J Intell Inf Syst (2013) 40:555–587
DOI 10.1007/s10844-013-0237-8

QUBIC: An adaptive approach to query-based recommendation

Lin Li · Luo Zhong · Zhenglu Yang · Masaru Kitsuregawa

Received: 26 December 2011 / Revised: 31 January 2013 / Accepted: 31 January 2013 /
Published online: 24 February 2013

Abstract Search engine users often encounter the difficulty of phrasing the precise query that could lead to satisfactory search results. Query recommendation is considered an effective assistant in enhancing keyword-based queries in search engines and Web search software. In this paper, we present a **Q**uery-**U**RL **Bi**partite based query re**C**ommendation approach, called QUBIC. It utilizes the connectivity of a query-URL bipartite graph to recommend related queries and can significantly improve the accuracy and effectiveness of personalized query recommendation systems comparing with the conventional pairwise similarity based approach. The main contribution of the QUBIC approach is its three-phase framework for personalized query recommendations. The first phase is the preparation of queries and their search results returned by a search engine, which generates a historical query-URL bipartite collection. The second phase is the discovery of similar queries by extracting a query affinity graph from the bipartite graph, instead of operating on the

The work was done when Lin Li was a Ph.D. student at Kitsuregawa Lab., University of Tokyo, Japan.

This research was undertaken as part of Project 61003130 funded by National Natural Science Foundation of China and and Project 2011CDB254 founded by Ministry of Education of China.

L. Li (✉) · L. Zhong
School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China
e-mail: cathylilin@whut.edu.cn

L. Zhong
e-mail: zhongluo@whut.edu.cn

Z. Yang · M. Kitsuregawa
Institute of Industrial Science, University of Tokyo, Tokyo, Japan

Z. Yang
e-mail: yangzl@tkl.iis.u-tokyo.ac.jp

M. Kitsuregawa
e-mail: kitsure@tkl.iis.u-tokyo.ac.jp

original bipartite graph directly using biclique-based approach or graph clustering. The query affinity graph consists of only queries as its vertices and its edges are weighted according to a query-URL vector based similarity (dissimilarity) measure. The third phase is the ranking of similar queries. We devise a novel rank mechanism for ordering the related queries based on the merging distances of a hierarchical agglomerative clustering (HAC). By utilizing the query affinity graph and the HAC-based ranking, we are able to capture the propagation of similarity from query to query by inducing an implicit topical relatedness between queries. Furthermore, the flexibility of the HAC strategy makes it possible for users to interactively participate in the query recommendation process, and helps 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 adaptively recommending related queries on demand. Our experimental evaluation results show that the QUBIC approach is highly efficient and more effective compared to the conventional query recommendation systems, yielding about 13.3 % as the most improvement in terms of precision.

Keywords HAC strategy · Monotonicity · Bipartite graph · Query recommendation

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-fold. 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 (Jansen et al. 1998). 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.

Query recommendation Commercial search engines give suggestions on the queries input by users, thus assisting them in rephrasing their query formulation to improve search quality, such as *Related search terms* in Google and *Search Assist* in Yahoo!. These services supplied by Google and Yahoo! highlight the importance of query recommendation which is considered an effective assistant in enhancing keyword-based queries in search engines and Web search software.

The main process of query recommendation 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, such as appending terms to the existing list of terms in the query, replacing some terms in the query and so on. 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. Finding candidate terms from documents is widely used in query expansion techniques (Collins-Thompson and Callan 2005; Cui et al. 2003; Qixia and Maosong 2011). However, some terms are difficult to be suggested because of their high document frequencies, e.g., "good", "delicious" and so on. If these terms appeared in some past queries, we can easily suggest them to users by using the whole query. Terms in related queries can also be an effective source for query recommendation. Recent researches are following this direction of utilizing a set of past search queries which embodies the collaborative knowledge of users (Aris et al. 2010; Baeza-Yates et al. 2007; Glance 2001; Wen et al. 2002; Xiaohui et al. 2011). *Related search terms* in Google is based on the assumption that sometimes the best search terms for what a user is looking for are related to the ones the user actually entered. It is intuitive to think that previous queries having common terms with the current query to be the similar queries and these queries are naturally recommended.

Web users, however, typically submit very short queries to search engines (Jansen et al. 1998), the very small term overlap between queries cannot accurately estimate their relatedness. For example, queries phrased in the same list of keywords may be meant for different results by different users. Furthermore, queries can be phrased with different terms but are meant for the same information need. As a concrete example, we take two queries, "IRS (Internal Revenue Service) form" and "file taxes online". Although they have no terms in common, both of them concern the application of filing taxes. The relatedness between the two queries can be induced from the overlap of the two lists of search results (URLs) returned. Thus, the query result-vectors are often a better similarity metric compared to query term-vectors (Raghavan and Sever 1995).

Utilization of Query-URL bipartite graph From the search result pages we usually can get two kinds of feature spaces, i.e, content-sensitive (e.g., nouns) and content-ignorant (e.g., URLs) which can be used to enrich the expressions of search queries. Then, the relatedness between search queries can be estimated on their enriched expressions, e.g., the overlap of their feature spaces. Researches (Baeza-Yates et al. 2007; Wen et al. 2002; Li et al. 2010) show that the URL feature space produces lower precision scores than the noun feature space which, however, is not applicable, at least in principle, in settings including: non-text pages like multimedia (image) files, Usenet archives, sites with registration requirement, and dynamic pages returned in response to a submitted query and so forth. It is crucial to improve the quality of the URL (content-ignorant) feature space since it is generally available in all types of Web pages. The problem of the URL feature space is that even though two queries share no common URLs in their search result pages, they may be related since Web pages with different URLs may be semantically related. In this paper, we propose an approach to solve this problem.

To utilize the query result-vectors, one obvious approach 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 a collection of 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 and 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 (Baeza-Yates et al. 2007; Wen et al. 2002; Hansen and Shriver 2001). Both bicliques and clusters are groups of related queries. Queries in different groups are regarded as unrelated, or at least much less related than queries in a same group. After finding groups of similar queries, the next challenge is, given a query, how to devise a ranking mechanism to order the related queries in a group, and making query-based recommendations.

Scope of this paper This paper presents a Query-URL Bipartite based approach to query reCommendation, called QUBIC. The QUBIC system fulfills two tasks: (1) finding groups of related queries, and (2) ranking queries in the same group as an input query by proceeding in three phases. The first phase is the preparation of queries and their search results returned by a search engine for constructing a query-URL bipartite graph. 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, thus realizing query result-vectors.

On the second phase, we generate query affinity graph based the similarity measure between pairs of queries. We use query-URL vector model to measure similarity between queries, instead of using query-term vector model. We argue that the query-URL vector better represents the meaning of a query, given the frequency and the amount of Web queries either phrased in the same list of keywords but meant for different results by different users, or phrased with different terms but meant for the same information need. Connected components can be extracted from the query affinity graph and each connected component is a group of related queries, thus completing the first task of the QUBIC system.

In the third phase, we discuss that it is insufficient that the naïve approach directly uses the similarity scores of pairs of queries computed in the second phase. Such naïve approach is widely adopted in the literature of mining related or similar items (e.g., queries, pages, etc.) (Baeza-Yates et al. 2007; Glance 2001; Wen et al. 2002; Calado et al. 2006; Otsuka and Kitsuregawa 2006). We propose to utilize the monotonicity of the merging distances of a hierarchical agglomerative clustering (HAC). By using HAC-based algorithms we can globally capture the diffusive transition of similarity on each connected component to rank queries. 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.

Contribution of this paper In a summary, the contribution of this paper is listed as follows.

1. We generate query affinity graph (QAG) based the similarity measure between pairs of queries. Traditional naïve approaches just find related queries which have a edge connected with the target query in QAG, i.e., adjacent nodes. Our proposed approach tries to get its related queries from adjacent nodes and

un-adjacent nodes as long as they are reachable from the target query following a path.
2. A novel tree distance based ranking is proposed for query recommendation by making use of the dendrogram of clustering to reflect the global similarity through QAG.
3. We give empirical evidence as to how different HAC strategies affect the quality of recommendation and use the flexibility of the α-F HAC strategy to allow users to interactively participate in the query recommendation process.

The remainder of this paper is organized as follows. We discuss the related work in Section 2. An overview of our approach is presented in Section 3. We describe the three phases of the QUBIC adaptive approach for query recommendation in Sections 4, 5, and 6. An extensive experimental evaluation on the effectiveness of the QUBIC approach is reported in Section 7. The conclusion of the paper is in Section 8.

2 Related work

Finding related queries for query recommendation has become an important topic, motivated by several research challenges including query clustering, query expansion, and different graph partitionings. A considerable amount of research has focused on the three aspects. We summarize a small part of these researches.

2.1 Query clustering

Clustering queries submitted to search engines appears to be less explored than clustering Web pages or documents. The idea of exploiting the collaborative knowledge of users, embodied as a set of past search queries, was proposed early (Glance 2001; Raghavan and Sever 1995; Wen et al. 2002; Fitzpatrick and Dent 1997). Glance (2001) introduced a software agent that collects queries from previous users, and determined the query similarity based on the Web pages returned by queries, and not the actual terms in the queries themselves. The goal of Raghavan and Sever (1995) was to accelerate the formation of optimal queries from past queries. Fitzpatrick and Dent (1997) improved the effectiveness of a user-supplied query by identifying key terms from potentially relevant documents from past queries. Treating the top ranked documents as a special case of relevance feedback is a variation of the original work on local feedback (Xu and Croft 1996). Wen et al. (2002) proposed to cluster similar queries to recommend URLs to frequently asked queries of a search engine. They combined similarities based on query contents and user clicks, and regarded user clicks as an implicit relevance feedback instead of using the top ranked Web pages. Hansen and Shriver (2001) distilled search-related navigation information from proxy logs to cluster queries. The data they relied on differed from those used in the above other studies.

