# DQR: A Probabilistic Approach to Diversified Query Recommendation

Ruirui Li[†] Ben Kao[†] Bin Bi[‡] Reynold Cheng[†] Eric Lo[§]
[†]The University of Hong Kong [‡]University of California, Los Angeles [§]Hong Kong Polytechnic University
[†]{rrli, kao, ckcheng}@cs.hku.hk [‡]bbi@cs.ucla.edu [§]ericlo@comp.polyu.edu.hk

## ABSTRACT

Web search queries issued by casual users are often short and with limited expressiveness. Query recommendation is a popular technique employed by search engines to help users refine their queries. Traditional similarity-based methods, however, often result in redundant and monotonic recommendations. We identify five basic requirements of a query recommendation system. In particular, we focus on the requirements of redundancy-free and diversified recommendations. We propose the DQR framework, which mines a search log to achieve two goals: (1) It clusters search log queries to extract query concepts, based on which recommended queries are selected. (2) It employs a probabilistic model and a greedy heuristic algorithm to achieve recommendation diversification. Through a comprehensive user study we compare DQR against five other recommendation methods. Our experiment shows that DQR outperforms the other methods in terms of relevancy, diversity, and ranking performance of the recommendations.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

## Keywords

Query recommendation, query concept, diversification

## 1. INTRODUCTION

The effectiveness of keyword-based search engines, such as Google and Bing, depends largely on the ability of a user to formulate proper queries that are both *expressive* and *selective*. An expressive query clearly and unambiguously describes a user's search intent, while a selective query results in a relatively small set of matching documents. Properly formulated queries bring users directly to the desired information, which make search engines powerful tools for tapping into the wealth of knowledge accessible through the world-wide-web. In practice, translating human thoughts into concise sets of keywords to form queries is never straightforward [3]. This is particularly true for search engine users, who are mostly untrained casual users. In many cases, casual users have very limited background knowledge about the information they are searching for. To assist users in formulating queries, modern search engines are equipped with machine intelligence. This includes techniques like automatic query completion [8] and query recommendation [6, 12, 15, 16]. For the latter, given a user query, a search engine deduces the search intent of the user and recommends a set of queries that are more expressive and selective than the original user input. Our goal is to study the various properties and requirements of a query recommendation system and to propose a novel approach to achieve the desired requirements.

Web search queries are usually very short, typically with only one or two keywords each [21]. Short queries lead to two issues. First, they are often *ambiguous*. For example, the query "Jaguar" can refer to a big cat, an automobile brand-name, or an operating system. In [18], it is reported that up to 23.6% of web search queries are ambiguous. Ambiguity weakens the expressiveness of queries because the search engine may not get the users' hidden search intents. This causes poor retrieval results.

The second issue is that short queries are often not specific enough. For example, a user issuing the query "Disney" may really be interested only in "Disney Pictures". Without knowing the hidden intent, a search engine would return web pages on Disney's theme parks, stores, and cartoon characters in addition to those on the film-making studios. Although there is no ambiguity in the meaning of the keyword "Disney", it is too general a query for the search engine to pinpoint the specific information the user is interested.

In recent years, research has been carried out to tackle the problems of query ambiguity and weak selectivity. One prominent approach is query recommendation. Given a query, e.g., "Jaguar", a search engine provides a list of recommended queries such as "Jaguar Cat", "Jaguar Car", and "Jaguar Mac". This helps the user refine his query by picking the most relevant recommendation.

Many of existing query-recommendation techniques are based on query similarity. Given an input query $\tilde{q}$, a recommender system suggests a set of queries that it has frequently seen and which are the most similar to $\tilde{q}$. The various similarity-based methods differ in the way they measure query similarity. The rationale behind this approach is that similar queries reflect the same or similar search intent, and it is likely that the search intent is more properly expressed by the frequently seen queries, which are the results of the collective wisdom of the community. We observe that similarity-based methods are not always the best approach. This is because queries that are the most similar to $\tilde{q}$ tend to be similar among themselves. This may lead to *redundancy* and *monotonicity* in the recommendation.

Recommendation redundancy refers to the situation where some


*CIKM'12,* October 29–November 2, 2012, Maui, HI, USA.

recommended queries are of equivalent meaning, describing almost the same search intent. For example, consider the input query "Recommendation" and two recommended queries returned by the Bing search engine: "Letter of recommendation examples" and "Recommendation letter samples". Obviously, the recommended queries refer to the same information and they are likely to give very similar search results. They are thus redundant.

Recommendation monotonicity refers to the situation where the recommended queries all refer to search intents of a common predominant interpretation of the input query keywords. For example, consider the input query "Jaguar", the recommendation {"Jaguar parts", "Jaguar sedan", "Jaguar used car", "Jaguar dealer"} all follow the interpretation of "Jaguar as an automobile manufacturer". The recommendation is thus monotonic and it could miss out the real search intent of the user, such as "Jaguar as an animal".

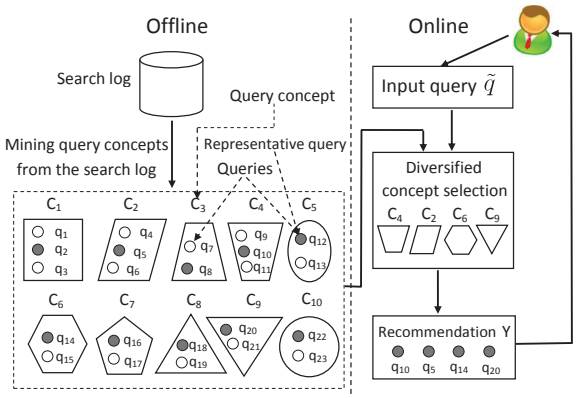


**Figure 1: The DQR framework**

We propose that a good query recommender should go beyond pure similarity and towards providing redundancy-free and diversified recommendations. To avoid redundancy, we highlight the notion of *query concept* (see Figure 1). In our model, a query concept is a set of queries that reflect the same or similar search intent. For example, the queries "Letter of recommendation examples" and "Recommendation letter sample" both describe the same search intent and therefore they should belong to the same query concept. An important pre-processing step of our query recommendation system is to group (previously seen) queries into query concepts. Given an input query $\tilde{q}$, our system first selects and ranks a small set of relevant concepts. A representative query from each of the selected concepts is then extracted. These extracted queries form a list $Y$ of ranked recommended queries. Through query concept selection, we are able to avoid recommendation redundancy. This is because only one (representative) query from each relevant concept is returned to the user. Other redundant queries, which belong to the same concept, are not extracted.

To diversify recommendation, the concept selection component (Figure 1) employs a probabilistic model whose goal is to maximize the likelihood that the recommended concepts (or equivalently their representative queries) cover as many hidden search intents and interpretations of the input query keywords as possible. For example, the recommendation $Y_1$: {"Jaguar Cat", "Jaguar Car ", "Jaguar Mac", "Jaguar NFL"} is superior to the recommendation $Y_2$: {"Jaguar Parts", "Jaguar Dealer", "Jaguar sedan", "Jaguar used cars"}, because $Y_1$ covers more interpretations of the query keyword "Jaguar". It is therefore more likely that at least one of the recommended queries is relevant to the hidden search intent of the user.

To extract query concepts and to construct the probabilistic model for recommendation diversification, we mine search logs. A search log contains historical records, each of which registers the details of a web search conducted by a user. Table 1 shows some sample records extracted from the AOL search engine's search log [17][1]. Each record includes a query string, an anonymous user ID by whom the query was formulated, a query submission time, and the URL subsequently clicked by the user (if done). To distinguish user queries for which recommendations are to be made from those historical queries recorded in the search log, we call the former *input queries* (denoted by $\tilde{q}$) and the latter *log queries* or simply queries.

| UserID | Query String | Time | Clicked URL |
|---|---|---|---|
| 193661 | maps | 2006-05-13 13:16:32 | maps.yahoo.com |
| 193661 | maps | 2006-05-13 13:16:32 | maps.google.com |
| 2356008 | map search | 2006-03-14 07:35:04 | maps.google.com |
| 2356008 | map search | 2006-03-14 07:35:04 | maps.yahoo.com |
| 16462215 | map search | 2006-03-31 10:34:32 | www.mapquest.com |
| 93537 | driving directions | 2006-03-22 23:34:40 | www.mapquest.com |
| 366529 | driving directions | 2006-04-04 20:03:50 | www.randmcnally.com |
| 1965790 | rand mcnally | 2006-03-01 07:21:40 | www.randmcnally.com |

**Table 1: Sample AOL search log records**

Before we proceed to describe the details of our approach, let us enumerate the properties that a good query recommendation system should observe:

**[Relevancy]** Recommended queries should be semantically relevant to the user search query.

**[Redundancy Free]** The recommendation should not contain redundant queries that repeat similar search intents.

**[Diversity]** The recommendation should cover search intents of different interpretations of the keywords given in the input query.

