Query-URL Bipartite Based Approach to Personalized Query Recommendation

Lin Li and **Zhenglu Yang**
Dept. of Info. and Comm. Engineering
University of Tokyo, Japan

Ling Liu
College of Computing
Georgia Tech, Atlanta, GA

Masaru Kitsuregawa
Institute of Industrial Science
University of Tokyo, Japan

Abstract

Query recommendation is considered an effective assistant in enhancing keyword based queries in search engines and Web search software. Conventional approach to query recommendation has been focused on query-term based analysis over the user access logs. In this paper, we argue that utilizing the connectivity of a query-URL bipartite graph to recommend relevant queries can significantly improve the accuracy and effectiveness of the conventional query-term based query recommendation systems. We refer to the **Q**uery-**U**RL **Bi**partite based query re**C**ommendation approach as QUBIC. The QUBIC approach has two unique characteristics. First, instead of operating on the original bipartite graph directly using biclique based approach or graph clustering, we extract an affinity graph of queries from the initial query-URL bipartite graph. The affinity graph consists of only queries as its vertices and its edges are weighted according to a query-URL vector based similarity (distance) measure. By utilizing the query affinity graph, we are able to capture the propagation of similarity from query to query by inducing an implicit topical relatedness between queries. We devise a novel rank mechanism for ordering the related queries based on the merging distances of a hierarchical agglomerative clustering. We compare our proposed ranking algorithm with both naïve ranking that uses the query-URL similarity measure directly, and the single-linkage based ranking method. In addition, we make it possible for users to interactively participate in the query recommendation process, to bridge the gap between the determinacy of actual similarity values and the indeterminacy of users' information needs, allowing the lists of related queries to be changed from user to user and query to query, thus personalizing the query recommendation on demand. The experimental results from two query collections demonstrate the effectiveness and feasibility of our approach.

1. Introduction

Web search engines today provide simple and yet friendly user interfaces which allow users to pose queries simply in terms of keywords. Keyword search is much more popular than SQL queries for Web information access. The main limitation with keyword-based search is two folds. First, some keywords have different meanings in different context, such as mouse trap, Jaguar, Java and so on. Thus it is hard for search engines to return high quality results when only a couple of keywords are used in defining users' queries (Wen, Nie, & Zhang 2002). This is because search engines primarily rely on the matching of the query terms to the document terms in the desired documents to determine which Web pages will be returned given a keyword-based query. Furthermore, users often fail to choose proper terms that best express their information needs. Ambiguous keywords used in Web queries and the limited ability of users to precisely express what they want to search in a few keywords have been widely recognized as a challenging obstacle in improving search quality.

Researches (Cui *et al.* 2003; Collins-Thompson & Callan 2005) have investigated the utilization of query expansion techniques to help users formulate better queries. The idea of exploiting the collaborative knowledge of users, embodied as a set of past search queries, was proposed early (Raghavan & Sever 1995; Glance 2001; Wen, Nie, & Zhang 2002). The main process of these techniques consists of two steps: (1) Finding the terms from queries or documents that are most similar to the current query, and (2) Ranking similar terms and utilizing the ranked similar terms to reformulate the current query. We call this process query recommendation. Existing query recommendation techniques differ from one another in terms of the methods they use to find similar terms and the techniques they use to rank the similar terms for query recommendation. Most of the existing research to date considers previous queries having common terms with the current query to be the similar queries and naturally recommends these queries. However, queries can be phrased with different terms but are meant for the same information need. Raghavan et al. (Raghavan & Sever 1995) have shown that the query result-vectors are a better similarity metric compared to query term-vectors.

One obvious approach to utilizing the query-result information is to represent queries and their result URLs as a bipartite graph with edges connecting queries on one side of the graph to the corresponding URLs of search results returned by a search engine on the other side of the graph. It is clear that related queries can be found by examining the collection of queries and URLs in the query-URL bipartite graph. Queries that are more strongly connected to each other are considered more similar (related). Different approaches can be used for finding similar queries.