Different from the utilization of clustering in these studies, we make use of the monotonicity of the HAC strategies to adaptively output the ordered list of related queries. In addition, we examine three URL-based similarity measures analytically and empirically to provide better understanding of the propagation of similarity from query to query by inducing an implicit topical relatedness between queries.

2.2 Query expansion

We design a query recommendation system which suggest related queries to help users refine their original queries, while query expansion is also an alternative to revise users' queries. The conventional research efforts on query expansion are classified into three categories according to the information sources they use to find relevant terms for query expansion: (1) corpus-based statistics analysis (Voorhees 1994), (2) relevance feedback based recommendation (Salton and Buckley 1990), and (3) local context based recommendation (Buckley et al. 1994; Xu and Croft 2000). In addition, some researches combined multiple sources of knowledge on term associations (Collins-Thompson and Callan 2005; Vuong and Tru 2010; Chirita et al. 2007; Sun et al. 2006).

Others used Web logs to bridge the term gap between user-centric query space and author-centric Web page space (Cui et al. 2003; Yunlong et al. 2011; Zhu and Gruenwald 2005). Most query expansion techniques suggest terms used extracted from Web pages. However, some terms are difficult to be suggested because of their high document frequencies. We think that if these terms appear in some past queries, we recommend them to users by using the full term list of a query. Therefore, terms in related queries can also be an effective source of expansion terms. Yunlong et al. (2011) proposed a Two-Stage SimRank (called TSS in this paper) algorithm based on SimRank (Jeh and Widom 2002) and some clustering algorithms to compute the similarity among queries, and then use it to discover relevant terms for query expansion. Further experiments on query expansion using related queries would be an interesting topic in our future work.

2.3 Graph theory

The query-URL relationship can be represented by a bipartite graph as modelled in Section 4. Finding biclique is a natural way of collecting the most related queries and URLs. A well known problem related to biclique is the maximum clique, which is one of the most widely studied NP-complete problems in the literature (Dawande et al. 2001). On the other hand, graph partitioning is an alternative for grouping (Zha et al. 2001; Rege et al. 2006) which is done by cutting the set of vertices into disjoint sets. Beeferman and Berger (2000) viewed the click-through data as a bipartite graph, and utilized an iterative, agglomerative clustering algorithm to the vertices of the graph for clustering queries and URLs, respectively. Their method just extracted connected components from the entire graph and used *frequency* to measure the similarity between queries. The limitation of this method is the weakness of selecting queries from the most frequently occurred connected component that contains the input query (keyword list). However, 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.

Our QUBIC approach can be regarded as a combination of traditional clustering strategy and graph analysis, which makes use of the dendrogram of clustering to reflect the global similarity through a similarity graph and propose a novel tree distance based ranking for query recommendation. SimRank (Jeh and Widom 2002) measured the object-to-object relationship by recursively scoring the similarity of their related objects. Sun et al. (2005) employed random walk with restarts and graph

partitioning to solve two problems, neighborhood formation and anomaly detection. These studies reflect the global similarity since they work by iteratively transmitting weights through the whole graph. However, they supplied all the users with the same list of related queries in response to an input query. Namely, they ignored the subjectivity of similarity, especially the users' diverse needs. Our approach can adaptively output the recommendation list of related queries according to users' needs.

3 QUBIC system overview

The QUBIC system is designed to implement an adaptive approach to query recommendation based on query-URL bipartite graph. Figure 1 shows a sketch of the QUBIC system architecture. As illustrated in the rightmost three boxes in Fig. 1, the proposed query recommendation approach consists of query-based recommendation preparation to build the query-URL bipartite graph (First Phase), generating query affinity graph to discovery groups of similar queries (Second Phase), and the ranking of similar queries by taking into account both the propagation of the similarity and the subjectivity of the similarity (Third Phase).

Because the queries other users previously entered may be related to the information need of current user, finding related queries will give hints to an individual user. In the first phase, for each query in a set of past queries, we retrieve the top N search results from a search engine to represent this query (the parameter N will be studied in experiment part). The query-URL bipartite graph is built by creating edges between a query and its corresponding search results, as shown in Fig. 2. Although

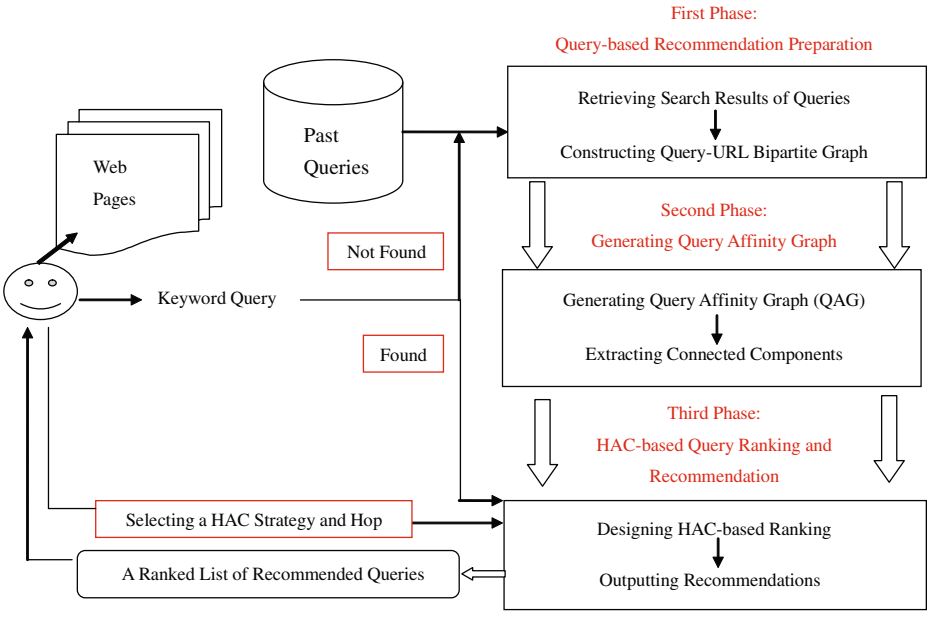


Fig. 1 QUBIC architecture

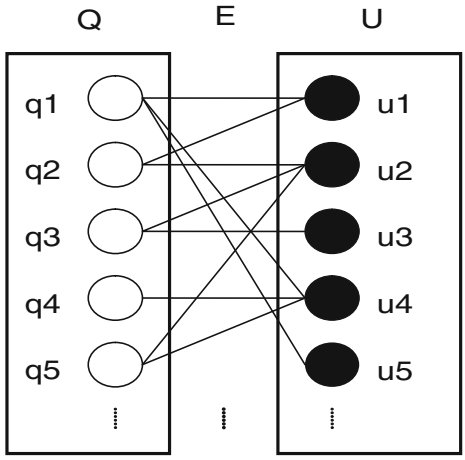


Fig. 2 Query-URL bipartite graph

it is intuitive to represent queries by their terms, we have argued that this approach fails to handle several typical cases in Web queries, for example, queries using the same keyword list but meant for different set of results by different users and queries with different terms but for the same information needs.

The second phase generates query affinity graph (QAG) and extracts connected components from QAG, thus forming groups of similar queries. Instead of using graph-theoretical approach of finding maximum biclique on the bipartite graph built in the first phase, we construct a query affinity graph (QAG) using a query-URL based dissimilarity (similarity) measure. A system defined parameter δ is introduced to control the level of query similarity to be considered, thus reducing the set of candidate queries in QAG for considering most relevant ones. Last, connected components are extracted from QAG. Each connected component is regarded as a group of queries with high relevance or similarity in terms of their query-URL vector similarity measure.

In the third phase, given an input query, we need to rank queries in the same group as the input query, to produce the best adaptive query recommendation. The naïve approach to ranking 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. We generalize the concept of relatedness in the sense that given the an input query in QAG, its related queries include adjacent nodes and un-adjacent nodes as long as they are reachable from the given query following a path in QAG. The monotonic merging operations of hierarchical agglomerative clustering (HAC) strategies generate the dendrogram (also called cluster tree) for displaying the cluster membership between data points. Based on this kind of tree structure, we argue that a tree distance based ranking can better address the problem of similarity propagation. Moreover, the flexibility of a HAC strategy allows the lists of related queries to be changed from user to user and query to query, thus adaptively recommending related queries on demand.

The general work flow of our system with the three phases consists of initialization and recommendation processings. Because we have a collection of queries, the

system initialization processing only needs to apply the first and second phases to these queries and obtain groups of related queries. In the recommendation processing, a user inputs a keyword query to our system. If the input query can be found in one of these groups of related queries, the third phase ranks each query in this group and generates a recommendation list. If the input query cannot be found, the whole three phases are executed one by one. Thus, additional cost is incurred. We need to add this new query and its search results to the original query-URL bipartite graph, re-compute the similarity (dissimilarity) scores between the input query with all the queries in the query collection, and update the query affinity graph. We will discuss some optimization techniques to reduce the updating cost, especially when updating the query affinity graph in Section 5.

We can see that our system returns **a list of recommended queries**. In real applications, we expect the system to return **informational web pages as results** using the recommended queries, as shown in the rightmost part of Fig. 1. In our published paper (Li et al. 2011), we address the Web search problem on aggregating search results of related queries to improve the retrieval quality. Given an initial query and the suggested related queries, we concurrently processes their search result lists from an existing search engine and then forms a single list aggregated by all the retrieved lists. Some experimental results are reported in the last paragraph of Section 7.

4 First phase: query-based recommendation preparation

Web search engines are useful to find alternative information sources for more accurate representation of a query than query-term vectors. Given a query, a Web search engine will return relevant Web pages (search results) according to its own rank mechanism. Many advanced Web searching techniques have been developed and used in commercial Web search engines, e.g., Google and Yahoo!, which can retrieve search results with high relevance to an input query, but also high page quality like Pagerank (Page et al. 1998) and HITS (Kleinberg 1999). Therefore, we are interested in the top search results in a retrieved list after submitting a query to a search engine. From these returned search results, usually we can extract their contents and their URLs to enrich the query. The content of a Web page, however, is not applicable, at least in principle, in settings including: non-text pages like multimedia (image) files, Usenet archives, sites with registration requirement, dynamic pages returned in response to a submitted query and so forth. In this paper we study the utilization of the URLs of returned search results to represent a query since URLs are generally available in all types of Web pages.