**[Ranking]** Highly relevant queries should be ranked first ahead of less relevant ones in the recommendation list.

**[Efficiency]** Query recommendation provides online help. Therefore, recommendation algorithms should achieve fast response times.

Here we summarize the major contributions of our work:

(1) We study the main features that a query recommender should possess. These include relevancy, redundancy-free, diversity, ranking, and real-time response. Our approach, called DQR, is the first to address all the 5 requirements.

(2) We highlight the notion of query concept to tackle the recommendation redundancy problem. We propose a clustering method to achieve query concept construction.

(3) We propose a concept-based probabilistic query recommendation model, which considers both the requirements of relevancy and diversity.

(4) We present a comprehensive empirical evaluation of our approach against a number of recommendation methods using real search engine datasets. The results show that our approach is both highly effective and efficient in suggesting relevant and diversified recommendations.

## 2. RELATED WORK

Query recommendation is an active research topic. In this section we give a brief account of some representative works on general web search recommendation techniques (Section 2.1). Since our approach uses concept extraction to avoid recommendation redundancy and we aim to achieve recommendation diversification, we highlight some related work on recommendation diversification (Section 2.2) and concept mining (Section 2.3).

### 2.1 Query Recommendation

There are a number of studies on search-log-based query recommendation methods [2, 4, 6]. Many of these studies derive click-through bipartite graphs (CTBG) from search logs to measure

[1]Each log record is associated with many fields. We are displaying only those fields that are of our interest.

query similarity. The set of queries ($Q$) and the set of clicked URLs ($D$) in a search log form two disjoint sets of vertices in a CTBG. An edge connects a query $q \in Q$ and an URL $d \in D$ if $q$ and $d$ appear together in a log record. Figure 2 shows the CTBG derived from the search log shown in Table 1. Assuming that the URLs a user clicked reflect his search intent, queries that share more clicked URLs are considered more similar.

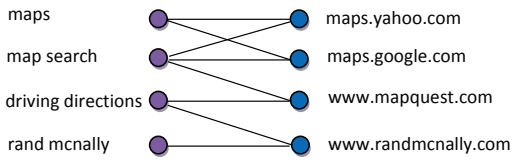


**Figure 2: A click-through bipartite graph**

Based on a similarity measure, queries are clustered. For example, in [2], each query is associated with a term vector that is derived from the contents of the web pages referred to by its clicked URLs. The similarity of two queries is given by the cosine similarity of the queries' term vectors. The k-means algorithm is then applied to cluster the queries. Given an input query $\tilde{q}$, the cluster $C$ to which $\tilde{q}$ belongs is determined. Queries in the cluster are then ranked and the highly ranked queries are returned in the recommendation. In [4], a similar cluster-based recommendation approach is proposed, except that an agglomerative clustering method applied on the CTBG is used instead of k-means. In [21], the similarity of two queries is measured by a combination of the clicked URLs they share as well as the keywords shared by their query strings. A density-based clustering algorithm (DBSCAN) is then applied to form query clusters. Our approach is in sharp contrast to the above methods. Although we also cluster queries, the purpose is not to recommend queries that are in the same cluster of the input query. Instead, each cluster forms one *query concept* and only one representative query from each cluster (concept) is considered in the recommendation procedure to avoid redundancy.

Some studies apply various probabilistic models for query recommendation. For example, in [12, 15, 16], a *query graph* is constructed from a search log in which vertices represent queries in the log. An edge connecting two queries is weighted by a *transition probability*, which is related to the similarity of the two queries. A *hitting time algorithm*, which performs a random walk, is then executed on the graph. The hitting time $h(q_j|q_i)$ is the expected number of steps it takes to reach a query vertex $q_j$ from a starting vertex $q_i$. In [12], given an input query $\tilde{q}$, queries $q_j$'s with the smallest hitting times $h(q_j|\tilde{q})$ are recommended. One weakness of the hitting time approach is that query graphs are huge. For example, the query graph derived from the AOL search log [17] is about 100 GB. Typically, either a depth-first search or a breadth-first search on the query graph is executed to obtain a reduced graph for the execution of the hitting time algorithm. This is computationally expensive and it works against the real-time requirement of query recommendation.

## 2.2 Diversified Recommendation

In [7], Carbonell et al. introduce Maximal Marginal Relevance (MMR) as a technique to diversify results in traditional document retrieval systems. Given a keyword query, MMR selects relevant documents incrementally. When a relevant document $d$ is selected, MMR updates the relevancy of each remaining document $d'$ by computing a penalty, which is based on the similarity between $d$ and $d'$. The next relevant document picked by MMR is then based on the updated relevancy scores. By doing so, MMR favors documents that are not only relevant to the query, but are also not so similar to those that have already been included in the retrieval result. In [15], Ma et al. study diversified query recommendation. Similar to MMR, a recommendation list is constructed incrementally. They employ a reversed hitting time algorithm to ensure that the next query selected for recommendation is not very similar to those that have already been selected. Like other hitting-time-based algorithms, their method suffers from high computational cost in processing huge query graphs. Our approach is similar to that of [15] in that we also construct the recommendation list incrementally. However, we pick recommended queries at the query concept levels to avoid recommendation redundancy. Also, instead of using query graphs, we propose a probabilistic model that allows recommendations to be determined much more efficiently to satisfy the real-time requirement.

## 2.3 Concept Mining

In traditional IR, concepts are often extracted from unstructured text collections to improve retrieval accuracy. Techniques such as LSI [9] are often employed. However, the literature on mining query concepts from search log queries remains sparse.

In [6], Cao et al. propose CACB, which employs a one-pass clustering algorithm to cluster search log queries into query concepts. In Section 4.1 we will give the details of this clustering algorithm, point out its potential weaknesses, and propose a method to improve the clustering results. In [11], Fonseca et al. extract query concepts from *query relation graphs* (QRGs). The idea is to first identify user sessions from the search log (e.g., based on the queries' submission times). Search log queries that frequently co-occur in the same sessions are considered highly associated, and so they are connected by an edge in the QRG. Subsets of queries that are strongly connected in the QRG are taken as query concepts.

# 3. NOTATIONS AND PROBLEM STATEMENT

A search log $R$ consists of a number of log records. Given a log record $r \in R$, we use $r^q$, $r^u$, $r^t$, and $r^d$ to denote the query, the user who issued the query, the query submission time, and the clicked URL of that record, respectively. We use $Q$, $U$, and $D$ to denote the set of all unique queries, the set of all unique user ID's, and the set of all unique clicked URLs, respectively, in the search log.

Given an input query $\tilde{q}$ issued by a user $u$, we say that a recommended query $q$ *matches* $\tilde{q}$ if user $u$ finds $q$ relevant to his search intent. Note that this concept of matching is subjective to the user, and there could be more than one query matching $\tilde{q}$. Without referring to a particular user, we use $e_q$ to denote the event that *an arbitrary user* finds $q$ matching his input query $\tilde{q}$. Similarly, given a set of recommended queries $Y \subseteq Q$, we use $e_Y$ to denote the event that *at least one* query in $Y$ matches $\tilde{q}$. We use $p(e_q)$ and $p(e_Y)$ to denote the probabilities of the corresponding events.

PROBLEM STATEMENT 1. ***(Diversified Query Recommendation)*** *Given an input query $\tilde{q}$ and an integer $m$, the diversified query recommendation problem is to find a set $Y \subseteq Q$ of $m$ queries that maximizes the probability $p(e_Y)$.*

# 4. THE DQR APPROACH

The diversified query recommendation problem is a difficult one. First, the number of queries in $Q$ is huge. For example, there are more than 10 million queries in the AOL dataset. Even picking, say, $m$ = 10 recommended queries from $Q$ involves a huge search space. Second, it is not apparent how the probability $p(e_Y)$ is estimated. To strike a balance between efficiency and recommendation quality, we propose a heuristic approach called DQR. DQR

consists of an offline component and an online component. Figure 1 shows the framework. For the offline component, queries in the search log are clustered. Each cluster forms a query concept. For the online component, given an input query $\tilde{q}$, DQR employs a probabilistic model to find a set $Y_c$ of $m$ query concepts. For each concept $C_i \in Y_c$, a representative query $q_i$ is selected from the cluster of $C_i$. These $m$ selected queries, one from each concept in $Y_c$, form the recommendation $Y$. In this section we first discuss our offline concept mining algorithm (Section 4.1), followed by our probabilistic model for online concept selection and recommendation construction (Section 4.2).