From a graph theoretical viewpoint, extracting the collection of most related queries and URLs can be approximated by the problem of partitioning a bipartite graph (Beeferman & Berger 2000; Zha *et al.* 2001) and then finding the maximum biclique, one of the well-known NP complete problems in the literature (Dawande *et al.* 2001). The problem of partitioning a bipartite graph can also be addressed by clustering techniques (Hansen & Shriver 2001; Wen, Nie, & Zhang 2002). After finding the collection of similar queries for a given input query, the next challenge is how to devise a ranking mechanism to order these related queries and making personalized query recommendations. In this paper we present a Query-URL BIpartite based approach to personalized query reCommendation, called QUBIC. In the QUBIC system, the process of finding and ranking similar queries proceeds in two phases: discovery and ranking. In the discovery phase, we construct the query-URL bipartite graph from the query-URL historical collection and find related queries based on the *connectivity* of the graph and the similarity measure between pairs of queries. Instead of using query-term vector model to compute the similarity among queries, we use query-URL vector model to measure the query similarity. In the query-URL vector model, a query is represented in terms of the corresponding set of URLs returned by the search engine in responding to the query, instead of the set of terms used in the query keyword list. In the ranking phase, we argue that the naïve approach that directly uses the similarity score computed in the discovery phase is insufficient. We propose to utilize the monotonicity of the merging distances of a hierarchical agglomerative clustering (HAC) which can capture the diffusive transition of similarity on the affinity graph of queries to rank similar queries discovered in the discovery phase. Furthermore, instead of fixing the degree of similarity of queries based on their query-URL vector similarity, the QUBIC system adaptively controls the output of different lists of related queries in terms of the level of users' satisfaction.

The remainder of this paper is organized as follows. An overview of our approach is described in Section 2. We present the QUBIC two-phase query recommendation algorithm in Section 3 and Section 4 and report our experimental evaluation of the effectiveness of the QUBIC approach in Section 5. The related work and summary are discussed in Section 6 and Section 7 respectively.

2. QUBIC System Overview

The QUBIC system is designed to implement a query-URL bipartite based approach to personalized query recommendation. Figure 1 shows a sketch of the QUBIC system architecture where query recommendation consists of two phases: (1) The discovery of similar queries from the historical query-URL collection (the top red box in Figure 1); and (2) The ranking of similar queries by taking into account both the propagation of the similarity and the subjectivity of the similarity (the bottom red box in Figure 1).

Concretely, the discovery phase consists of the following three steps: (1) constructing query-URL bipartite graph; (2) generating query affinity graph (QAG); and (3) extracting connected components in QAG to form the set of similar

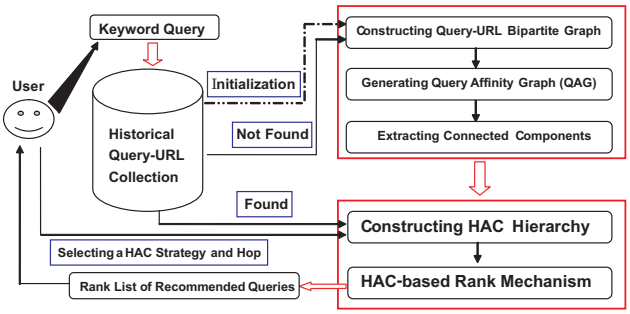


Figure 1: QUBIC architecture

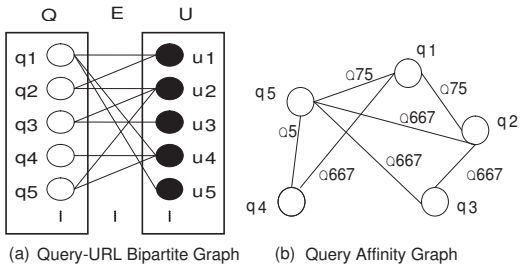
query groups. In the first step, we take the query-URL historical collection and construct a query-URL bipartite graph. Instead of using graph-theoretical approach of finding maximum biclique, in the second step we construct a query affinity graph (QAG) using the query-URL based distance (similarity) measures. We introduce a system defined parameter δ to control the level of query similarity to be considered, thus reducing the set of candidate queries in the historical collection, which are considered most relevant. In the third step of the discovery phase, connected components are extracted from the QAG. Each connected component is seen as a group of queries with high relevance or similarity in terms of their query-URL vector similarity measure.