Now a query is associated with the URLs of the retrieved search results. In the query-based recommendation preparation of our QUBIC system, the query-URL relationship can be intuitively represented as a bipartite graph. A bipartite graph, also called a bigraph, is a special graph from which the set of vertices can be decomposed into two disjoint sets such that no two vertices within the same set are adjacent. In the mathematical definition, a simple undirected graph $G := (Q \cup U, E)$ 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. This graph is used as our original model where Q is a set of queries, the U is a set of URLs, as shown in Fig. 2. An edge e connects a query q and an URL u, if the URL u is returned by a search engine on the query

Table 1 First phase—query-based recommendation preparation

Input: a set of queries **Output:** query-URL bipartite graph		
1.	Submit each query to a search engine (e.g., Google) and get the top N search results returned;	$O(\|Q\|)$
2.	Extract the URLs of all the search results	$O(\|U\|)$
3.	If a query q appears with an URL u in the set of its search results, place an edge in the bipartite graph (BG) between the corresponding vertices in Q and U;	$O(\|E_{BG}\|)$

q. In the context of QUBIC, we propose to recommend queries based on the inter-relationship of their corresponding URLs. As a by-product, related Web pages could be mined as well by applying the proposed algorithm on the side of the URLs with a relatively small modification.

The pseudo code of this phase is listed in Table 1. The run time cost of this phase is mainly dependent on the first step since search results have to be downloaded from a search engine. Downloading time is the average time cost of the response speed of a search engine and the network delay. The whole first phase can be done as a preparation for the next two phases. Even if an input query is not in the current query set, we only need add the query as a node to Q and the URLs of its search results to U. The total number of added edges is the number of returned search results.

5 Second phase: generating query affinity graph

In this section we describe the design of the generating query affinity phase of QUBIC, which applies a URL-based similarity (dissimilarity) measure for all pairs of queries. The pseudo code of this phase algorithm is listed in Table 2.

5.1 URL-vector based similarity measures

As discussed in the first phase, the query q in Q is represented by the list of URLs returned as search results. Furthermore, let $w_{q,u}$ be the weighted value associated to the query-URL pair (q, u). $w_{q,u}$ is 0 if u is not included in the list of URLs returned by a search engine in response to the query q, otherwise $w_{q,u}$ is 1 in an unweighted strategy. $w_{q,u}$ can also be in the range [0,1], determined by different weighting strategies in different similarity (dissimilarity) measures. Therefore, the query q is denoted by $w_{q,u_1}, w_{q,u_2}, \cdots, w_{q,u_{|U|}}$ dissimilarity where $|U|$ is the number of URLs in U. Using the notations described above, we define the different similarity

Table 2 Second phase—generating query affinity graph

Input: query-URL bipartite graph **Output:** connected components		
4.	Compute the dissimilarity scores between each pair of queries according to one of five similarity measures;	$O(\|Q\|^2)$
5.	Link every pair of queries with a weighted edge if their dissimilarity score is smaller than δ, thus producing the query affinity graph (QAG).	$O(\|E_{QAG}\|)$
6.	Extract connected components from QAG;	$O(\|Q\| + \|E_{QAG}\|)$

measures between queries. Although they are called similarity measures, they, in fact, can be used as dissimilarity scores by a simple transformation. We use *similarity* and *dissimilarity* interchangeably to conveniently describe our problem in different contexts. In this paper we discuss three dissimilarity functions that utilize the URL-bipartite graph to compute the distances between two queries, quite different from the term-based traditional measures.

All the three dissimilarity functions rely on the overlap of the search result URLs of two queries with unweighting or weighting function on each URL. We can utilize the different path levels of a URL to get the overlap. Generally, using the full URL of a search result will result in smaller overlap and higher precision than using its hostname. We experimentally calculated the overlap of hostnames of top 10 search results between queries and about 80 % of the pairs of queries had no common hostnames (details in Section 7.2). The situation is even worse when using the full URL. We can increase the number of top search results to get larger overlap of full URLs, but it will degrade the precision of measuring the query-to-query dissimilarity (similarity) since the top search results are better dictators of a query than the bottom ones. Therefore, we use the hostname of a URL to represent a search result, instead of its full path. From now the denotation URL means *hostname* if no specific explanation.

(1) Jaccard similarity

Jaccard similarity is an intuitive and popular measure, defined as

$$Jaccard(q_i, q_j) = \frac{|P(q_i) \cap P(q_j)|}{|P(q_i) \cup P(q_j)|}. \tag{1}$$

Here q_i and q_j are queries, $P(q_i)$ and $P(q_j)$ 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 dissimilarity between two queries lies in the range [0, 1]: 1 if they return exactly the same URLs, and 0 if they have no URLs in common.

For example, if $P(q_1) = \{u_1, u_3, u_4\}$ and $P(q_2) = \{u_1, u_2\}$, the Jaccard similarity between q_1 and q_2 is 0.25. We use $d(q_i, q_j) = 1 - Jaccard(q_i, q_j)$ as Jaccard dissimilarity for the clustering computation in the third phase of our system. This definition is an unweighted strategy for URLs which only count the number of commmon URLs shared by two queries. To consider the frequency of u that occurs in a query and the number of queries containing u in their results, we propose the following two different weighting methods.

(2) L_1 norm

Specifically, the weighting function for each URL is defined as:

$$w_{q,u} \equiv p(u|q) = \frac{n(u|q)}{\sum_{u \in U} n(u|q)}, \tag{2}$$

where $n(u|q)$ is the number of occurrences of the URL u in the query q. For example, if a URL (hostname) occurs five times in the search results of a query, this measure thinks that this website can be more informative to represent the query than other websites with lower frequency. Each query has its own URL (hostname) distribution because of different search results.

In document clustering, a natural measure of similarity of two documents is the similarity between their word conditional distributions. Roughly speaking, documents with similar conditional word distributions would belong to the same cluster. This idea was first introduced in Pereira et al. (1993) and was called "distributional clustering". Similarly, we would like to recommend related queries with similar conditional URL distributions. L_1 norm (or the variational dissimilarity) is common to measure the distancedissimilarity between distributions, defined as

$$L_1(p(u|q_i), p(u|q_j)) \equiv \frac{\sum_{u\in U} |p(u|q_i) - p(u|q_j)|}{\sum_{u\in U} p(u|q_i) + \sum_{u\in U} p(u|q_j)}. \tag{3}$$

We also have experimentally evaluated Ward and Jensen-Shannon (JS) divergence (Lin 1991) measures which showed no better performance than L_1 norm on average. Therefore, we only discuss the L_1 norm in this paper. However, popular sites like news websites or spam websites may be returned within the search results of a query, which will be noisy irrespective of the content of the query. If two queries both retrieve such sites, some relatedness between them will be deduced, even though they may be completely unrelated. In the following we put forward a URL-based cosine similarity which scales down the coordinates of hostnames that occur in many result lists of queries.

(3) Cosine similarity

Similar to the traditional *tf*idf* weighing method,[1] a URL-vector based $w_{q,u}$ is defined as:

$$w_{q,u} = (1 + \ln(1 + \ln(n(q,u)))) * \ln(1 + |Q|)/m_u, \tag{4}$$

where $n(q,u)$ is the number of times the URL u occurs in the query q, $|Q|$ is the total number of queries in the query set, and m_u is the number of queries containing u. The similarity of each pair of query vectors is calculated using the following cosine formula:

$$Cosine(q_i, q_j) = \frac{\sum_{u=1}^{|U|} w_{q_i,u} * w_{q_j,u}}{\sqrt{\sum_{u=1}^{|U|} \left(w_{q_i,u}\right)^2} \sqrt{\sum_{u=1}^{|U|} \left(w_{q_j,u}\right)^2}}. \tag{5}$$

$d(q_i, q_j) = 1 - Cosine(q_i, q_j)$ is the dissimilarity between q_i and q_j. Intuitively, more frequent URLs (hostnames) are more likely to be better indicators for a query ((2) in L_1 norm); while URLs (hostnames) with higher query frequency might be less informative to represent a distinct query, such as the homepages of popular news websites, link directory and so on.

In L_1 norm and *Cosine*, u is weighted based on its occurrences in a query and its query frequency. For each q and u, $w_{q,u}$ can easily be valued by the rank of u for q. Using the rank values of URLs will take into account the ordering of search results when computing similarity scores. Our current implementation weight methods do not yet consider the rank values of search results. We will discuss this problem further in Section 6.3.

[1] http://en.wikipedia.org/wiki/Tf*idf

5.2 Generating query affinity graph

For each pair of queries, we add an edge in the query affinity graph (QAG) with the queries in Q as the nodes if their result URL sets have at least one common URL (see Fig. 3 for an example). Then, we compute a dissimilarity (similarity) score for each pair of queries in QAG using one of the above three measures. The edge between two queries is weighted by the dissimilarity score (Step 5). In QUBIC system, we use the system dissimilarity threshold δ (e.g., $\delta = 0.85$) to remove the edges between queries whose dissimilarity scores are larger than δ when extracting connected components (step 8). This helps to keep the overall data within a reasonable size. The extraction of connected components from an undirected graph is calculated in Step 6. On the basic initialization of the disjoint-sets structure (Cormen et al. 1990), each node in QAG is in its own set. 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. Readers can refer to Cormen et al. (1990) for detail. The time complexity for calculating the connected components is only slightly larger than $O(|Q| + |E_{QAG}|)$ where $|Q|$ is the number of queries and $|E_{QAG}|$ is number of edges in the QAG.

Now we can see that in Fig. 3 queries are connected by weighted edges. The related queries of the query q_2 are q_1 and q_4, which are adjacent nodes to q_2 in the query affinity graph. The similarity score between un-adjacent nodes (e.g., q_2 and q_3) is zero, which, however, does not mean the two queries are definitely unrelated. We think that the ranking of similar queries for query recommendation should take into account the similarity propagation in the sense that similarity scores may propagate from query to query, exhibiting implicit relatedness between un-adjacent nodes in QAG (e.g., q_2 is related to q_3 via q_1 in our running example.). Therefore, based on the QAG, two queries are related if there are paths from one to another. Under this definition, we generalize the concept of relatedness.