## 4.1 Concept Mining

We group similar queries in the search log to form query concepts. This step is done for two purposes. First, as we have explained in the introduction, by recommending at most one query from each cluster, we avoid recommendation redundancy. Second, the number of query concepts is much smaller than the number of queries in a search log ($|Q|$). Selecting $m$ query concepts (for recommendation) thus involves a much smaller search space than the problem of selecting $m$ queries out of $Q$. This allows the online recommendation process to be done efficiently enough to satisfy the real-time requirement of the recommendation problem.

To cluster queries, we need a similarity measure. We use the set of URLs as the features of queries. Each query $q_i \in Q$ is thus represented by a $|D|$-dimensional vector $\overrightarrow{q_i}$. More specifically, given a query $q_i$ and an URL $d_j$, we follow [10] and assign the *user-frequency-inverse-query-frequency* (UF-IQF) score as the component weight of $q_i$ for dimension $d_j$. This weight, $w'(q_i, d_j)$, is given by the formula:

$$w'(q_i, d_j) = N_u(q_i, d_j) \times \log(|Q|/N_q(d_j)), \quad (1)$$

where $N_u(q_i, d_j)$ is the number of unique users who have issued query $q_i$ and subsequently clicked URL $d_j$, and $N_q(d_j)$ is the number of queries that led to the clicking of URL $d_j$:

$$N_u(q_i, d_j) = |\{u \in U | \exists r \in R, r^u = u, r^q = q_i, r^d = d_j\}|,$$
$$N_q(d_j) = |\{q \in Q | \exists r \in R, r^q = q, r^d = d_j\}|.$$

We further normalize the weights to $w(q_i, d_j)$ by

$$w(q_i, d_j) = \frac{w'(q_i, d_j)}{\sqrt{\sum_{d_k \in D} w'(q_i, d_k)^2}}. \quad (2)$$

The *distance*, $d(q_i, q_j)$, between two queries $q_i$ and $q_j$ is measured by the Euclidean distance of their normalized vectors. We also use $s(q_i, q_j)$ to denote a similarity measure of the queries:

$$d(q_i, q_j) = ||\overrightarrow{q_i} - \overrightarrow{q_j}||, \qquad s(q_i, q_j) = 1 - d(q_i, q_j)/\sqrt{2}. \quad (3)$$

Before we describe our clustering algorithm, let us point out two requirements: (1) *Adaptability*: The clustering algorithm should be able to automatically determine the number of clusters in a search log, since this number is unknown beforehand. (2) *Efficiency*: The algorithm has to be efficient because the number of queries in the search log is huge.

With these requirements k-means, for example, is not suitable. Not only does it require the number of concepts be known, it also requires numerous iterations through the search log queries during the clustering process. As an example, we have applied k-means to cluster the queries in the AOL search log. The algorithm did not terminate in two days.

In view of the efficiency requirement, a one-pass clustering algorithm is proposed in [6]. The algorithm maintains a set of clusters $\mathcal{C}$ (initially $\mathcal{C} = \emptyset$). The algorithm then scans the search log queries. For each query $q_i$, the algorithm locates the cluster $C \in \mathcal{C}$ that is the "*closest*" to $q_i$. If adding $q_i$ to $C$ does not violate a *compactness requirement* of $C$, $q_i$ is added to $C$; otherwise, $q_i$ forms a cluster by itself and this new cluster is added to $\mathcal{C}$.

This one-pass algorithm, although very efficient, is highly sensitive to the order in which log queries are scanned. As a simple example, Figure 3(a) shows three queries and their pairwise distances. Suppose we impose the following compactness requirement: "the average pairwise distance of queries in a cluster $\leq$ 0.5". Then, if the queries are scanned in the order $q_1, q_2, q_3$, the clustering result is $\mathcal{C}_1 = \{\{q_1, q_2\}, \{q_3\}\}$. Note that $q_3$ is not added to the cluster $\{q_1, q_2\}$ because doing so will violate the compactness requirement. We observe that a better clustering result is $\mathcal{C}_2 = \{\{q_1\}, \{q_2, q_3\}\}$ because $q_2$ is closer to $q_3$ than it is to $q_1$. It is easy to see that $\mathcal{C}_2$ can be obtained by scanning the queries in a different order: $q_2, q_3, q_1$.

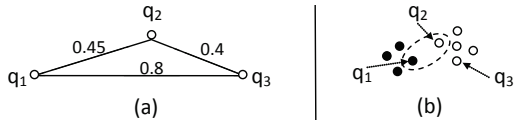


**Figure 3: Clustering example**

Figure 3(b) further illustrates this problem when the one-pass algorithm is applied to cluster a number of queries. We note that if the compactness requirement is set too loose, the one-pass algorithm runs the risk of putting two far points (e.g., $q_1$ and $q_2$) into the same cluster, leading to sub-optimal clustering. By limiting to one pass through the data, the algorithm cannot rectify the clustering result. This argues for a more stringent compactness requirement. However, if the requirement is too stringent, we will end up with a large number of very small clusters, which is not desirable either. Our solution is to use hierarchical clustering: we start with a very stringent compactness requirement to make sure that only points that are very close among themselves form a cluster. We then merge these clusters by gradually relaxing the compactness requirement. Algorithm 1 shows our clustering method.

Given a query cluster $C$, we use the diameter measure ($L(C)$) proposed in [22] as our cluster compactness measure:

$$L(C) = \sqrt{\frac{\sum_{q_i \in C} \sum_{q_j \in C} (||\overrightarrow{q_i} - \overrightarrow{q_j}||)^2}{|C|(|C| - 1)}}. \quad (4)$$

We use $\mu(C)$ to represent the centroid of cluster $C$. Our algorithm maintains a set of clusters $\mathcal{C}$ and the set of their centroids, $\mathcal{P} = \{\mu(C) | C \in \mathcal{C}\}$. It also maintains a compactness requirement $L(C) \leq L \; \forall C \in \mathcal{C}$, where $L$ is a compactness bound that changes from 0 (initially) to $L_{max}$ with an increment step of $L_\delta$ across iterations of the algorithm. Initially, each query $q_i \in Q$ forms a cluster $\{q_i\}$ by itself. This is equivalent to having a compactness requirement of $L(C) = 0$ for each cluster. In each iteration of the algorithm, we group the centroids (in $\mathcal{P}$) using the one-pass algorithm of [6] with the compactness bound $L$. If a set of centroids are grouped together by the one-pass algorithm, we merge their corresponding clusters to form a bigger cluster. At the end of an iteration, we update the set of cluster centroids ($\mathcal{P}$), increase the compactness bound $L$ by $L_\delta$, and repeat until $L$ exceeds the maximum bound $L_{max}$. Figure 4 illustrates our clustering algorithm.