In the second phase, we need to rank the set of similar queries discovered by the first phase to produce the best personalized query recommendation. The naïve approach to ranking of similar queries is to use the query-URL vector based similarity scores, which fails to properly capture the propagation of similarity in the query affinity graph and the subjectivity of similarity from users' requirements. An alternative approach is to use single-linkage clustering (Beeferman & Berger 2000) to rank the clusters of every pair of queries that are connected in the query affinity graph. The limitation of this approach is due to its severe constraint on the relatedness. Another weakness of selecting queries from the most frequently occurred connected component that contains the input query (keyword list) is the fact that the selected queries may not be the best query recommendation, as the frequency is not always the best descriptor of relatedness because it does not discern the individual targeted queries. We argue that the monotonic merging operations of hierarchical agglomerative clustering (HAC) strategies can better address this problem. In HAC, the pair of queries that has the shortest distance will be merged first. At each remaining step, the next closest pair of queries (or groups) should be merged. The sequence of merge operations scores the relevance of two queries and produces an ordered list of related queries for a specific query.

We would like to note that additional cost might be incurred when the input query is not in the historical query-URL collection, since we need to add this new query and its search results to the original query-URL bipartite graph, recompute the distance scores between the input query with all the queries in the historical collection, and update the affinity graph of queries and the connected components. One of our ongoing research is to devise incremental optimization

Table 1: QUBIC discovery phase

Input: query-URL data in the form of (query, URL).		
Output: connected components (a group of relevant queries).		
1.	If a query q appears with an URL u, place an edge in the bipartite graph (BG) between the corresponding vertices in Q and U;	O(E_{BG})
2.	Compute the distance scores between each pair of queries according to one of two similarity measures;	O(Q^2)
3.	Link every pair of queries with a weighted edge if their distance score is smaller than δ, thus producing the query affinity graph (QAG).	O(E_{QAG})
4.	On the basic initialization of the disjoint-sets structure (Cormen *et al.* 1990), each vertex in QAG is in its own set;	
5.	The connected components are calculated based on the edges in QAG, so update the disjoint-sets structure when each edge is added into the graph;	
6.	Extract the connected components;	O($Q+E_{QAG}$)



(a) Query-URL Bipartite Graph (b) Query Affinity Graph

Figure 2: Query-URL BG and QAG

techniques to reduce the cost of updating the query-URL bipartite graph and the query affinity graph.

3. QUBIC Discovery Phase

We dedicate this section to describe the design of the QUBIC similar query discovery phase as listed in Table 1.

3.1 Constructing Query-URL Bipartite Graph

A simple undirected graph G: =($Q \cup U$, E_{BG}) is called bipartite if Q and U are disjoint sets, where Q and U are the vertex set and E is the edge set of the graph. In the context of QUBIC, the query-URL relationship can be intuitively represented as a bipartite graph (step 1) and hence, it is used as our original model where Q is a set of queries, U is a set of URLs, as shown in Figure 2(a). An edge e connects a query q and an URL u, if the URL u is returned by a search engine on the query q.

3.2 URL-Vector Based Similarity Measures

In QUBIC we model queries in terms of query-URL vectors, instead of query-term vectors. Let $U(q)$ denote the list of y URLs returned as results to the query q, denoted by $u_{q1}, u_{q2}, \cdots, u_{qy}$ and let $w_{q,u}$ be the weighted value associated to the query-URL pair (q, u). Using the above notations, we define the different similarity measures between queries which can be simply transformed to distance measures. We use $similarity$ and $distance$ interchangeably to describe our problem in different contexts. Due to the space constraint, in this paper we discuss two functions that utilize the above URL-vector model to compute the distance between two queries (step 2).