5.3 Optimization of similarity computation and update of query affinity graph

The first and second steps are used in the system initialization process and in the recommendation process when an input query is not in our query collection. Given that the query-URL bipartite graph is stored in our system, should we have to compute similarities for all pairs of queries that will count a quadratic number of values in the initialization process? Moreover, additional cost is incurred in the

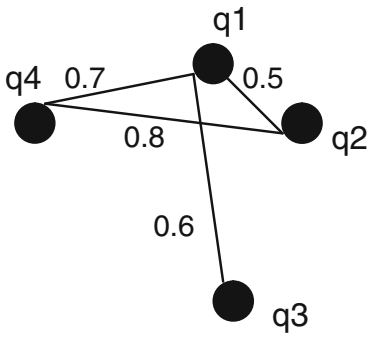


Fig. 3 An example of query affinity graph

recommendation process when the input query is new to our system, since this new query and its search results have to update the original query-URL bipartite graph and the query affinity graph. As we discussed in Section 4, the cost of updating the query-URL bipartite graph is acceptable. Should we compute similarity scores for all the stored queries with a new query to update the query affinity graph?

We introduce an optimization technique to reduce the cost of updating the query affinity graph, especially similarity computation. Recall that we are interested in pairs of queries whose similarity is above a specified threshold, a high quality collection. The recent work Bayardo et al. (2007) addressed this scalability issue without relying on approximation methods or extensive parameter tuning, and described different monotone minimum constraints for different similarity measures before computing the similarity scores. Here we use Jaccard measure as an example to explain their basic idea. For simplicity, let δ' =1-δ. We set up the query affinity graph using pairs of queries whose similarity is above δ'. The following inequalities are established:

$$\begin{aligned} Jaccard &= \frac{|P(q_i) \cap P(q_j)|}{|(P(q_i)| + |P(q_j)| - |P(q_i) \cap P(q_j)|} \\ &\leq \frac{|P(q_i)|}{|(P(q_i)| + |P(q_j)| - |P(q_i) \cap P(q_j)|} \\ &\leq \frac{|P(q_i)|}{|(P(q_i)| + |P(q_j)| - |(P(q_i)|} \\ &\leq \frac{|P(q_i)|}{|P(q_j)|} \leq \delta' . \end{aligned}$$

The above equation tells us if the $|P(q_i)|/|P(q_j)| \leq \delta'$, their similarity score does not need to be stored to reduce the amount of candidate pairs. Therefore, we can sort queries in the decreasing order of $|P(q)|$ to save on computation time. This preprocessing means that if $|P(q_i)|/|P(q_j)| \leq \delta'$ is met, the queries q_i to last query with the smallest number of search results can be skipped without doing similarity computation with q_j, thus accelerating our approach. Based on the similar idea, Bayardo et al. (2007) described how to get constraints for *Cosine*, *Dice*, and other popular measures in details (L_1 norm can be regarded as a kind of *Dice* measure). Therefore, we can sort our data set according to particular criteria such as vector size (e.g., $|P(q)|$) to improve the system computation performance.

6 Third phase: HAC-based query ranking and recommendation

The primary task of the third phase is to rank queries in the same group as an input query. We first discuss the design of our ranking methods and then describe how to adaptively output a recommendation list of related queries.

6.1 Our HAC-based ranking methods

Our ranking methods are HAC-based, and their respective clustering results (i.e., dendrograms) can estimate the relatedness between queries. Our idea is based on the hierarchical structure of the dendrogram. Hierarchy is more informative than

the unstructured set clusters in flat clustering algorithms like k-means and EM. As we know, HAC treats each query as a singleton group at the beginning, and then merges pairs of groups iteratively until all groups have been merged into a single group structured as a hierarchy that contains all queries. Furthermore, HAC has an interesting property that distance measures associated with successive merge operations can 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.

Urged by the monotonic property, 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 then such relevance is encoded as a dendrogram (a cluster tree) where edges are weighted by combination distance. We make use of this kind of tree structure to produce an ordered list of related queries for a specific query and the tree distance based ranking can better address the problem of similarity propagation.

The monotonic property is desirable in our context. However, not all HAC algorithms are monotonic. Lance and Williams (1966, 1967) proved that single-linkage and complete-linkage strategies are monotonic by definition. Moreover, they introduced a generalized recurrent formula including all special cases, defined as:

$$d_{hk} = \alpha_1 d_{hi} + \alpha_2 d_{hj} + \beta d_{ij} + \gamma \left| d_{hi} - d_{hj} \right| , \tag{6}$$

where the parameters α_1, α_2, β, and γ determine the nature of the strategy. (h), (i), and (j) are three groups, containing n_h, n_i, n_j elements respectively with inter-group distances already defined as d_{hi}, d_{hj}, d_{ij}. They further assume that the smallest of all distances still to be considered d_{ij}, so that (i) and (j) fuse to form a new group (k), with n_k $(= n_i + n_j)$ elements. The strategy is necessarily monotonic so long as $\alpha_1 + \alpha_2 + \beta \geq 1$ and $\gamma = 0$.

Previous work usually utilized HAC strategies to find clusters, while our approach designs a rank mechanism based on the clustering result (i.e., dendrogram) of HAC strategies. No much experience can be learned from previous work. Therefore, we adopt three popular monotonic HAC strategies for empirical study and propose another flexible HAC strategy for adaptive outputting recommendation lists. All the following HAC strategies can be derived from (6).

(1) Single-linkage strategy (SL):

The single-linkage strategy merges the two clusters with the smallest minimum pairwise distance, defined as:

$$\begin{aligned} d_{hk} &= MIN\left[d(q_h, q_k)\right] \\ &= \frac{1}{2} d_{hi} + \frac{1}{2} d_{hj} + 0 d_{ij} - \frac{1}{2} \left| d_{hi} - d_{hj} \right| , \end{aligned}$$

where although $\gamma <> 0$ does not satisfy the monotonicity constraint, the single-linkage strategy is monotonic by definition Lance and Williams (1966, 1967). However, in the single-linkage clustering, the similarity of two clusters is the similarity of their most similar members. This criterion is local Manning et al. (2008). We pay attention solely to the area where the two clusters come closest to each other. Other, more distant parts of the cluster and the clusters' overall structure are not taken into account. Therefore, it is easily affected by outliers and a chain of points can

be extended for long distances without regard to the overall shape of the emerging cluster. This effect is called chaining. Since its merge criterion is strictly local, other strategies are studied for comparison.

An alternative monotonic clustering is the complete-linkage strategy which is the opposite of the single linkage strategy, but unsuitable for our problem. For example, in Fig. 3 there are $d(q_1, q_2) = 0.5$, $d(q_1, q_3) = 0.6$, and $d(q_2, q_3) = 1$. In the complete-linkage strategy, q_2 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_2 since there is a path from q_2, to q_3 via q_1.

(2) Weighted-average strategy (WA):

In the weighted-average strategy, the distance between two clusters is the average distance between all possible pairs of nodes in the two clusters. Its definition is given as follow:

$$\begin{aligned} d_{hk} &= \frac{\sum_{q_h \in n_h} \sum_{q_k \in n_k} d(q_h, q_k)}{2} \\ &= \frac{\sum_{q_h \in n_h} \sum_{q_i \in n_i} d(q_h, q_i) + \sum_{q_h \in n_h} \sum_{q_j \in n_j} d(q_h, q_j)}{2} \\ &= \frac{1}{2} d_{hi} + \frac{1}{2} d_{hj} + 0 d_{ij} + 0 |d_{hi} - d_{hj}| \,, \end{aligned}$$

where

$$\alpha_i + \alpha_j + \beta = \frac{1}{2} + \frac{1}{2} + 0 = 1 \;\; and \;\; \gamma = 0 \,,$$

satisfies the constraint of monotonicity. Since the distance between clusters is calculated as a simple arithmetic average, the weighted-average strategy is computationally easy. When there are unequal numbers of elements in the clusters, the original distances of pairs of queries do not contribute equally to the intermediate calculations, and the final result is therefore said to be weighted. An obvious weakness is that the number of elements in each cluster are not taken into account.

(3) Group-average strategy (GA):

Similar to the weighted-average strategy, the group-average clustering also considers the distance between two clusters, defined as ($n_k = n_i + n_j$):

$$\begin{aligned} d_{hk} &= \frac{\sum_{q_h \in n_h} \sum_{q_k \in n_k} d(q_h, q_k)}{n_h n_k} \\ &= \frac{\sum_{q_h \in n_h} \sum_{q_i \in n_i} d(q_h, q_i) + \sum_{q_h \in n_h} \sum_{q_j \in n_j} d(q_h, q_j)}{n_h (n_i + n_j)} \\ &= \frac{n_h n_i}{n_h (n_i + n_j)} \times \left(\frac{\sum_{q_h \in n_h} \sum_{q_i \in n_i} d(q_h, q_i)}{n_h n_i} \right) \\ &\quad + \frac{n_h n_j}{n_h (n_i + n_j)} \left(\frac{\sum_{q_h \in n_h} \sum_{q_j \in n_j} d(q_h, q_j)}{n_h n_j} \right) \end{aligned}$$

$$= \frac{n_i}{n_i + n_j} d_{hi} + \frac{n_j}{n_i + n_j} d_{hj}$$
$$= \frac{n_i}{n_i + n_j} d_{hi} + \frac{n_j}{n_i + n_j} d_{hj} + 0 d_{ij} + 0 |d_{hi} - d_{hj}| ,$$

where

$$\alpha_i + \alpha_j + \beta = \frac{n_i}{n_i + n_j} + \frac{n_j}{n_i + n_j} + 0 = 1 \ \ and \ \ \gamma = 0 ,$$

satisfies the constraint of monotonicity, thus holding the monotonic property.