We have argued that a query clustering algorithm should not execute too many iterations because the number of log queries are typically huge. By controlling the value of the compactness bound increment $L_\delta$, we can control the number of clustering iterations,

**Algorithm 1:** Clustering Algorithm

**Input**: query set $Q = \{q_1, q_2, ..., q_{|Q|}\}$, increment step $L_\delta$ and compactness threshold $L_{max}$
**Output**: a set of clusters $\mathcal{C} = \{C_i | C_i \subset Q\}$

**Initialization:** $\mathcal{C} \leftarrow \{\{q_i\} | q_i \in Q\}$; $\mathcal{P} \leftarrow Q$; $L \leftarrow 0$;
**repeat**
  $CP \leftarrow \emptyset$; `//clustering points in` $\mathcal{P}$
  **foreach** $p_i \in \mathcal{P}$ **do**
    **if** $CP = \emptyset$ **then**
      $CP \leftarrow \{\{p_i\}\}$;
    **else**
      $C^* \leftarrow \arg\min_{C \in CP} d(\mu(C), p_i)$;
      **if** $L(C^* \cup \{p_i\}) \leq L$ **then**
        $C^* \leftarrow C^* \cup \{p_i\}$;
      **else**
        $CP \leftarrow CP \cup \{\{p_i\}\}$;
  `// Merge clusters`
  Update $\mathcal{C} \leftarrow \{X | X = \cup_{C \in \mathcal{C}, \mu(C) \in A} C, \ A \in CP\}$;
  Update $\mathcal{P} = \{p_i | p_i = \mu(C_i), C_i \in \mathcal{C}\}$;
  $L = L + L_\delta$;
**until** $L > L_{max}$;
**return** $\mathcal{C}$

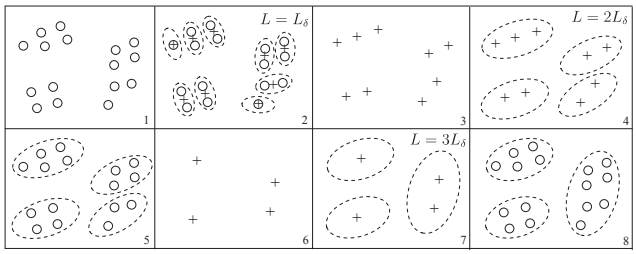


**Figure 4: Clustering illustration: (1) Some queries. (2) Grouped with bound $L = L_\delta$. (3) Cluster centroids. (4) Centroids grouped with $L = 2L_\delta$. (5) Merged query clusters. (6) Centroids of the merged clusters. (7) Centroids grouped with $L = 3L_\delta$. (8) Merged query clusters.**

and thus the clustering efficiency. We remark that the choice of the maximum bound $L_{max}$ is more of an engineering effort. One way to determine a "good" $L_{max}$ is to perform a user study on the recommender's effectiveness under a few values of $L_{max}$. Fortunately, our experiment results show that our DQR approach gives very good performance over a wide range of $L_{max}$ values so it is not critical that an optimal value of $L_{max}$ be found.

## 4.2 Query Recommendation

Let $\mathcal{C}$ be the set of query concepts extracted from the search log. Given an input query $\tilde{q}$, we construct a list of recommended queries $Y$ with a two-step process. First, we use a probabilistic model to select a set $Y_c \subseteq \mathcal{C}$ of $m$ recommended query concepts. Then, $Y$ is constructed by selecting a representative query from each of the concepts in $Y_c$.

Recall that in Section 3 we defined the notion of a recommended query $q$ *matches* a user input query $\tilde{q}$. Since the first step of our recommendation procedure operates at the query concept level, we extend our notion of *matching* to the concept level as well. In particular, given a user $u$ who has issued an input query $\tilde{q}$, we say that a query concept $C \in \mathcal{C}$ matches $\tilde{q}$ if user $u$ finds some queries in $C$ relevant to his search intent. We use $e_C$ to denote the event that an arbitrary user $u$ finds $C$ matches $\tilde{q}$. Note that, depending on the perspective of the user, there could be more than one concept $C$ matching $\tilde{q}$. Furthermore, given a set $Y_c \subseteq \mathcal{C}$ of query concepts we use $e_{Y_c}$ to denote the event that an arbitrary user $u$ finds some concepts in $Y_c$ matching $\tilde{q}$. With these notations, the first step (concept selection) of our two-step recommendation procedure can be performed as followed:

**[Concept Selection]** Given an input query $\tilde{q}$, let $C_{\tilde{q}} \in \mathcal{C}$ be the query concept $\tilde{q}$ belongs[2]. Since the user has expressed his search intent with the input query $\tilde{q}$, the concept $C_{\tilde{q}}$ is deemed relevant to the user's search intent. To recommend queries selected from other concepts, our heuristic algorithm finds a set $Y_c \subseteq \mathcal{C} - \{C_{\tilde{q}}\}$ of $m$ query concepts such that $p(e_{Y_c} | e_{C_{\tilde{q}}})$ is maximized. To better explain the model and the algorithm, we first give a few definitions.

After a user issues a query $q$ to a search engine, he may click a set of URLs, $s$, returned by the search engine. We use $I$: $q \rightarrow s$ to denote such an interaction, which is captured by a set of records in the search log. For example, the first two records in the search log shown in Table 1 shows that a user issued the query "maps" and subsequently clicked two URLs: *maps.yahoo.com* and *maps.google.com*. We call $s$ the *click-set* of the interaction $I$. Note that records that constitute an interaction all share the same user ID, query string, and query-submission time. We use $\mathcal{I}$ to denote the set of all such interactions found in the search log $R$. We use $CS$ to denote the set of all click-sets found in $R$.

For example, from the log in Table 1, we get four queries:

$$Q = \left\{ \begin{array}{ll} q_1 = \text{"maps"}, & q_2 = \text{"map search"}, \\ q_3 = \text{"driving directions"}, & q_4 = \text{"rand mcnally"} \end{array} \right\},$$

three click-sets:

$$CS = \left\{ \begin{array}{l} s_1 = \{\texttt{maps.yahoo.com, maps.google.com}\}, \\ s_2 = \{\texttt{www.mapquest.com}\}, \quad s_3 = \{\texttt{www.randmcnally.com}\} \end{array} \right\},$$

and six interactions:

$$\mathcal{I} = \left\{ \begin{array}{lll} q_1 \rightarrow s_1, & q_2 \rightarrow s_1, & q_2 \rightarrow s_2 \\ q_3 \rightarrow s_2, & q_3 \rightarrow s_3, & q_4 \rightarrow s_3 \end{array} \right\}.$$

There are $\binom{|\mathcal{C}|-1}{m}$ or $O(|\mathcal{C}|^m)$ ways of picking $m$ concepts into the set $Y_c$ out of the pool of query concepts $\mathcal{C} - \{C_{\tilde{q}}\}$. Enumerating them to find the one that maximizes $p(e_{Y_c} | e_{C_{\tilde{q}}})$ is too inefficient. Instead, we apply a greedy strategy and construct the set $Y_c$ incrementally.

Let $Y_c'$ (which is initially empty) be a set of concepts our algorithm has picked so far. We want to add more concepts to $Y_c'$ (one concept at a time) until the selection contains $m$ concepts. At each step, our greedy heuristic picks the concept $C \in \mathcal{C} - \{C_{\tilde{q}}\} - Y_c'$ that maximizes the probability increment:

$$\Delta(Y_c', C, C_{\tilde{q}}) = p(e_{Y_c' \cup \{C\}} | e_{C_{\tilde{q}}}) - p(e_{Y_c'} | e_{C_{\tilde{q}}}).$$

To estimate this value, we make one reasonable assumption: we assume that the search intent of a user, who issues a query $q$ to a search engine, is truly reflected by the URLs he would click when presented with the search results of $q$. This has two implications:

IMPLICATION 1. *Given concepts $C_a$ and $C_b$, and a click-set $s$, we have $p(e_{C_b} | e_s, e_{C_a}) = p(e_{C_b} | e_s)$, where $e_s$ represents the event that the user clicks the URLs in $s$. That is, knowing that concept $C_a$ matches the input query $\tilde{q}$ does not add more information to estimating $p(e_{C_b})$ given that we already know the user would click the click-set $s$.*

[2]If $\tilde{q}$ has previously appeared in the search log, we simply look up which concept in $\mathcal{C}$ contains $\tilde{q}$. If $\tilde{q}$ is a new query previously unseen, we find the query $q_\star \in Q$ whose content (such as the keywords contained in the query strings) is the most similar to that of $\tilde{q}$ and use the concept $q_\star$ belongs as $C_{\tilde{q}}$.