(1)**Jaccard Distance**

Jaccard Distance is an intuitive and popular measure, defined as

$$Jaccard(q_i, q_j) = \frac{U(q_i) \cup U(q_j) - U(q_i) \cap U(q_j)}{U(q_i) \cup U(q_j)} . \quad (1)$$

Here q_i and q_j are queries, $U(q_i)$ and $U(q_j)$ are the two sets of URLs returned by a search engine in response to the two queries q_i and $q_j$respectively. The value of the defined distance between two queries lies in the range [0, 1]: 0 if they are exactly the same URLs, and 1 if they have no URLs in common. For example, in Figure 2(a) where $U(q_1)$=$\{u_1$, u_4, $u_5\}$ and $U(q_2)$=$\{u_1$, $u_2\}$, the Jaccard distance between q_1 and q_2 is 0.75, as shown in Figure 2(b). This definition is convenient to work with, but it suffers from not considering the frequency of a URL that occurs in a query and the number of queries containing the URL in their results. This motivates us to consider the following measure.

(2)**Cosine Similarity**

$w_{q,u}$ represents the weight of the URL u in the query q. It is similar to the traditional ***tf*idf*** weighing method, but is URL-vector based. We define $w_{q,u}$ as:

$$w_{q,u} = (1 + ln(1 + ln(n(q, u)))) * ln(1 + M)/m_u .$$

In this definition $n(q, u)$ is the number of times the URL u occurs in the query q, M is the total number of queries in the query set, and m_u is the number of queries containing u. If we use the full path of a URL, $n(q, u)$ is always 1. If different path levels of a URL are evaluated, $n(q, u)$ could be larger than 1. For example, if the hostname of a URL occurs five times in the search results of a query, we could say that this website can be more informative to represent the query than other websites with lower frequency. The similarity of each pair of query vectors is calculated using the following Cosine formula:

$$Cosine(q_i, q_j) = \frac{\sum_{u=1}^{|U|} w_{q_i,u} * w_{q_j,u}}{\sqrt{\sum_{u=1}^{|U|}(w_{q_i,u})^2}\sqrt{\sum_{u=1}^{|U|}(w_{q_j,u})^2}} , \quad (2)$$

where $|U|$ is the number of URLs. The distance between q_i and q_j is simply computed by $d(q_i, q_j) = 1 - Cosine(q_i, q_j)$. Intuitively, more frequent URLs are more likely to be better indicators for a query; while URLs with higher query frequency might be less informative to represent a distinct query.

3.3 Generating Query Affinity Graph

Based on the query-URL vector based similarity measure for each pair of queries, we can generate the query affinity graph (QAG) with the queries in Q as the vertices. For each pair of vertices, we add an edge in the QAG if their result URL sets have at least one common URL (see Figure 2(b) for an example). Edges in QAG are weighted according to the similarity (distance) measure (Step 3). We define two queries are related if there are paths from one to another. Under this definition, we generalize the concept of relatedness in the sense that given a query, its related queries include adjacent vertices (e.g., q_1 and q_2 in Figure 2(b)) and un-adjacent vertices

(e.g., q_1 and q_3 in Figure 2(b)) of a query as long as they are reachable from the given query following a path in the QAG. Weights will be assigned by using either Jaccard or Cosine similarity measures. In QUBIC system, we use the system supplied parameter δ to remove the edges between queries whose distance scores are larger than δ in the third step of extracting connected components (e.g., $\delta = 0.85$). This helps to keep the overall data within a reasonable size. The extraction of connected components from an undirected graph is calculated in Step 4 $\sim$ Step 6 and readers can refer to (Cormen *et al.* 1990) for detail.

4. QUBIC HAC-based Ranking Phase

The primary task of this phase is to rank the set of candidate queries (connected components) produced in the discovery phase based on their distance to the given input query.