The group-average strategy computes the average of distances between all pairs of objects, where each pair is made up of one object from each group, but also consider the size of each cluster. The calculation is slightly more complicated. As a result, each distance contributes equally to the final result, which is therefore said to be unweighted. Its dendrogram is a tree with intense clustering. Furthermore, both weighted- and group- average strategies avoid the pitfalls of the single-link and complete-link criteria, which equate cluster similarity with the similarity of a single pair of queries. One weakness of the group-average strategy is that pairs from the same cluster are not included in the average, i.e., $d_{ij}$when i and j are merged into a cluster k.

(4) Flexible strategy (α-F):

We can also consider a HAC strategy including self-similarities. The flexible strategy is defined as: $d_{hk} = \alpha d_{hi} + \alpha d_{hj} + (1 - 2\alpha) d_{ij}$, which was proposed by Lance and Williams (1966, 1967) as a flexible and monotonic strategy. When α increases from 0 to 1, its hierarchy changes from an almost completely *chained* system into one with increasingly *intense* clustering. A given set of queries may now, by varying the parameters, be made to appear as sharp as a dendrogram a user may desire. Simply speaking, when α is small, e.g., near to zero, the dendrogram of the α-F strategy is similar to the single-linkage strategy while when α is large, e.g., near to one, the dendrogram of α-F strategy is similar to the group-average strategy. If $\alpha = 0.5$, α-F strategy becomes the weighted-average strategy. In experiments, we will show its flexibility to change the dendrograms of clustering, thus producing adaptive recommendation lists for users.

Parameters of the above four strategies in (6) are listed in Table 3. We will give empirical evidence as to how different HAC strategies affect the quality of recommendation and make use of the flexibility of the α-F strategy to allow users to interactively participate in the query recommendation process by changing the value of α.

Table 3 Parameters of different HAC strategies

Strategies	α_1	α_2	β	γ
SL	1/2	1/2	0	−1/2
WA	1/2	1/2	0	0
GA	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	0	0
α-F	α	α	$1 - 2\alpha$	0

6.2 The design of our ranking and recommendation procedure

We provide a sketch of our QUBIC HAC-based ranking procedure in the format of pseudo code in Table 4. After a HAC strategy is selected, our algorithm will utilize its result to rank queries in the same group as an input query. 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 (Steps 7∼9) before forming the recommendation list (Step 10).

First, we create an n-tuple $P = (P_1, \cdots, P_n)$ index to compute for each cluster i a nearest neighboring clustering P_i and each P_i for $1 \leq i \leq n$ satisfies the condition that $d(i, P_i) = \min\{d(i, j) : 1 \leq i, j \leq n, i \neq j\}$. Second, we replace the two clusters i and j by an agglomerated cluster k based on the chosen HAC method (see Step 9.4), and update the matrix D by removing clusters i and j and add the new cluster k. This process continues until no more merge is needed. Although the main weakness of agglomerative clustering methods is that they do not scale well with time complexity of at least $O(n^2)$, the complexity of the clustering and ranking process depends on the number of candidate queries obtained in Step 8 which is smaller than the

Table 4 Phase 3—HAC-based query ranking and recommendation

Input: an HAC strategy chosen by a user and an input query *iq*		
Output: a ranked list of related queries to the input query *iq*		
7.	If find the the connected component containing the query *iq*; Else update the graph processing 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.	
9.1	Initialize n-tuple P to identify a nearest neighbor of each cluster.	$O(n^2)$
9.2	For m=n downto 2 do	
	Begin Loop	
9.3	Search D with P to identify a closest pair(i,j) of clusters;	$O(m)$
9.4	Replace clusters i and j by an agglomerated cluster k; $d_{hk} = \alpha_1 D[h][i] + \alpha_2 D[h][j] + \beta D[i][j] + \gamma \lvert D[h][i] - D[h][j] \rvert$	$O(1)$
9.5	Update D to reflect deletion of i and j and compute distances between k and all remaining clusters(h);	$O(m)$
9.6	Update P to reflect deletion of i and j and inclusion of k;	$O(m^2)$
	End Loop	
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	$R[cq] = \lvert M[iq] - M[(iq, cq)] \rvert + \lvert M[cq] - M[(iq, cq)] \rvert$;	O(1)
	End Loop	
10.6	Quicksort $R[n]$ BY ASCENT;	$O(n \log n)$
10.7	Return the top similar queries but delete the candidate queries whose $D[iq][cq]$ are smaller than 0.2;	$O(n)$
13.	If the user is unsatisfied with the list, selects another HAC strategy; Go to 9.1; Else end this searching process.	

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 and increase 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. If a number j in the row is negative, then the single page j is merged at this stage. If j is positive, the merger is with the cluster formed at stage j of the algorithm. We show that the successive merging operations of the HAC algorithm naturally form an ordered list.

A HAC strategy builds a hierarchical structure (dendrogram) where an input query is merged with a group(query) at a distance ($M[iq]$) in the first time (Step 10.1 in Table 4), and the first merging for a candidate 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). Therefore, each of the three steps Step 10.1, 10.4, and 10.5 requires $O(n)$ searching the matrix. 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 use a concrete example to explain how HAC-based ranking works. A clustering structure produced by our experiments is illustrated in Fig. 4 where "Height", the label of the vertical axis, means the combination distance at each merging operation. As shown in Fig. 4, the dendrogram shows the relationships of queries arranged like the branches of a tree. Computing the distance (similarity) of two queries in the dendrogram can utilize the tree-distance based ranking mechanism which usually counts the number of edges between two nodes in a tree. In our HAC dendrogram, the edges are weighted by the combination distances of clustering, e.g., A, B, and C edges in Fig. 4. Therefore, our HAC-based ranking computes the sum of weights of edges between two queries (nodes). For example, given query 6 as the input

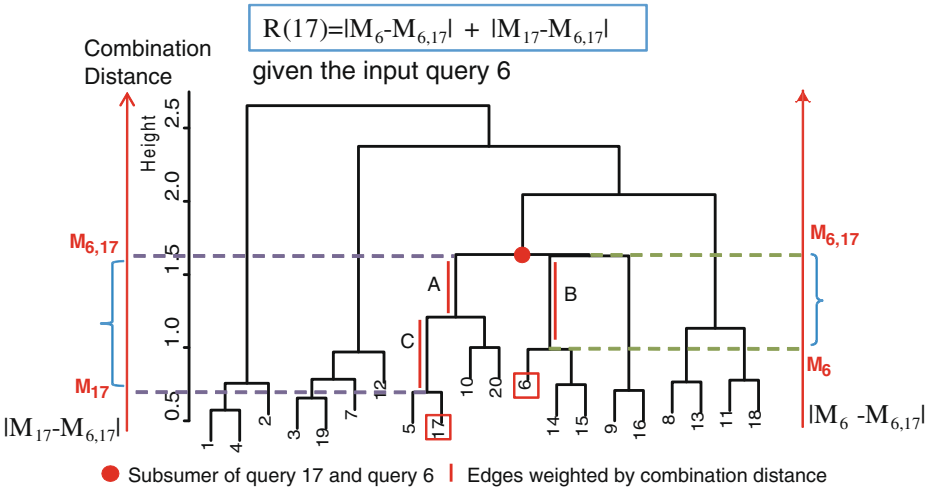


Fig. 4 HAC-based ranking using tree distance

query, the subsumber between query 6 and query 17 is marked as a dot in Fig. 4 and this subsumber is produced at the combination distance $M_{6,17}$. The sum of weights of edges between query 17 and the subsumber is $|M_{17} - M_{6,17}| = |0.7 - 1.6| = 0.9$ (leftmost) and the sum of weights of edges between query 6 and the subsumber is $|M_6 - M_{6,17}| = |1.0 - 1.6| = 0.6$ (rightmost). Then, the distance between query 6 and query 17 is $|M_{17} - M_{6,17}| + |M_6 - M_{6,17}| = 0.9 + 0.6 = 1.5$, as shown in the top of Fig. 4. We empirically show this HAC-based ranking is effective in Section 7.

A naïve ranking is simply based on the Jaccard, L_1 or Cosine similarity measures. Compared with our ranking methods, the naïve ranking only considers the similarity between two nodes without similarity propagation. Give a simple example shown in Fig. 5 which is the dendrograms of α-F strategy ($\alpha = 0.02$ and 0.5) applied on Fig. 3. The similarity scores of pairs of queries are $sim(q1, q2) = 1 - 0.5 = 0.5$, $sim(b, c) = 0$, $sim(a, c) = 1 - 0.6 = 0.4$, $sim(d, a) = 1 - 0.7 = 0.3$, and $sim(d, b) = 1 - 0.8 = 0.2$, as illustrated in Fig. 3. In Fig. 5a the query $q1$ and the query $q2$ are merged in the first combination under $\alpha = 0.02$. The query $q3$ will be merged with $q1(q2)$ in the second combination, and then the query $q4$ will be merged in the third combination. The ordering of related pages of the query $q2$ is $q1$, $q3$, and $q4$, despite of $sim(q2, q3) = 0$ and $sim(q2, q4) = 0.2$. Therefore, the tree distance based ranking naturally completes the similarity propagation from $q3$ to $q2$ via $q1$. Moreover, if the user chooses $\alpha = 0.5$ in Fig. 5b, the ordering list becomes $q1$, $q4$, and $q3$. Our HAC-base ranking is able to not only capture the similarity propagation between queries, but also adaptively output the ordering list of related queries. Users will be supplied with more candidates of related queries. More experiments results are reported in Section 7.