IMPLICATION 2. *Given two concepts $C_a$, $C_b$ and a click-set $s$, we have $p(e_{C_a}, e_{C_b}|e_s) = p(e_{C_a}|e_s)p(e_{C_b}|e_s)$. That is, the event of $C_a$ matching $\tilde{q}$ and the event of $C_b$ matching $\tilde{q}$ are conditionally independent given $e_s$.*

With the conditional independence assumption, we derive $p(e_{Y'_c}|e_{C_{\tilde{q}}})$ (and likewise for $p(e_{Y'_c \cup \{C\}}|e_{C_{\tilde{q}}})$) by marginalizing the probability across all click-sets in $CS$:

$$\begin{aligned} p(e_{Y'_c}|e_{C_{\tilde{q}}}) &= \sum_{s \in CS} p(e_{Y'_c}|e_s, e_{C_{\tilde{q}}}) p(e_s|e_{C_{\tilde{q}}}), \\ &= \sum_{s \in CS} p(e_{Y'_c}|e_s) p(e_s|e_{C_{\tilde{q}}}). \end{aligned} \quad (5)$$

Then,

$$\begin{aligned} &\Delta(Y'_c, C, C_{\tilde{q}}) \\ = &\sum_{s \in CS} p(e_{Y'_c \cup \{C\}}|e_s) p(e_s|e_{C_{\tilde{q}}}) - \sum_{s \in CS} p(e_{Y'_c}|e_s) p(e_s|e_{C_{\tilde{q}}}), \\ = &\sum_{s \in CS} \left\{ p(e_{Y'_c \cup \{C\}}|e_s) - p(e_{Y'_c}|e_s) \right\} p(e_s|e_{C_{\tilde{q}}}). \end{aligned} \quad (6)$$

Since $p(e_{Y'_c}|e_s) = 1 - \prod_{C_j \in Y'_c}(1 - p(e_{C_j}|e_s))$, we have

$$\begin{aligned} &p(e_{Y'_c \cup \{C\}}|e_s) - p(e_{Y'_c}|e_s) \\ = &\left\{ 1 - (1 - p(e_C|e_s)) \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)) \right\} \\ &- \left\{ 1 - \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)) \right\}, \\ = &\left\{ \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)) \right\} - \left\{ (1 - p(e_C|e_s)) \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)) \right\}, \\ = &\; p(e_C|e_s) \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)). \end{aligned} \quad (7)$$

Substituting Equation (7) into Equation (6), we get

$$\Delta(Y'_c, C, C_{\tilde{q}}) = \sum_{s \in CS} \left\{ p(e_s|e_{C_{\tilde{q}}}) p(e_C|e_s) \prod_{C_j \in Y'_c} (1 - p(e_{C_j}|e_s)) \right\}. \quad (8)$$

To evaluate the right-hand side of Equation (8) we need to estimate $p(e_s|e_{C_{\tilde{q}}})$, $p(e_C|e_s)$, and $p(e_{C_j}|e_s)$. To do so, we consider the set of interactions recorded in the search log. Note that $p(e_s|e_{C_{\tilde{q}}})$ is the probability that a user would click the click-set $s$ if he finds the concept $C_{\tilde{q}}$ relevant to his search intent. We estimate this probability by counting the number of interactions in $\mathcal{I}$ that involve some query of the cluster $C_{\tilde{q}}$ (we denote this number by $(NI(C_{\tilde{q}}))$), and the number of those interactions that involve click-set $s$ (denoted by $NI(C_{\tilde{q}}, s)$). We then approximate the probability $p(e_s|e_{C_{\tilde{q}}})$ by the fraction $NI(C_{\tilde{q}}, s)/NI(C_{\tilde{q}})$. Formally,

$$p(e_s|e_{C_{\tilde{q}}}) \approx \frac{NI(C_{\tilde{q}}, s)}{NI(C_{\tilde{q}})} = \frac{|\{(I : q \to s) \in \mathcal{I} \text{ s.t. } q \in C_{\tilde{q}}\}|}{|\{(I : q \to s') \in \mathcal{I} \text{ s.t. } q \in C_{\tilde{q}} \text{ and } s' \in CS\}|}.$$

Similarly, $p(e_{C_j}|e_s)$ is estimated by

$$p(e_{C_j}|e_s) \approx \frac{NI(C_j, s)}{NI(s)} = \frac{|\{(I : q \to s) \in \mathcal{I} \text{ s.t. } q \in C_j\}|}{|\{(I : q' \to s) \in \mathcal{I} \text{ s.t. } q' \in Q\}|}.$$

Note that the computation of all these probabilities can be done offline. With these quantities, the online estimation of $\Delta(Y'_c, C, C_{\tilde{q}})$ can be done efficiently (see Section 5.5.4). Finally, the concept $C$ that gives the largest $\Delta(Y'_c, C, C_{\tilde{q}})$ is added to the set $Y'_c$. We repeat this process until $Y'_c$ contains $m$ concepts. Algorithm 2 summaries the procedure.

**Algorithm 2:** Diversified Concept Recommendation

**Input**: $m, C_{\tilde{q}}, \mathcal{C}$
**Output**: a concept list $Y_c =< C_1, C_2, \cdots, C_m >$
$\mathcal{X} \leftarrow \mathcal{C} - \{C_{\tilde{q}}\}$; $Y'_c \leftarrow \emptyset$
**for** $i \leftarrow 1$ **to** $m$ **do**
  $C_i \leftarrow \arg\max_{C \in \mathcal{X}} \Delta(Y'_c, C, C_{\tilde{q}})$
  $Y'_c \leftarrow Y'_c \cup \{C_i\}$
  $\mathcal{X} \leftarrow \mathcal{X} - \{C_i\}$
**return** $Y'_c$

With the $m$ concepts selected, the next step of DQR is to pick one representative query from each concept to form the recommendation list $Y$. We pick representative queries by popularity votes. More specifically, given a concept $C$, its representative query is the one that is issued by the largest number of distinct users in the search log among all the queries in $C$. For example, from the search log of Table 1, we see that the query "maps search" has been issued by two distinct users, while the query "maps" has been issued by just one. The former is therefore more popular. Finally, the queries in $Y$ are ranked according to the order in which their corresponding concepts are selected into the set $Y'_c$, e.g., if a concept is added to $Y'_c$ first, then its representative query will be ranked first in $Y$.

# 5. EXPERIMENTS

In this section we present the experimental results for evaluating our DQR method. We first briefly describe the datasets used in Section 5.1. Then we describe five other recommendation methods against which we compare DQR in Section 5.2. In Section 5.3 we compare the recommendations made by the various methods for an example input query to highlight the differences of the methods. Finally, in Section 5.4, we give a comprehensive analysis of the various methods with a user study.

## 5.1 Data Sets

Our experiments were done on two real datasets: (1) `AOL` is a search log collected from the *AOL* search engine [17], and (2) `SOGOU` is the search log of the Chinese search engine *Sogou*[3]. We follow the method employed in [13, 20] to clean the data. This involves: (1) removing non-alpha-numerical characters from query strings except for '.' (dot) and '␣' (space), (2) removing redundant space characters, and (3) removing queries that appear only once in the search log. (These queries are likely typos.) Similarly, we applied the cleaning steps to the `SOGOU` dataset except that we retain Chinese characters in `SOGOU` queries. Table 2 shows the statistics of the datasets.

| Dataset | | $\|Q\|$ | $\|D\|$ | $\|U\|$ | $\|R\|$ |
|---|---|---|---|---|---|
| AOL | Raw | 10,154,742 | 1,632,788 | 657,426 | 36,389,567 |
| | Cleaned | 2,516,156 | 1,346,752 | 491,720 | 16,895,112 |
| SOGOU | Raw | 5,630,793 | 15,951,255 | 10,679,392 | 51,537,377 |
| | Cleaned | 3,599,539 | 14,504,373 | 10,328,952 | 49,506,111 |

**Table 2: Dataset statistics**

## 5.2 Recommendation Algorithms

Besides our DQR method, we have implemented five other query recommendation methods for the comparison study.

[3]Available at http://www.sogou.com/labs/dl/q-e.html.

**[SR (Similarity-based Ranking)]** Given an input query $\tilde{q}$, we first find the log query $q_\star \in Q$ whose query string is the most similar to that of $\tilde{q}$. (Since a search log typically consists of a large number of queries, it is very likely that $\tilde{q}$ already exists in $Q$. This is true especially for frequently asked queries [6, 12, 16]. In this case, $q_\star = \tilde{q}$.) The top $m$ queries $q \in Q - \{q_\star\}$ with the smallest distances $d(q, q_\star)$ (see Equation 3) are put into the recommendation list $Y$. These queries are sorted in increasing value of $d(q, q_\star)$. SR is purely similarity-based method. It considers recommendation relevancy but does not consider redundancy or diversity.

**[MMR (Maximal Marginal Relevance)]** The second recommendation method adopts the idea of MMR from [7]. The recommendation set $Y$ is constructed incrementally. Initially, $Y = \emptyset$. Recommended queries are added to $Y$ one at a time. Given a $Y$ that contains some $k < m$ queries, we define the *marginal relevance* score $MR(q)$ of each query $q \in Q - Y$ by

$$MR(q) = \lambda \cdot s(q, q_\star) - (1 - \lambda) \cdot \max_{q' \in Y} s(q, q'), \qquad (9)$$

where $0 \leq \lambda \leq 1$ is a constant parameter[4] and $s(q_i, q_j)$ measures the similarity of two queries $q_i$ and $q_j$ (Equation 3). The MR score thus favors queries that are similar to the input query (represented by $q_\star$) but penalizes those that are similar to some query $q'$ that is already selected and put in $Y$. The query $q$ that gives the highest $MR(q)$ score is added to $Y$. The process repeats until $Y$ contains $m$ recommended queries. The queries in $Y$ are sorted according to the order in which they are added to $Y$. MMR considers both recommendation relevancy and diversity. However, it does not operate at the concept level and does not consider redundancy.