4.1 Our Ranking Methods

We first discuss four ranking methods to produce the ordered list of related queries given an input query. One is simply based on the Jaccard or Cosine similarity measures, called naïve ranking. The other three methods are HAC-based, and their respective clustering process can estimate the relatedness between queries. Our idea is based on the hierarchical structure of HAC where distance measures associated with successive combination operations could be monotonic. If d_1, d_2, $\cdots$, d_k (the definition will be expressed soon) are successive combination distances of an HAC strategy, then $d_1 \leq d_2 \leq \cdots \leq d_k$ must hold. The monotonic property is desirable in our context. However, not all HAC algorithms are monotonic. In the QUBIC design, we consider the following three kinds of monotonic HAC strategies, assuming that (h), (i), and (j) are three groups, containing n_h, n_i, and n_j elements respectively with inter-group distances defined as d_{hi}, d_{hj}, and d_{ij}. We further assume that the smallest of all distances is d_{ij}, such that (i) and (j) fuse to form a new group (k), with n_k (=n_i+n_j) elements.

(1)**Single-linkage strategy (SL):**

$d_{hk} = MIN[d(q_h, q_k)]$.

Single-linkage and complete-linkage strategies are monotonic by definition (Lance & Williams 1966; 1967). The former merges the two clusters with the smallest minimum pairwise distance. The latter is the opposite of the single linkage strategy and unsuitable for our problem. As illustrated in Figure 2(a), there are $d(q_1, q_2) = 0.75$, $d(q_2, q_3) = 0.667$, and $d(q_1, q_3) = 1$. In the complete-linkage strategy, q_1 and q_3 will not be in the same group until all the queries are clustered into a single group. But it is apparent that q_3 is a candidate of related queries to q_1 since there is a path from q_1, q_2, to q_3. Therefore, the single-linkage strategy is considered in our QUBIC system.

(2)**Group-average strategy(GA):**

$d_{hk} = \frac{\sum_{q_h \in n_h} \sum_{q_k \in n_k} d(q_h, q_k)}{n_h n_k}$.

The group-average clustering considers the distance between two clusters is defined as the average of distances between all pairs of objects, where each pair is made up of one object from each group. It is easy to prove that the group-average strategy holds the monotonic property as well. The detailed procedure is not described here due to page limit.

(3)**Flexible strategy (F-α):**

$d_{hk} = \alpha d_{hi} + \alpha d_{hj} + (1 - 2\alpha) d_{ij}$.

Lance et al. (Lance & Williams 1966; 1967) have derived a flexible and monotonic strategy. As α increases from 0 to 1, its hierarchy changes from an almost completely *chained* system into one with increasingly intense clustering.

Table 2: QUBIC HAC-based ranking phase

Input: an HAC strategy chosen by a user and an input query iq.		
Output: a ranking list of related queries to the input query iq.		
7.	If find the the connected component containing the query iq; Else update the discovery phase to reflect inclusion of iq.	
8.	Choose the H-hop neighbors of the input query as candidates of related queries, and store the distance matrix D ($n \times n$ and $n \leq Q$) of candidates (including iq).	O(n^2)
9	Apply the selected HAC strategy on this matrix.	
10.	The successive merging operations of HAC naturally form an ordered list.	
10.1	$M[iq]$: the first merging distance for the input query iq;	O(n)
10.2	For cq=1 to n ($cq \neq iq$) Being Loop	
10.3	$M[cq]$: the first merging distance for the candidate query cq;	O(n)
10.4	$M[(iq, cq)]$: the merging distance for the cluster that include cq and iq;	O(n)
10.5	$R[cq] = \|M[iq] - M[(iq, cq)]\| + \|M[cq] - M[(iq, cq)]\|$; End Loop	O(1)
10.6	Quicksort $R[n]$ BY ASCENT;	O($nlogn$)
10.7	Return the top similar queries but delete the candidate queries whose $D[iq][cq]$ are smaller than 0.2;	O(n)
11.	If the user is unsatisfied with the list, selects another HAC strategy; Go to 9; Else end this searching process.	