6.3 Discussions

As we discussed earlier, our similarity measure between queries is based on the common URLs returned from submitting both queries to a search engine (e.g., the number of common URLs in (1) and the weighted common URLs in (3) and (5)). 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,

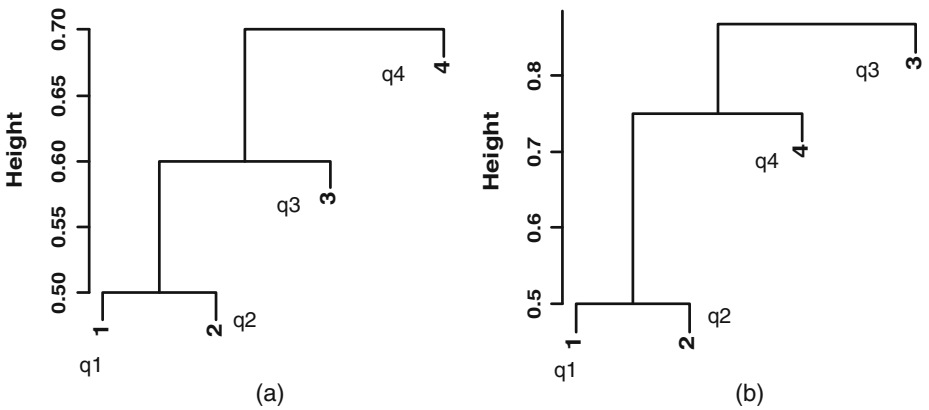


Fig. 5 **a** $\alpha = 0.02$ and **b** $\alpha = 0.5$ in (6)

and thus cannot improve the level of satisfaction of the users. Naturally, we want to suggest queries that achieve comparable search results. One way to approach this goal is to remove the candidates whose distance scores are smaller than a threshold.(e.g., in (1) in step 10.7, the value of 0.2 means that the overlap of common URLs exceeds 80 %). This ensures that our approach does not try to recommend *more similar* queries, but *less-similar* ones. From this viewpoint, query recommendation is different from query clustering. The latter is to mine queries which are more similar, but the former is to find less 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 5 of the second phase is because these candidates can work as bridge to propagate similarity between queries during the HAC clustering.

7 Experiments

In order to test the effectiveness of the QUBIC system for query recommendation, several experiments are conducted. Firstly, we test the precision quality of query recommendation to evaluate the number of related queries in a recommendation list. Then, we introduce Entropy as an evaluation measure to reflect the quality of individual lists in terms of homogeneity of related queries in a recommendation list. In addition, Kappa statistics and T-test for statistical testing are done for validating our manual evaluation. Finally, we also provide a case study to illustrate how our system can adaptively output recommendation lists by utilizing the flexibility of the HAC strategy.

7.1 Data sets

In the experiments, we used two data sets: QueryKDD and QueryTREC for evaluation. The former is the 800 queries of KDD cup 2005 data set[2] that are labelled by three people. The latter is the "10k" stream of the Efficiency task topics of the 2006 TREC Terabyte Track[3] (this task contains 6 topic streams: "all": full 100k topics, "stream-1" through "stream-4": four concurrent streams, and "10k": a small stream for the comparative efficiency task). Given each query, the top 50 URLs of search results were offered by Google API.[4] We preprocessed the URLs and queries by mapping their characters to lowercase and by chopping each retrieved URL into its hostname.

Evaluation is not an easy job because there is not an objective test data set for our problem at the current stage. The common solution is that we have to collect users judgments manually. However, we should evaluate the effectiveness of three similarity measures for each query, let alone parameter study in HAC algorithm

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

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

[4] http://code.google.com/apis/soapsearch/

Table 5 Selection of the number of top search results on the QueryKDD data set

#	Top 10	Top 20	Top 30	Top 40	Top 50
0	272,301 (85 %)	241,310 (76 %)	207,593 (65 %)	177,225 (55 %)	147,111 (46 %)
1	34,936 (11 %)	58,553 (18 %)	79,903 (25 %)	94,533 (30 %)	105,327 (33 %)
≥2	1,823 (0.6 %)	9,196 (2.9 %)	21,564 (6.7 %)	37,301 (12 %)	56,622 (18 %)

selection and graph generation. Therefore, we conduct experiments on the above two data sets. QueryKDD is for automatic evaluation which shows preliminary results in parameter study. Then QueryTREC is for manual evaluation which further verify the effectiveness of our proposed rank strategies.

7.2 Statistics about data sets

We did a preliminary analysis of the number of top URLs used in our scheme on the QueryKDD data set, as shown in Table 5. The first column (#) represents the number of common URLs between queries, and the first row (top 10∼top 50) represents the number of pairs of queries whose total number reaches $(X * X - X)/2$ (here X is set to 800). From Table 5, we observed that about 85 % of the pairs of queries had no common URLs in the top 10 of the search results, while in the top 50 search results, the number of pairs of queries with no common URLs was reduced to 46 %. The distance (similarity) measures discussed in Section 5 rely on the common URLs shared by two queries. If the number of returned search results is too small to get the URL overlap between two queries, we have to increase the number of returned results. Therefore, in the following experiments we stored the top 50 of search results for each query.

Tables 6 and 7 summarize the statistics of the query-URL bipartite graphs and the affinity graphs of queries generated from our two data sets. Note that Table 6, "Hostnames" means such URLs that the original URLs of search results are chopped into their hostnames. Therefore, it is possible that there are duplicate hostnames in the top 50 search results. "URLs" represents the original URLs of search results without any additional processing.

In Table 7, the row "ALL" represents that the affinity graphs of queries are set up with the threshold $\delta = 1$ (Step 5 in Table 2). For each query, we sorted the other queries in terms of distance scores. We found that the similarity scores of the bottom queries (from 100th or 200th) are larger than 0.98 in QueryKDD or 0.85 in QueryTREC. Moreover, our task is not to find all the similar queries like query clustering, but to recommend the more relevant ones. Therefore, the values of the threshold dissimilarity δ for QueryKDD and QueryTREC are 0.98 and 0.85 respectively.

One observation from Table 7 is that the number of queries in the affinity graph is smaller than that in Query-URL bipartite graph (695 vs. 800 and 9559 vs. 10,000)

Table 6 Query-URL bipartite graph using top 50 search results

Data set	Queries	Hostnames & edges	URLs & edges
QueryKDD	800	26,206 with 39,599	39,509 with 39,624
QueryTREC	10,000	147,761 with 491,956	491,954 with 502,749

Table 7 Affinity graph of queries using top 50 search results

	Queries	Edges	Average number of neighbors per query
QueryKDD data set			
ALL	695	18,762	27
Jaccard	678	8,841	13
Cosine	683	9,511	14
L1 norm	684	10,149	15
QueryTREC data set			
ALL	9559	1,751,148	183
Jaccard	1297	6,738	5
Cosine	2355	7,600	3
L1 norm	1275	3,156	2.5

because some queries are isolated. "By isolated" means that a query has no common URLs with any other queries. The column "Neighbors/Query" gives the single hop neighbors per query. To include more candidates during recommendation, users can set $H \geq 2$ hops from any input query (Step 8 in Table 4). With increasing the value of H, the HAC clustering will become slowly. We computed the number of candidates of each query according to different values of H. For QueryKDD dataset, when $H = 4$, the number of queries whose total number of candidates is larger than 700 is 702 (the total number of queries is 800); when $H = 2$, there is only one query whose number of candidates reaches 571. For QueryTREC dataset, if $H = 4$, most of queries have the number of candidates larger than 8,000; while $H = 2$, most of them have the number of candidates smaller than 2,000. Therefore, for evaluation, we choose $H = 3$ to keep the overall data within a reasonable size.

We observed that the produced affinity graph is an unconnected graph which may be subdivided into connected components. This fact encouraged us to extract such connected components in advance (Step 6 in Table 2), thus reducing the computation complexity in the HAC-based ranking phase.

7.3 Evaluation measures

7.3.1 Precision

The precision (P) at the top N queries of a recommendation list is defined as:

$$P@N = \frac{Related\ Queries}{N}. \tag{7}$$

The measure $P@N$ means how many valuable answers our algorithm gives at the top of recommendation lists. We set $N = 10$ in the following evaluations. Query recommendation is a ranking problem. We can evaluate our system by other evaluation measures like MAP and $NDCG$ (Manning et al. 2008) which are also used for ranking problems, especially in Web search. Because the length of a term-based query is much shorter than that of a Web page, Web users can read the top ten or twenty recommended queries much more quickly than they browse and click the top Web pages one by one. In the evaluation of query recommendation, we are more interested in the number of related queries in a recommendation list, i.e., *Precision* than the order in which the queries are returned such as MAP, $NDCG$ and so on.

7.3.2 Entropy

In the queryKDD data set the manual categorization information of queries are available. Each query is followed by its top five categories labeled by three human experts. There may be fewer than five categories for some of the queries and these categories come from 67 predefined categories. The category information assumes that it is possible to perform a direct comparison of the recommendation output. Our hypothesis is that an optimal recommendation output consists of queries with the same class labels. We describe a quantitative evaluation measure based on this criteria. An information entropy approach can evaluate the quality of a set of clusters according to the original class labels of the data (Boley 1998). By replacing *cluster* with *ranking list*, we compute the entropy of each ranking list of query recommendation, defined as

$$Er_i = -\sum_j \frac{n(l_j, r_i)}{n(r_i)} log \frac{n(l_j, r_i)}{n(r_i)}, \tag{8}$$

where $n(l_j, r_i)$ is the number of the related queries in the ranking list r_i with a predefined label l_j and $n(r_i) = \sum_j n(l_j, r_i)$ is the number of the related queries in the ranking list r_i. The overall *ranking list entropy* Er is then given by a weighted sum of individual cluster entropies by

$$Er = \frac{1}{\sum_i n(r_i)} \sum_i n(r_i) Er_i . \tag{9}$$

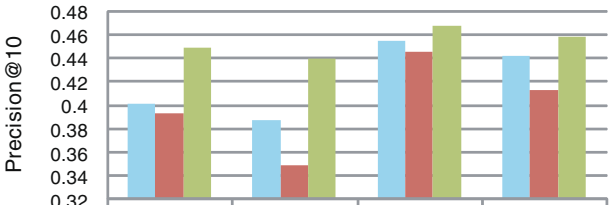
The entropy of ranking lists reflects the quality of individual lists in terms of homogeneity of related queries in a recommendation (a smaller value indicates a higher homogeneity).