**[CACB (Context-Aware Concept-Based Method)]** CACB, proposed in [6], is based on search sessions. CACB first scans the search log to identify user search sessions. Queries are considered to be in the same session if they are issued by the same user and their submission times all fall within a small window. (In many studies a window size of about 30 minutes is used.) Moreover, CACB clusters log queries using the one-pass algorithm proposed in [6] (see Section 4.1) to derive query concepts. Given a user search session $W$, CACB extracts the sequence of queries issued in $W$. These queries are then mapped to their query concepts to form a concept sequence. For example, let $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$ be the query sequence found in a session. This sequence is transformed to $C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$, where $C_i$ is the concept $q_i$ belongs. If successive concepts in the concept sequence are the same, they are merged. For example, if $C_2 = C_3$, then the concept sequence becomes $C_1 \rightarrow C_2 \rightarrow C_4$.

Given an input query $\tilde{q}$, CACB first finds the concept $C_{\tilde{q}}$ that $\tilde{q}$ belongs. It then selects the top $m$ concepts that follow $C_{\tilde{q}}$ most frequently according to the concept sequences derived from the search log. Finally, the most popular query from each such selected concept is put into the recommendation list $Y$.

Like DQR, CACB uses query concepts and thus it combats recommendation redundancy. However, it does not consider diversity. Moreover, its one-pass clustering algorithm, as we have explained, may give low-quality result in concept extraction.

**[DQR-ND (DQR with No Diversity)]** DQR-ND is exactly the same as DQR except that it does not consider diversity. When picking a concept $C$ to be included into the concept set $Y_c$, DQR-ND maximizes the following objective function:

$$\Delta_{ND}(Y_c', C, C_{\tilde{q}}) = \sum_{s \in CS} p(e_s|e_{C_{\tilde{q}}}) p(e_C|e_s). \qquad (10)$$

[4]In our experiments, we set $\lambda = 0.4$, which gives the best overall performance for MMR over a wide range of $\lambda$ values we tested.

Comparing Equation 10 with Equation 8, we see that the term $\prod_{C_j \in Y_c'}(1 - p(e_{C_j}|e_s))$ is removed. So, the concepts already selected in $Y_c'$ are not taken into account. We compare the performance of DQR and DQR-ND to evaluate the effectiveness of our probabilistic model in achieving recommendation diversity.

**[DQR-OPC (DQR with One Pass Clustering)]** DQR-OPC also follows the same framework of DQR except that it extracts query concepts by employing the one-pass clustering algorithm proposed in [6] instead of Algorithm1 we proposed. We compare the performance of DQR and DQR-OPC to evaluate the effectiveness of our clustering algorithm in query concept extraction.

Table 3 summarizes the features of the various methods.

| Method | Consider redundancy | Consider diversity | Remarks |
|---|---|---|---|
| SR | ✗ | ✗ | No query concepts |
| MMR | ✗ | ✓ | No query concepts |
| CACB | ✓ | ✗ | One-pass clustering |
| DQR-ND | ✓ | ✗ | |
| DQR-OPC | ✓ | ✓ | One-pass clustering |
| DQR | ✓ | ✓ | |

**Table 3: Features of the recommendation methods**

## 5.3 A Recommendation Example

We illustrate the differences of the recommendation methods by comparing their recommendation lists for the query "yahoo" using the AOL dataset. Table 4 shows the results. In the table, we also show the recommendation made by Google Search[5]. We remark that a direct comparison against Google search is difficult because we do not have Google's log or the details of Google's recommendation algorithm. Nonetheless, Google's recommendation serves as an interesting reference.

The first thing we notice from Table 4 is that many queries recommended by SR and MMR are "typo queries"[6], while there are no typo queries recommended by CACB, DQR-ND, or DQR. The reason is that the latter three methods employ query concept mining and recommend only representative queries from relevant concepts. Since a representative query is the most popular one within its concept, it is unlikely a typo. The list recommended by DQR-OPC also contains some typo queries, showing that the one-pass clustering algorithm is less than perfect in extracting query concepts. The results show the importance of mining query concepts for recommender systems in avoiding typo queries.

Next we observe that queries recommended by SR are all relevant to the input query, but they are all redundant. This shows the weakness of a pure similarity-based recommender that does not perform concept extraction. The list recommended by MMR, which does not operate at the concept level either, also contains redundancy. By properly clustering queries into concepts, the lists recommended by DQR-ND and DQR are redundancy-free. Minor redundancy is also found in the lists of CACB and DQR-OPC, such as "myspace" and "myspace.com"; and "yahoo search engine" and "yahoo search". This shows that the one-pass clustering algorithm sometimes fails to group redundant queries into the same concept.

If we remove redundant queries from MMR's list, MMR's recommendation is quite diverse. For example, the queries "yahoo web messenger", "yahoo maps", "yahoo canada" and "yahoo finance" cover different aspects of "yahoo". This emphasis of diversity, however, could overshoot at times. For example, "play 13" and "minnesotajaycees" do not seem relevant to the input query. CACB

[5]The recommendation made by Google varies with time. The list shown is the one we obtained at the time of writing.

[6]Although we have removed queries that appear only once in the log to remove typo queries, the typo queries shown in Table 4 are quite common and so are not removed by our cleaning procedure.

| Rank | SR | MMR | CACB | DQR-ND | DQR-OPC | DQR | Google |
|---|---|---|---|---|---|---|---|
| 1 | www yahoo | www yahoo | google | yahoo.com | yahoo.c om | yahoo.com | yahoo login |
| 2 | yahoo.con | wwwyahoo.om | myspace | yahoo mail | yahoo.m | yahoo mail | yahoo sign up |
| 3 | www yahoo.com | y yahoo | msn | yahoo search | yahoo.c | yahoo groups | yahoo email |
| 4 | www.yahoo | yahoo web messenger | ebay | sign in yahoo id | messenger.yahoo.com | yahoo maps | yahoo messenger download |
| 5 | www.yahoo.comhttp | play 13 | yahoo.com | yahoo maps | mail.yahoo.com | yahoo games | yahoo news |
| 6 | www.yahoo com | yahoo maps | hotmail | yahoo groups | yahoomaps.com | yahoo instant messenger | yahoo weather |
| 7 | yahoo.comhttp | minnesotajaycees | mapquest | yahoo games | yahoo.com | yahoo finance | yahoo messenger |
| 8 | www.yahoo.com | yahoo canada | craigslist | yahoo instant messenger | yahoo search engine | yahoo people search | yahoo groups |
| 9 | http yahoo.com | yahoo.com chat rooms | myspace.com | yahoo finance | my yahoo | yahoo dictionary | |
| 10 | yahoo.vcom | yahoo finance | ask jeeves | yahoo news | yahoo search | yahoo news | |

**Table 4: Top 10 queries recommended by the 6 methods for the input query "yahoo"**

offers a very interesting list. Most of the recommended queries do not match the user's search intent ("yahoo"). However, those queries point to other interesting sources that are similar to yahoo services. Recall that CACB's objective is to *predict* a user's next query, the recommendation it makes does not necessarily has to be relevant to the user's input query.

DQR's recommendation is very diversified. This diversity is weakened in DQR-ND. For example, "yahoo search" is a key function of "yahoo.com", and "sign in yahoo id" is a step of logging into "yahoo mail". They thus cover the same aspects. By applying Equation 8, DQR considers the concepts that are already in the set $Y_c$ when it adds more concepts to $Y_c$. This further diversifies the recommendation and allows DQR to replace the query "sign in yahoo id" by, e.g., "yahoo dictionary" in its recommendation.

Google only recommended 8 queries among which we see some redundancy, e.g., "yahoo messenger download" and "yahoo messenger" are similar. This example shows that DQR is able to give a better recommendation despite it was trained with a much smaller search log.