4.2 The Design of Our Ranking Procedure

We provide a sketch of our QUBIC HAC-based ranking procedure in the format of pseudo code in Table 2. After a HAC strategy is selected, our algorithm will utilize the process of hierarchical clustering to rank the candidates for query recommendation. Concretely, given n candidate queries, we create an $n \times n$ distance matrix of candidate queries, and apply the chosen HAC strategy on this matrix in three step process (Step 7 $\sim$ Step 9) before we rank them to form the recommendation list (Step 10). Although the main weakness of agglomerative clustering methods is they do not scale well: time complexity of at least O(n^2), the complexity of the clustering and ranking process depend on the number of candidate queries obtained in Step 8 which is smaller than the total number of queries and is typically dependent on the number of hops used in traversing the query affinity graph. If the users want more diverse queries by increasing the number of hops, it will take more time to traverse the affinity graph and to cluster and rank them. The process of the HAC clustering is stored as an n by 2 matrix M which contains the combination distances between merging clusters at the successive stages. Let n be the number of candidates. Row i of the matrix describes the merging of clusters at the step i of the clustering. We show that the successive merging operations of the HAC algorithm naturally form an ordered list. A HAC strategy builds a hierarchical structure where an input query is merged with a group(query) at a distance ($M[iq]$) in the first time (Step 10.1), and the first merging for a candi-

Table 3: Query-URL bipartite graph

Data Set	Queries	Distinct URLs	Edges
QueryKDD	800	26,206	39,599
QueryTREC	10,000	147,761	491,956

date query is at a distance ($M[cq]$) (Step 10.4). In addition, the input query (iq) and the candidate query (cq) will come together at a combination distance ($M[(iq, cq)]$) (Step 10.5). Then, the distance score between the two queries is estimated to $|M[iq]-M[(iq,cq)]|+|M[cq]-M[(iq,cq)]|$ which ranks each candidate (Step 10.6). We empirically show this HAC-based ranking is effective in the Section 5.

A clustering structure produced by our experiments is illustrated in Figure 3 where "Height", the label of the vertical axis, means the combination distance at each merging operation. Using query 1 and query 5 in Figure 3(b) as a concrete example, we know $M[1]$= 0.575, $M[5]$=0.7, and $M[(1,5)]$=0.8. Then, $R[5]$ is 0.375 which may vary with hierarchical structures produced by different HAC strategies.

4.3 Discussions

Our similarity measure between queries is based on the common URLs returned from submitting both queries to a search engine. If these related queries are identical or give an identical result to the original query, then they add no value to the query recommendation quality and effectiveness, and thus cannot improve the level of satisfaction of the users. Naturally, we want to suggest queries that yield comparable search results. One way to approach this goal is to remove the candidates whose distance scores are smaller than a threshold. This ensures that our approach does not try to recommend *exactly similar* queries, but *sub-similar* ones. It is worth noting that in the first prototype of QUBIC we simply choose 0.2 as a filtering threshold, in real application, users can get more similar queries by decreasing the value or obtain less similar ones by increasing it. The reason that we did not apply such filtering in Step 3 of the discovery phase is because these candidates can work as bridge to propagate similarity between queries during the HAC clustering.

5. Experiments

5.1 Data Sets and Evaluation Metric

The experiments use two data sets: QueryKDD and QueryTREC. The former is the 800 queries of KDD cup 2005 data set [1] that are labelled by three people. The latter is the "10k" stream of the Efficiency task topics of the 2006 TREC Terabyte Track [2]. Table 3 summarizes the statistics about the query-URL bipartite graphs of them. We empirically did an analysis on parameters. Due to page limit, only results are reported here. We stored the top 50 of search results for each query, and the values of the threshold δ for QueryKDD and QueryTREC are 0.98 and 0.85 respectively. H is set as 3 to keep the overall data within a reasonable size. The performance is evaluated by the precision measure called $P@N$ which calculates the percentage of the related queries at the top N results.