7.4 Results and discussions

As introduced in Section 1, usually the relatedness between the two queries is directly estimated according to the overlap of the two lists of search results (URLs) returned. This kind of methods is called naïve ranking where various measures based on the overlap are used to compute the relatedness values. In this paper, we show the results of three such measures, i.e., (1)–(3). The naïve ranking is our baseline which can represent the main idea of the existing techniques in this direction.

7.4.1 Experiments on the QueryKDD data set

Precision For precision evaluation, if two queries share at least a same category, we regard them as related queries. The experimental results averaged by three labellers are shown in Fig. 6. The single-linkage strategy performs worst among all the HAC-based rank mechanisms. It is even worse than the naïve ranking and the performance is lowered by 3.9 %, 11.3 %, and 1.9 % for the three similarity measures. The reason is that the single-linkage strategy is sensitive to outliers since it merges in each step the two clusters whose two closest members have the smallest distance or the two clusters with the smallest minimum pairwise distance. While the group-average and weighted-average strategies got higher precision than that of the naive one under all the three measures. The most improvement is about 13.3 % when using the



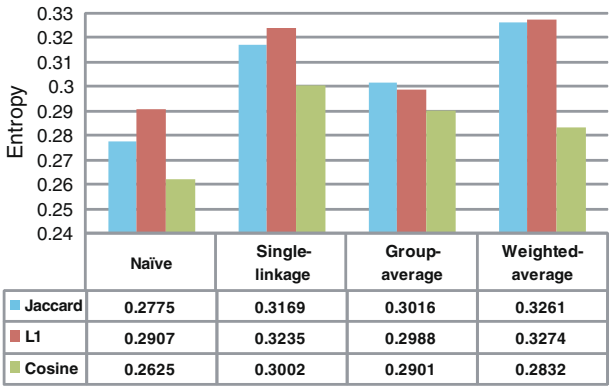
	Naïve	Single-linkage	Group-average	Weighted-average
Jaccard	0.4015	0.3868	0.4549	0.4416
L1	0.3932	0.3489	0.4449	0.413
Cosine	0.4482	0.4397	0.4675	0.4575

Fig. 6 Precision@10 of QueryKDD data set

Jaccard measure and the group-average strategy. The two strategies are robust to outliers because they consider the distance between one cluster and another cluster to be equal to the average distance (or (6)) from any member of one cluster to any member of the other cluster. Furthermore, the group-average strategy produces higher precision scores than the weighted-average strategy because the former also considers the size of clusters in its cluster processing.

Among the three similarity measures, The *Cosine* measure is the best one and the L_1 norm is worst, while the *Jaccard* measure shows better than the L_1 norm. The *Jaccard* measure simply computes the number of common URLs between queries, thus assigning equal weights to all the URLs in a query. While the L_1 measure weight a URL by its frequency in the search result of a query. As we know, popular sites like Google and Yahoo! may be returned within the search results of a query, which will be noisy irrespective of the content of the query. Since the L_1 norm may assign more weights on these URLs, this weighting function hurts the precision in our context. The other weakness of the *Jaccard* measure is that it does not distinguish between two queries which have the same one URL and two queries which have the same two URLs. The URL-vector based $tf * idf$ weighing method is more effective than both of the *Jaccard* measure and the L_1 norm since it assigns less weights on URLs with higher query frequency that might be less informative to represent a distinct query, and more weights on URLs with higher occurrences in a query. On average, the group-average strategy with the *Cosine* measure is the winning combination in terms of precision@10.

Entropy Entropy evaluation defined in (9) is illustrated in Fig. 7. With regard to the overall performance, Fig. 7 shows that the group-average strategy with the *Cosine* measure produced lowest entropy values among the three HAC-based rank mechanism (smaller is better for entropy). But the HAC-based ranking methods show higher entropy that the naïve ranking. Because the naive rank mechanism directly uses the similarity measures, it has lower possibility to include noisy queries. However, the HAC strategies try to get more relevant queries by similarity propagation at the same time they has higher risk to meet noisy queries. Therefore, the final lists of the naïve one are purer than those of HAC strategies. According to precision, entropy and robustness, the group-average strategy with the *Cosine* measure is the



	Naïve	Single-linkage	Group-average	Weighted-average
Jaccard	0.2775	0.3169	0.3016	0.3261
L1	0.2907	0.3235	0.2988	0.3274
Cosine	0.2625	0.3002	0.2901	0.2832

Fig. 7 Entropy of ranking lists of QueryKDD data set

optimal combination. These results prove that our QUBIC approach is effective in query recommendation.

7.4.2 Experiments on the QueryTREC data set

The group-average strategy generally shows the best performance among all HAC strategies according to the results of QueryKDD data set. Therefore, in the QueryTREC, we compare the group-average strategy with the naïve ranking. Unlike the QueryKDD data set, the QueryTREC data set does not supply us with the categorization information. Thus, it is not feasible to use the entropy of rank lists as our evaluation metric. Three human evaluators are invited to judge the performance of our algorithm. After running our three-phase approach with 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 algorithms work.

Kappa statistics After collecting users' judgment results, we study the quality of users. We want to know the variability of user's ratings to measure user disagreement which tell us how users classify individual subjects (queries) into the same category (relevant or irrelevant) on the measurement scale. The judgments from different users should largely reach a good agreement for a same test query.

Kappa statistics is one of the most common approaches (Siegel and Castellan 1988). Kappa can be thought of as the chance-corrected proportional agreement, and possible values range from +1 (perfect agreement) via 0 (no agreement above that expected by chance) to −1 (complete disagreement). Table 8 provides a rough guide of what is a good agreement. We require the three evaluators to answer

Table 8 Kappa and strength of agreement

Kappa	Strength	Kappa	Strength
0.00	Poor	0.41–0.60	Moderate
0.01–0.20	Slight	0.61–0.80	Substantial
0.21–0.40	Fair	0.81–1.00	Almost perfect

Table 9 Paired t-test results for statistically significant testing

Significance	Jaccard	L_1 norm	Cosine
Group-average vs. naïve	YES	YES	YES

questionnaires that supply two recommended queries per test query.[5] Because two users are grouped as a pair to compute a Kappa value, the total number of test pairs is 3 ($C_3^2 = 3$). We collected the relevance judgment results of the nine test queries and the average of all Kappa values is 0.421. This value is in the range [0.41,0.6], a moderate agreement in general.

To sum up, in terms of the statements in Table 3, the agreement in our results is moderate at 95 % confidence level. The number of the users is not very large, but our Kappa statistics analysis shows that the quality of the users is satisfactory. Thus, their judgments are reliable for evaluation.

T-test for statistical testing We will compare the performance of the naïve ranking and the group-average strategy using each similarity measure. Since the number of test queries is 30, a paired t-test for statistical testing is performed to verify whether the improvements of the group-average strategy over the naïve ranking are *statistically significant* or not.

Table 9 lists the results of t-test statistical analysis[6] with a confidence level of 95 %. In Table 9 *YES* means the mean difference of the percentage values of two feature spaces is significant at the 95 % level. Since experimental results also tell us that the group-average strategy produces higher precision scores than the naïve ranking, we can say that the improvements of the group-average strategy over the naïve ranking is statistically significant when the experiments are conducted on 30 test queries. In the following, we will give how much precision scores they produce.

Precision *Related Query* in (7) is the number of manually tagged related queries. Their averaged results are reported in Table 10. It is clear that the evaluation approach is somewhat subjective. However, the concept of similarity is also subjective due to the various information needs of users. It is not easy to construct an objective test data set for our problem at the current stage.

The evaluation results using precision measure are reported in Table 10. From the experiment results of the QueryKDD data set, the *Cosine* measure achieves best performances among all the three similarity measures because the *Jaccard* measure overlooks the frequency information of URLs and the L_1 measure cannot avoid the side-effect of popular sites on the quality of query recommendation. In addition, the group-average strategy still shows higher precision scores than the naïve ranking. These results are consistent with the results of QueryKDD data set. We can conclude that the *Cosine* measure using the conventional TFIDF 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.

[5]Thirty search queries * 2 recommended queries = 60 evaluated queries

[6]http://www.r-project.org/

Table 10 *P*@10 of QueryTREC data set

	Naïve	Group-average
Jaccard	0.43	0.56
L_1 norm	0.32	0.50
Cosine	0.64	0.73

Case study In our case study, we discuss the top ten related queries returned by the *Cosine* measure for the query "state of wa department of social and health service division of child support" (topic number = 8945 in the "10k" stream of the Efficiency task topics of the 2006 TREC Terabyte Track). The affinity graph of this query is shown in Fig. 8. A traditional method of finding related queries is to select queries with higher similarity scores computed by a measure, such as *Jaccard*, *Cosine*, and so forth. Such queries are only adjacent vertices of a targeted query (e.g., vertices 811, 6684, and 8036 in Fig. 8). As we have discussed, however, similarity propagates from query to query. From the graph theoretical viewpoint, it means that related queries refer to queries from which there are paths to the targeted query (e.g., vertices 1613, 4061, and 6604 in Fig. 8).

Our HAC strategy can identify unadjacent related queries through propagating similarity over other related ones, as listed in Table 11. Moreover, in Fig. 9, as α increases from 0.05 to 0.85 in the α-F strategy, the hierarchy changes from an almost completely "chained" system to one with increasingly intense clustering. Using query 1 as the input query and query 5 in Fig. 9d as a concrete example, we know $M[1] = 0.575$, $M[5] = 0.7$, and $M[(1, 5)] = 0.8$. Then, the distance between query 1 and query 5 ($R[5]$) is 0.375 which may vary with hierarchical structures produced by different values of α in the α-F strategies. We have listed the recommendation results for a query in Table 11 using different HAC strategies.