## 5.4 User Study

To evaluate the performances of the recommendation methods, we carried out a user study using the AOL dataset. We invited 12 people to judge the recommendations given by the six methods. We selected 60 test queries from the AOL search log that we thought the judges could comprehend. These queries are listed in [1]. We then randomly divided the 12 judges into two groups of 6 each. The 60 test queries were also randomly divided into two groups of 30 queries each. Each group of 6 judges were assigned 30 test queries. For each test query, each judge was presented with the recommendation given by a particular recommendation method. The judges were asked to evaluate the recommendations presented to them. We followed the user study procedure employed in [12] in assigning a set of recommendations to each judge to evaluate. Table 5 shows this assignment. The set of 30 queries were further randomly divided into 6 sets: Query-Set-1 to Query-Set-6, each containing 5 queries. The entry marked with a ▶ in Table 5, for example, shows that Judge No. 3 was presented with the recommendations given by the MMR method for the queries of Query-Set-6. Note that the judges were not told which methods were used to generate the recommendations. The evaluation assignment (Table 5) ensured that each judge evaluated recommendations that cover all 30 queries and all 6 methods. Also, no judge evaluated more than one recommendation (given by different methods) of the same query. This is purposely done to avoid bias induced by the evaluation order.

| | SR | MMR | CACB | DQR-ND | DQR-OPC | DQR |
|---|---|---|---|---|---|---|
| Judge1 | QuerySet1 | QuerySet2 | QuerySet3 | QuerySet4 | QuerySet5 | QuerySet6 |
| Judge2 | QuerySet6 | QuerySet1 | QuerySet2 | QuerySet3 | QuerySet4 | QuerySet5 |
| Judge3 | QuerySet5 | ▶ QuerySet6 | QuerySet1 | QuerySet2 | QuerySet3 | QuerySet4 |
| Judge4 | QuerySet4 | QuerySet5 | QuerySet6 | QuerySet1 | QuerySet2 | QuerySet3 |
| Judge5 | QuerySet3 | QuerySet4 | QuerySet5 | QuerySet6 | QuerySet1 | QuerySet2 |
| Judge6 | QuerySet2 | QuerySet3 | QuerySet4 | QuerySet5 | QuerySet6 | QuerySet1 |

**Table 5: Evaluation assignment**

To evaluate a recommendation, a judge was given an evaluation form. Table 6 shows an example. On the form, the judge was shown the test query (e.g., "interview") and a ranked list of 10 recommended queries (e.g., "interview preparation"). For each recommended query, the judge had to label it with a relevancy score: irrelevant (0), partially relevant (1), and relevant (2). Also, for those partially relevant or relevant recommended queries, the judge had to group them so that queries sharing the same search intent are put into the same group. An example evaluation done by a judge is indicated in Table 6 by ✓ marks. For example, the evaluation shows that the judge found four search intents from the recommendation that are relevant to the test query "interview". In particular, the judge regarded the recommended queries "interview questions" and "interview answers" to be of the same search intent.

| Query: "interview" | Relevance | | | Latent Search Intents | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| interview preparation | 0 | 1 | 2✓ | $I_1$✓ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| types of interview | 0 | 1 | 2✓ | $I_1$ | $I_2$✓ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| interview questions | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$✓ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| interview answers | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$✓ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| interview techniques | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$ | $I_4$✓ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| interview tips | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$ | $I_4$✓ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| interview skills | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$ | $I_4$✓ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| job interview tips | 0 | 1 | 2✓ | $I_1$ | $I_2$ | $I_3$ | $I_4$✓ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| working hours | 0✓ | 1 | 2 | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |
| admission | 0✓ | 1 | 2 | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_{10}$ |

**Table 6: An example evaluation form**

## 5.5 Results

We compare the quality of the recommendations given by the six methods in terms of *relevancy*, *diversity* and *ranking performance*. The results shown in this section are those for the AOL dataset. We have also conducted a similar study on the SOGOU dataset. The conclusions drawn for SOGOU is very similar to those of AOL. Due to space limitation, we report the results on SOGOU in [1].

### 5.5.1 Relevancy Performance

Table 7 shows the statistics of the relevancy scores the methods received. The first three rows show the score label distributions. The row $N_{1,2}$ shows the average number of partially relevant (score 1) or relevant (score 2) recommended queries in an evaluation. (Recall that there are 10 recommended queries on each evaluation form. See Table 6). The row $S_{1,2}$ shows the average score of non-irrelevant recommended queries and the row $S_{0,1,2}$ shows the average score of all the recommended queries.

| Relevancy score | SR | MMR | CACB | DQR-ND | DQR-OPC | DQR |
|---|---|---|---|---|---|---|
| 0 (irrelevant) | 11% | 35% | 48% | 14% | 12% | 14% |
| 1 (partially relevant) | 31% | 28% | 28% | 23% | 27% | 24% |
| 2 (relevant) | 58% | 37% | 24% | 63% | 61% | 62% |
| $N_{1,2}$ | 8.9 | 6.5 | 5.2 | 8.6 | 8.8 | 8.6 |
| $S_{1,2}$ | 1.65 | 1.57 | 1.47 | 1.73 | 1.70 | 1.72 |
| $S_{0,1,2}$ | 1.47 | 1.02 | 0.76 | 1.50 | 1.50 | 1.48 |

**Table 7: Relevancy performance**

From the results, we see that SR receives very good relevancy

scores. In particular, its recommendations contain the least percentage of irrelevant queries (score 0) among all the six methods. This is not surprising because SR only recommends queries that are the most similar to input queries. About 1/3 of MMR's recommended queries are irrelevant, and there are only slightly more than 1/3 of them are considered straightly relevant by the judges. This shows that MMR sacrifices too much relevancy to achieve diversity. The relevancy scores of CACB are even lower. As we have explained, the objective of CACB is to predict the next query a user is likely to issue following his current input query in a user session. Since the search intent of a user could drift over a session, CACB's recommendation does not necessarily have to be relevant to the original input query. This explains CACB's low relevancy scores even though its recommendations are sometimes interesting (Table 4). The relevancy scores of the three DQR methods are very good and are comparable to that of SR. In particular, more than 60% of their recommended queries are rated straightly relevant. These ratings are even better than SR's. The reason is that queries recommended by the DQR-based methods are representatives of query concepts. These representative queries are usually better-worded, more expressive and noise-free. Therefore, it is more likely that the judges found them straightly relevant. Among the three methods, DQR-ND gets slightly more '2' scores (63%). This is because DQR-ND focuses more on similarity than diversity. DQR-OPC has slightly fewer '2' scores (61%) because the one-pass clustering algorithm sometimes gives inferior concept extraction. Overall, the relevancy performance of DQR is very good. For example, on average, 8.6 out of the 10 recommended queries ($N_{1,2}$) DQR made are considered either partially relevant or relevant to the judges.

### 5.5.2 Diversity Performance

Next, we analyze how well the recommendations made by the various methods cover distinct relevant search intents. Recall that a judge had to group non-irrelevant recommended queries on an evaluation form into distinct search intents. We define the measure *Intent-Coverage@*$k$ (or *IC@*$k$ for short) [23] as the number of search intents so identified among the top $k$ recommended queries on an evaluation form. For example, consider Table 6, we have *IC@*4 equals 3 because three search intents were identified by the judge from the top 4 recommended queries given. Note that irrelevant queries are ignored and *IC@*$k$ is upper-bounded by $k$. Figure 5 shows the average Intent-Coverage of the recommendations made by the six methods as $k$ varies from 1 to 10.

From the figure, we see that SR has a low intent coverage. Even though most of SR's recommended queries are relevant, they are highly redundant and cover only a few search intents. Although CACB's recommendations tend to cover different aspects, however, they suffer from low relevancy. Therefore, the number of relevant distinct search intents given by CACB is small. This explains its low *IC@*$k$ scores. MMR, which focuses on diversification, has a larger intent coverage than CACB. However, many of its recommended queries are irrelevant, which keeps MMR's *IC@*$k$ scores low. DQR gives the best intent coverage. This shows that the probabilistic model DQR uses in diversifying recommended queries works very well. For example, its score at $k = 10$ shows that on average the 10 recommended queries given by DQR cover 7 distinct relevant search intents. DQR-ND, which does not employ Equation 8 to improve recommendation diversity has lower *IC@*$k$ scores than DQR. Fortunately, DQR-ND uses Algorithm 1 to form query concepts, and selects relevant concepts from which representative queries are obtained to compose recommendation lists. With high-quality concepts formed, representative queries recommended to a user tend to distinguish among themselves. This results in a reasonably good intent coverage. The importance of high-quality query concepts in diversified query recommendation is further reflected by the performance of DQR-OPC. From Figure 5, we see that DQR-OPC has lower *IC@*$k$ scores than those of DQR-ND and DQR. This is because, in some cases, the one-pass clustering algorithm fails to put queries that cover the same search intent into the same cluster. This may result in DQR-OPC recommending representative queries which, although come from different clusters, indeed are covering the same search intent.