[1] http://www.acm.org/sigs/sigkdd/kddcup/index.php

[2] http://trec.nist.gov/data/terabyte06.html

Table 4: $P@10$ of QueryKDD data set

	Naïve	SL	GA	F-$\alpha(0.5)$
Jaccard	0.4015	0.3868	0.4549	0.4416
Cosine	0.4482	0.4397	0.4675	0.4575

Table 5: $P@10$ of QueryTREC data set

	Naïve	GA
Jaccard	0.43	0.56
Cosine	0.64	0.80

5.2 Results and Discussions

Experiments on the QueryKDD Data Set For precision evaluation, if two queries share at least one category, we regard them as related queries. The experimental results averaged by three labelers are shown in Table 4. The single-linkage strategy merges in each step the two clusters whose two closest members have the smallest distance. Its main weakness is that it is sensitive to outliers, which leads to the fact that single-linkage strategy performs worst among the four rank mechanisms (naïve, single-linkage, group-average, and flexible(α=0.5) strategies). While the group-average strategy is robust to outliers because it considers the distance between one cluster and another cluster to be equal to the average distance from any member of one cluster to any member of the other cluster. The flexible($\alpha = 0.5$) strategy computes the sum of distance between members of clusters and is stable as well. Both of them got higher precision scores than the naïve one under both two similarity measures. The Cosine measure produces better results than the Jaccard measure in all four ranking methods. The group-average strategy with the Cosine measure is the optimal combination in terms of precision@10. These results prove that our URL-vector based $if * idf$ weighing method is more effective than the un-weighing Jaccard measure.

Experiments on the QueryTREC Data Set Different from the QueryKDD data set, the QueryTREC data set does not supply us with the categorization information. Three human evaluators are invited to judge the performance of our algorithm. After running the QUBIC two-phase algorithm with the group-average strategy, we can recommend a list of related queries for each query. We then randomly selected 30 queries as our test data. The three evaluators worked separately without knowing how our detection algorithms work. Queries are manually tagged as either related or unrelated. Their averaged results are reported in Table 5. The Cosine measure achieves much better performances than the Jaccard measure, and the HAC-based ranking obtains higher precision scores than the naïve ranking. The two results are consistent with the result of QueryKDD data set. Therefore, we conclude that the Cosine measure using the conventional $if * idf$ term weights of documents is effective to represent the queries in a URL-vector space model, and HAC-based ranking is more effective for query recommendation than the naïve ranking.

In Figure 3, as α increases from 0.25 to 0.45 in the Flexible-α strategy, the hierarchy changes. Thus, it is obvious that the ordering in the list of related pages partly changes as well. In our personalization scheme, if a user inputs a query to retrieve Web pages, our algorithm will supply

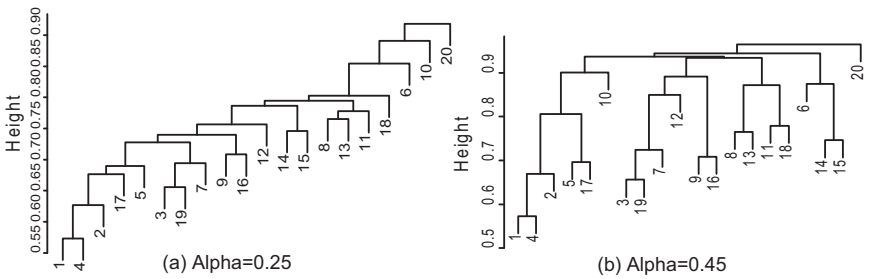


Figure 3: Effect of varying HAC strategies. Measure: Cosine; Height: the combination distance at each merging.

her with a list of related queries from historical collection. Then, the user can formulate her query and do a new search, or she may also ask our algorithm to produce a new list of related queries by selecting a different HAC strategy. This process may be repeated until the user is satisfied with the search results. The naïve ranking cannot supply users with such diverse lists of query recommendation due to its locality and lack of effective similarity propagation.

