cost(c_i)) \quad (4)$$

$\sum_{i=1}^{m} P(c_i)$ is the probability of visiting any of children of c. The cost is the sum of two terms. The first term computes the cost if users want to see all data records (the number of records is denoted as $|c|$) in c without visiting its children. The second term computes the cost of visiting its children and the subtree rooted at its children. Thus given $P(c_i)$ for every node in the hierarchy, we can recursively compute the total cost of navigating a hierarchy.

$P(c_i)$ can be estimated based on the query workload. Suppose there are q queries in the workload that overlap with cluster c (i.e., these queries select at least one record in c), and q_i queries in the workload that overlap with cluster c_i, then $P(c_i) = q_i/q$.

Cluster Generation for One Node Figure 5 shows the algorithm to expand a single node in the hierarchy. Here A is the set of attributes, Q is the query workload, c is the cluster for current node v, and A_u is the set of attributes that have been used to generate clusters associated with v and its ancestors.

The algorithm uses two heuristics to select clustering attributes: (1) those attributes with higher weights are important and shall be considered first; (2) attributes frequently appearing in queries are also important for users. To combine these two heuristics, we sort available attributes in $A - A_u$ in the descending order of weights (obtained using the method described in Section 3.1) and select the first k attributes that also appear in the query workload. We use $k = 3$ in this paper because we find it works quite well in experiments. In line 3 to 6, we consider all possible pairs of these k attributes to generate several clustering schemes, and estimate the cost for each clustering scheme. Line 7 selects the clustering scheme with the lowest cost.

We use the K-Means algorithm for clustering. K-Means requires us specify the number of clusters. We run K-Means with several numbers (3, 4, and 5 in this paper), and select the one with the minimal cost. This number can also be set by users and we find 3-5 work quite well in experiments. Finally, line 8 adds selected attributes to the set of used attributes A_u, and returns the selected attributes and generated clusters.

Hierarchy Generation: We propose a Greedy algorithm to generate the hierarchy. The algorithm first learns weights of attributes. We only consider attributes in workload. The algorithm then starts from the root, and repeatedly calls ExpandOneNode to expand the leaves in the partially constructed hierarchy if there are more than t records in the node. t is a parameter that can be set by users. We use $t = 10$ in our experiments.

ExpandOneNode considers $k(k-1)/2$ pairs of attributes. The clustering algorithm will be called 3 times for each pair (since we run K-Means for 3, 4, 5 clusters). We use $k = 3$ in this paper so K-Means is called 9 times per node. The total cost of the algorithm depends on the number of internal nodes in the hierarchy (leaf nodes do not need to be clustered). Each node can have at most 5 children (since we generate at most 5 clusters). Let q be the number of attributes in Q. The maximal height of the tree is $q/2$ because we do not use the same attribute twice in any root-to-leaf path. Thus the cost of the algorithm is $O(5^{q/2-1}9C)$ where C is the cost of one execution of K-Means. In practice, we can select only those attributes frequently appearing in workload. Thus q is usually small.

3.3 Experimental Evaluation

In this section we present the results of an empirical study.

Experimental Setup: We conducted our experiments on a machine with Intel PIII 996MHZ CPU, 512M RAM, and running Windows XP Professional version 2002.

Table 1: Navigational Cost Per Fund

User	Greedy	Non-Clustering	UPGMA-All	UPGMA-Q	Random-All	Random-Q	Freq-Q
User1	6.66	15.83	9.33	11.33	14.5	9.5	9.83
User2	12	22.5	29.83	23.83	22.5	21.16	14.6
User3	7.75	13.5	10.25	12.75	21.75	11.25	10.5
User4	10.25	13.25	11	12	13.5	12.5	14
User5	7.5	12	20	15	23.5	19	15
User6	8.5	17.16	18.5	15.66	19.5	15.2	18.2
User7	14.4	24	15.2	16.6	25.6	20.2	19.2
Average	9.75	17.69	16.57	15.66	19.84	15.42	14.54

We used a dataset of 18537 Mutual funds downloaded from *www.scottrade.com*. There are 33 attributes, 28 with numerical values and 5 with categorical values.

We asked 7 users to ask queries against the dataset. Each query had 0 to 6 range conditions. We collected altogether 62 queries. There were six attributes in search conditions of these queries and each query had 4.4 conditions on average.