### 5.5.3 Ranking Performance

Next, we analyze how well the methods rank the recommended queries. Intuitively, relevant queries should be ranked higher in the recommendation list. We consider two ranking measures, namely, *Normalized Discounted Cumulative Gain* (NDCG) [5, 14] and *Mean Reciprocal Rank* (MRR) [19]. The *NDCG@*$k$ score is defined as:

$$NDCG@k = Z_k \sum_{i=1}^{k} \frac{(2^{l_i} - 1)}{\log(i+1)}, \tag{11}$$

where $l_i$ is the relevancy score (0, 1, or 2) of the $i^{th}$-ranked recommended query, and $Z_k$ is a normalization factor so chosen such that the optimal ranking's *NGCG@*$k$ score is 1. Intuitively, the *NDCG@*$k$ score summarizes the relevancy scores of the $k$ highest-ranked recommended queries in such a way that a relevant query contributes more to the score if the query is ranked higher in the commendation list.

Given an input query $\tilde{q}$, let $Y(\tilde{q})$ be a ranked list of recommended queries determined by a recommendation method. We use $rank_j(Y(\tilde{q}))$ to denote the rank of the $j^{th}$ relevant query (score 2) in $Y(\tilde{q})$[7]. For example, if the $1^{st}$ and $2^{nd}$ recommended queries in $Y(\tilde{q})$ are not relevant but the $3^{rd}$ and the $4^{th}$ are, then $rank_1(Y(\tilde{q})) = 3$ and $rank_2(Y(\tilde{q})) = 4$. (If $Y(\tilde{q})$ contains fewer than $j$ relevant queries, $rank_j(Y(\tilde{q}))$ is defined as $\infty$.) We extend the MRR score proposed in [19] to the *MRR@*$h$ score, which is defined as:

$$MRR@h = \sum_{j=1}^{h} \frac{1}{rank_j(Y(\tilde{q}))}. \tag{12}$$

Essentially, the *MRR@*$h$ score summarizes the ranks of the first $h$ relevant queries in the recommendation list $Y(\tilde{q})$ — A larger score indicates that the first $h$ relevant queries are ranked higher in the list. Note that *MRR@*$h$ is bounded by $H_h = \sum_{i=1}^{h} 1/i$, which is the $h$-th harmonic number.

Figures 6 and 7 show the *NDCG@*$k$ and *MRR@*$h$ scores of the recommendation methods. In Figure 7, the optimal values (i.e., $H_h$) are also shown for reference. From the figures, we see that CACB and MMR give relatively poor ranking performance. This is because many of their recommended queries are not relevant and these queries are sometimes ranked high in the recommendation lists, pushing the relevant ones down the lists. The three DQR-based methods generally give better ranking scores than SR. In particular, their *MRR@*$h$ scores are always higher than those of SR. This is due to the fact that they use query concepts and recommend representative queries. These representative queries give high-quality recommendations which lead to more relevant (score 2) top-ranked recommended queries than SR does. Finally, we see that the ranking scores of DQR-OPC is slightly lower than those of DQR and DQR-ND. This is due to the less-than-perfect clustering achieved by the one-pass algorithm, which occasionally leads

[7]We have also considered counting both partially relevant queries (score 1) and relevant queries (score 2) as well. The conclusion drawn is similar so we skip the details of this alternative.

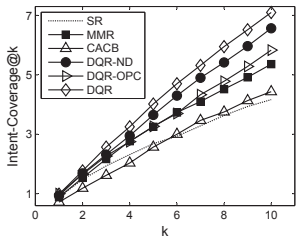


**Figure 5: Intent-Coverage**

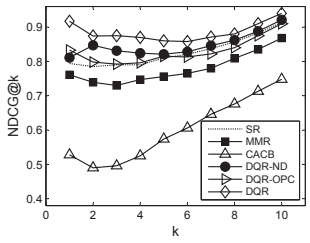


**Figure 6: NDCG**

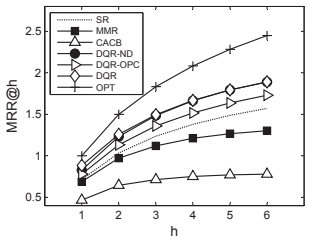


**Figure 7: MRR**

to the recommendation of irrelevant concepts at the top of the recommendation lists.

### *5.5.4 Efficiency*

Recall that the DQR method constructs the selected concept set $Y_c$ incrementally. Specifically, it iterates $m$ times where $m$ is the number of recommended queries, and in each iteration the concept $C$ with the largest $\Delta(Y_c', C, C_{\tilde{q}})$ is picked. To determine $\Delta(Y_c', C, C_{\tilde{q}})$ a sum over the set of all clicked-sets $CS$ is computed (Equation 8). This might sound computationally expensive. Fortunately, the values $p(e_s|e_{C_{\tilde{q}}})$ and $p(e_C|e_s)$ are 0 for most concepts $C$ and click-sets $s$. An index can be built offline to help DQR identify the few non-zero terms of the summation during online query recommendation. The recommendation can thus be done very efficiently. For example, we tested the efficiency performance of DQR written in Java on a Core 2 Duo 2.83GHz PC. In the experiment, 1000 test queries were randomly picked from the search log as input queries. The maximum and the average processing times were 630ms and 51ms, respectively.

## 6. CONCLUSIONS

In this paper we investigated the problem of recommending queries to better capture the search intents of search engine users. We identified five important requirements of a query recommendation system, namely, relevancy, redundancy-free, diversity, ranking-performance, and efficiency. We discussed the weaknesses of existing recommendation methods and proposed DQR. DQR mines a search log to extract query concepts based on which recommendations are made. DQR also employs a probabilistic model to achieve recommendation diversification. Through a comprehensive user study, we showed that DQR performed very well w.r.t. a number of measures that cover the five requirements of a recommender system.

## Acknowledgments

This project was supported by HKU Research Grant 201011159070. Reynold Cheng was supported by the Research Grants Council of Hong Kong (GRF Project 711309E). We would like to thank the anonymous reviewers for their insightful comments.

## 7. REFERENCES

[1] `http://www.cs.hku.hk/research/techreps/document/TR-2012-06.pdf`.

[2] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *EDBT Workshops*, 2004.

[3] R. Baraglia, C. Castillo, D. Donato, F. M. Nardini, R. Perego, and F. Silvestri. Aging effects on query flow graphs for query suggestion. In *CIKM*, 2009.

[4] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *KDD*, 2000.

[5] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

[6] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, 2008.

[7] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.

[8] P.-A. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *SIGIR*, 2007.

[9] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6), 1990.

[10] H. Deng, I. King, and M. R. Lyu. Entropy-biased models for query representation on the click graph. In *SIGIR*, 2009.

[11] B. M. Fonseca, P. B. Golgher, B. Pôssas, B. A. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *CIKM*, 2005.

[12] J. Guo, X. Cheng, G. Xu, and H. Shen. A structured approach to query recommendation with social annotation data. In *CIKM*, 2010.

[13] J. Guo, X. Cheng, G. Xu, and X. Zhu. Intent-aware query similarity. In *CIKM*, 2011.

[14] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4), 2002.

[15] H. Ma, M. R. Lyu, and I. King. Diversifying query suggestion results. In *AAAI*, 2010.

[16] Q. Mei, D. Zhou, and K. W. Church. Query suggestion using hitting time. In *CIKM*, 2008.

[17] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Infoscale*, 2006.

[18] M. Sanderson. Ambiguous queries: test collections need more sense. In *SIGIR*, 2008.

[19] E. M. Voorhees. The TREC-8 question answering track report. In *TREC*, 1999.

[20] X. Wang and C. Zhai. Learn from web search logs to organize search results. In *SIGIR*, 2007.

[21] J.-R. Wen, J.-Y. Nie, and H. Zhang. Clustering user queries of a search engine. In *WWW*, 2001.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD Conference*, 1996.

[23] X. Zhu, J. Guo, X. Cheng, P. Du, and H. Shen. A unified framework for recommending diverse and relevant queries. In *WWW*, 2011.