6. Related Work

Utilizeing a set of past search queries, was proposed early (Raghavan & Sever 1995; Glance 2001; Wen, Nie, & Zhang 2002). Treating the top ranked documents as a special case of relevance feedback is a variation of the original work on local feedback (Xu & Croft 1996). Wen et al. (Wen, Nie, & Zhang 2002) proposed to cluster similar queries to recommend URLs to frequently asked queries of a search engine. Hansen et al. (Hansen & Shriver 2001) distilled search-related navigation information from proxy logs to cluster queries. The query-URL relationship can be represented by a bipartite graph as modelled in Section 3. Graph partitioning is an alternative for grouping (Zha *et al.* 2001; Rege, Dong, & Fotouhi 2006) which is done by cutting the set of vertices into disjoint sets. The drawbacks of the approach proposed by Beeferman et al. (Beeferman & Berger 2000) have already been discussed in Section 2. Different from the above studies, our approach makes use of the HAC monotonicity to rank related queries based on query-URL bipartite, which is a combination of clustering-based and graph-based methods to capture the diffusive transition of similarity. Furthermore, we propose a personalization framework to adaptively rank the collection of related queries according to users' needs.

7. Conclusions

We have presented QBUIC – a query-URL bipartite graph based approach to personalized query recommendation. Our approach recommends related queries to a given query by analyzing the query-URL history through a novel two-phase algorithm (i.e, discovery and HAC-based ranking phases). The experimental results demonstrate the usefulness of graph effectiveness and the feasibility of our approach in helping users optimally represent their information needs by keyword queries. Our research continues along several dimensions, including optimization for incremental updates of our query-URL bipartite graph and query affinity graph maintenance and more comprehensive user study in terms of usability and performance of our QUBIC system.

References

Beeferman, D., and Berger, A. L. 2000. Agglomerative clustering of a search engine query log. In *KDD*, 407–416.

Collins-Thompson, K., and Callan, J. 2005. Query expansion using random walk models. In *CIKM*, 704–711.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 1990. Introduction to Algorithms. McGraw-Hill.

Cui, H.; Wen, J.-R.; Nie, J.-Y.; and Ma, W.-Y. 2003. Query expansion by mining user logs. *IEEE Trans. Knowl. Data Eng.* 15(4):829–839.

Dawande, M.; Keskinocak, P.; Swaminathan, J. M.; and Tayur, S. 2001. On bipartite and multipartite clique problems. *J. Algorithms* 41(2):388–403.

Glance, N. S. 2001. Community search assistant. In *IUI*, 91–96.

Hansen, M., and Shriver, E. 2001. Using navigation data to improve ir functions in the context of web search. In *CIKM*, 135–142.

Lance, G. N., and Williams, W. T. 1966. A generalized sorting strategy for computer classifications. *Nature* 212:218.

Lance, G. N., and Williams, W. T. 1967. A general theory of classificatory sorting strategies: 1. hierarchical systems. *The Computer Journal* 9:373–380.

Raghavan, V. V., and Sever, H. 1995. On the reuse of past optimal queries. In *SIGIR*, 344–350.

Rege, M.; Dong, M.; and Fotouhi, F. 2006. Co-clustering documents and words using bipartite isoperimetric graph partitioning. In *ICDM*, 532–541.

Wen, J.-R.; Nie, J.-Y.; and Zhang, H. 2002. Query clustering using user logs. *ACM Trans. Inf. Syst.* 20(1):59–81.

Xu, J., and Croft, W. B. 1996. Query expansion using local and global document analysis. In *SIGIR*, 4–11.

Zha, H.; He, X.; Ding, C. H. Q.; Gu, M.; and Simon, H. D. 2001. Bipartite graph partitioning and data clustering. In *CIKM*, 25–32.