We implemented the proposed Greedy algorithm using JAVA. We compare the Greedy algorithm with 6 other algorithms: (1) Non-Clustering: the non-clustering based algorithm proposed in [4]. (2) UPGMA-all: this is a well known hierarchical clustering algorithm [7]. It first uses K-Means to create clusters, using the same weight over all attributes. It then generates a hierarchy by repeatedly merging clusters with the minimal inter-cluster distances [7]. (3) UPGMA-Q: this is the same as UPGMA but using only attributes in query workload, and still using the same weight on those attributes. (4) Random-all: this algorithm creates the hierarchy by using 2 attributes for clustering at each node. However, the clustering attributes are randomly selected from all attributes in the data and each clustering attribute has the same weight. (5) Random-Q: same as Random-all but only selects attributes in the query workload. (6) Freq-Q: same as Random-all but uses a heuristics to choose clustering attributes. Attributes in the workload are sorted in descending order of their frequencies in the workload, and the first pair is used for clustering data in the first level of the hierarchy, and the second pair is used for the second level, and so forth.

We use the average navigational cost to measure the quality of hierarchies. The same set of users were asked to provide us several mutual funds that they found worth investing. Each user selected 2-6 funds and 33 funds were selected in total. For each user, we computed the total cost to retrieve these funds. We then divided this cost by the number of funds that the user selected.

Results: We have run all algorithms on the results of various SQL queries. Due to space limit, we only report the results of the following SQL query: "select * from AllFunds where FundFamily = 'Vanguard' ". The running time of all algorithms is well under one second. Table 1 reports the average navigational cost for each user. The results for Random-all and Random-Q are the average of 5 runs.

Greedy is the best among all algorithms for all users, and is on average about 103%, 58%, and 49% better than Random-All, Random-Q, Freq-Q, respectively. The average cost of Greedy is about 81% smaller than the cost for Non-Clustering. Non-Clustering algorithm generates very deep trees because it ignores the correlations in data. The results also show that Greedy is on average over 60% better than the two conventional hierarchical clustering algorithms UPGMA-All and UPGMA-Q. Furthermore, the hierarchy generated by hierarchical clustering algorithms is difficult for users to navigate because clusters in the hierarchy are generated over many attributes (33 for UPGMA-All and 6 for UPGMA-Q), making it difficult for users to visualize.

Interestingly, the performance of Freq-Q is only slightly better than Random-Q. The reason is that simply looking at the frequencies of attributes in the workload may not give the correct order of importance. For example, '1 year total return' and '3 year total return' are the most frequent attributes in the workload. However, these two attributes are highly correlated in data. Thus clustering using these two attributes do not provide more information than clustering using just one of them.

Our distance learning algorithm is based on the assumption that when data records are similar, they tend to share similar preferences by many users, i.e., retrieved together by many queries. The experimental results verify this assumption. For example, we found that most users were aware of the correlation between attributes such as “1 year return” and “3 year return”, so they often included only one of these two attributes in the queries along with other attributes such as ‘risk’ and ‘minimal investment’. Thus the proposed distance learning algorithm gives higher weights to ‘1 year total return’, ‘risk’, and ‘minimal investment’, leading to clusters that give more information for users.

4 Outlook

In this paper, we show how to address diverse user preferences when helping users navigate SQL query results. There are many directions for future research. First, it will be interesting to study how to address diverse user preferences for the two other approaches: ranking and query refinement. Further, most research efforts so far focus on reducing the amount of information presented to user (e.g., through clustering, grouping, or ranking). Another interesting direction is to study how to better visualize the results. More rigorous usability study is also needed in this field. So far, most research work only uses 10-20 participants and reports the degree of satisfaction. More rigorous studies need to recruit more participants, consider more human factors (e.g., gender and age), and collect more measurements (e.g., the time spent by a participant).

References

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD Conference*, pages 383–394, 2006.

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

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] K. Chakrabarti, S. Chaudhuri, and S. won Hwang. Automatic categorization of query results. In *SIGMOD*, pages 755–766, 2004.

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

[6] Z. Chen and T. Li. Addressing diverse user preferences in SQL-query-result navigation. In *SIGMOD ’07*, pages 641–652, 2007.

[7] C. D. Michener and R. R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[8] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1):121–132, 2009.

[9] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512, 2003.