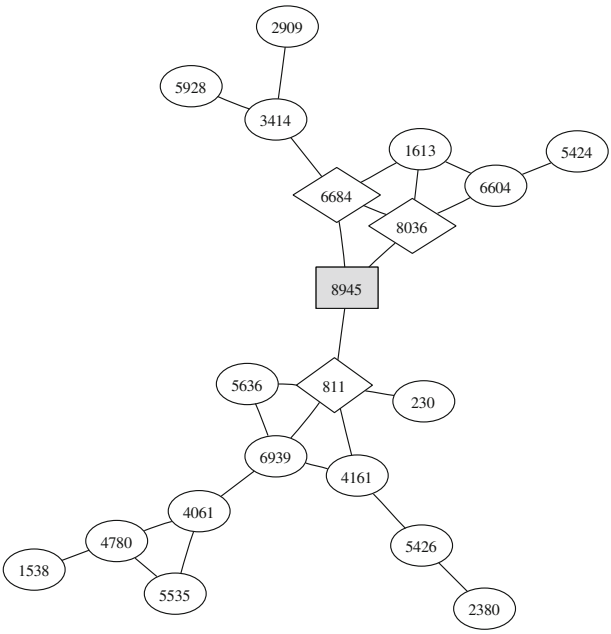


Fig. 8 Affinity graph of the query with the topic number = 8945

Table 11 Recommendation lists of the query "8945"

Rank	$\alpha = 0.05$	$\alpha = 0.25$	$\alpha = 0.45$	$\alpha = 0.85$
0	8945 : 1	8945 : 1	8945 : 1	8945 : 1
1	**6684 : 4**	**6684 : 4**	**6684 : 4**	**6684 : 4**
2	**8036 : 2**	**8036 : 2**	**8036 : 2**	**8036 : 2**
3	1613 : 17	1613 : 17	5424 : 10	5424 : 10
4	6604 : 5	6604 : 5	6604 : 5	230 : 20
5	5424 : 10	4161 : 12	1613 : 17	5928 : 6
6	**811 : 19**	5636 : 7	230 : 20	4161 : 12
7	6939 : 3	5426 : 9	5928 : 6	5636 : 7
8	5636 : 7	2380 : 16	4161 : 12	4780 : 11
9	4161 : 12	6939 : 3	4780 : 11	1538 : 18
10	5426 : 9	**811 : 19**	1538 : 18	**811 : 19**

Bold values show that even the same queries are suggested by adjusting alpha, but their positions in suggestion lists may different

It is obvious that the order in the list of related pages partly changes as well. We represented this change in Table 11 where "8945 : 1" means that "8945" is the topic number in QueryTREC data set and "1" is the query number in Fig. 9. In our adaptive scheme, if a user inputs a query to retrieve web pages, our system will supply her with a list of related queries from our data set. Then, the user can formulate his/her query and do a new search, or she may also ask our system to produce a new list of related queries by changing the value of α, e.g., adjusting a sidebar. As we discussed in Section 6, when α is near to zero, the dendrogram of the α-F strategy is similar to the single-linkage strategy; when α is near to one, its dendrogram is similar to the group-average strategy. If $\alpha = 0.5$, α-F strategy becomes the weighted-average strategy. This can guide users how to adaptively get recommendation lists. The naïve

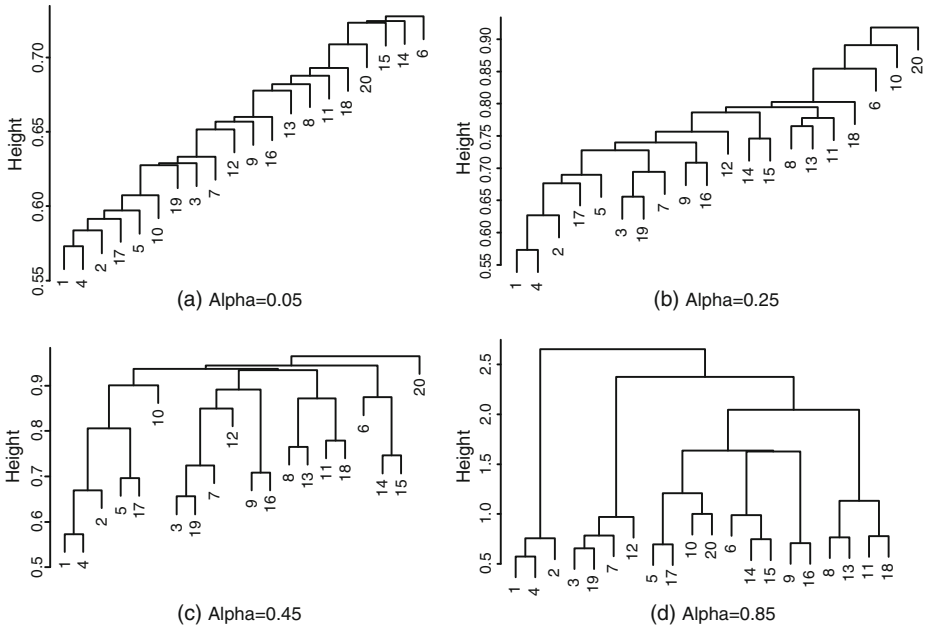


Fig. 9 Effect of varying HAC strategies. Measure: Cosine; Height: the combination distance at each merging

Table 12 MAP at tops 5 and 10

	Google	Borda	Copeland
Top 5	0.3423	0.3875	0.4021
Top 10	0.3659	0.4107	0.4268

ranking cannot supply users with such diverse lists of query recommendation due to its locality and lack of effective similarity propagation.

Improving web search by utilizing recommended related queries In Web search, with the aid of related query recommendation, search users can revise their initial queries in several serial rounds in pursuit of finding needed Web pages. Here, we aggregate search results of related queries to improve the retrieval quality. Given an initial query and the recommended related queries, their search result lists are concurrently processed from an existing search engine and then a single list is formed by aggregating all the retrieved lists.

First, for each of the 30 test search queries, we select the top three recommended queries given by our group-average ranking strategies. Second, the original 30 test queries with their three recommended queries are send to Google search engine and their top 40 search results are obtained. Third, two rank aggregation algorithms, i.e., Borda (1781) and Copeland (1951), are used to aggregate the search result lists of each original test queries and its top three recommended queries. Then, we manually judge whether the top 10 Web pages in each aggregated list are relevant or not to the original test query. Finally, Mean Average Precision (MAP) (Manning et al. 2008) is used for evaluation. Parameter selection and more discussions are in our paper (Li et al. 2011). The experimental results of the 30 test queries are shown in Table 12. *Google* is our baseline in which Google's MAP score averaged by all the related queries of each test query is calculated. It is shown in Table 12 that the two rank aggregation algorithms produce higher MAP scores than the baseline at tops 5 and 10. For example, the improvements of the two rank aggregation algorithms at top 5 are 13.2 % and 17.47 %, respectively. At top 10, the improvements are 12.24 % and 16.64 %. The experiment results validate the effectiveness of utilizing recommended related queries to achieve the search quality improvement.

8 Conclusions

We presented QBUIC, a query-URL bipartite graph based approach to query recommendation. Our query recommendation system can adaptively recommend related queries to a given query by analyzing the query-URL history, consisting of three phases, i.e, preparation, graph generating, and HAC-based ranking phases. In the first phase, a query-URL bipartite graph was build by retrieving search results of queries from a search engine. In the second phase we generated the query affinity graph using the query-URL vector based similarity measures, and extracted connected components from the query affinity graph. In the third phase, we employed a selected HAC strategy to output recommendation lists of related queries. Experimental results are very encouraging: many similar queries are grouped, which enables users to pick up different recommendation lists of related queries through the adjusting of the parameter α in the flexible strategy. Moreover, the group-average

strategy based ranking with the *Cosine* measure shows the highest precision in two data sets. These experimental results demonstrate the effectiveness of our approach. Last, but not the least, we evaluated the effect of related queries on the quality of Web information retrieval since the goal of query recommendation tries to help users get their desired information by formulating better search queries.

Our research continues along several dimensions, including more comprehensive user study in terms of usability and performance of our QUBIC system, considering the effects of user click log information on recommendation quality, and so on.

References

Aris, A., Luca, B., Carlos, C., Aristides, G. (2010). An optimization framework for query recommendation. In *Proc. of the Third international conference on web search and web data mining (WSDM'10)* (pp. 161–170). New York: ACM.

Baeza-Yates, R.A., Hurtado, C.A., Mendoza, M. (2007). Improving search engines by query clustering. *JASIST, 58*, 1793–1804.

Bayardo, R.J., Ma, Y., Srikant, R. (2007). Scaling up all pairs similarity search. In *Proc. of the 16th international conference on World Wide Web (WWW'07), Banff, Alberta, Canada* (pp. 131–140). New York: ACM.

Beeferman, D., & Berger, A.L. (2000). Agglomerative clustering of a search engine query log. In *Proc. of the 6th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'00), Boston, MA, USA* (pp. 407–416). New York: ACM.

Boley, D. (1998). Principal direction divisive partitioning. *Data Mining and Knowledge Discovery, 2*, 325–344.

Borda, J. (1781). Mémoire sur les élections au scrutin. *Comptes rendus de l'Académie des sciences, 44*, 42–51.

Buckley, C., Salton, G., Allan, J., Singhal, A. (1994). Automatic query expansion using SMART. In *Proc. of text retrieval conference (TREC'03)* (pp. 69–80). Gaithersburg: National Institute of Standards and Technology (NIST).

Calado, P., Cristo, M., Gonçalves, M.A., de Moura, E.S., Ribeiro-Neto, B.A., Ziviani, N. (2006). Link-based similarity measures for the classification of web documents. *JASIST, 57*, 208–221.

Chirita, P.-A., Firan, C.S., Nejdl, W. (2007). Personalized query expansion for the web. In *Proc. of the 30th annual international ACM SIGIR conference on research and development in information retrieval (SIGIR'07), Amsterdam, The Netherlands* (pp. 7–14). New York: ACM.

Collins-Thompson, K., & Callan, J. (2005). Query expansion using random walk models. In *Proc. of the 14th ACM CIKM international conference on information and knowledge management (CIKM'05), Bremen, Germany* (pp. 704–711). New York: ACM.

Copeland, A. (1951). A reasonable social welfare function. In *Seminar on Mathematics in Social Sciences*. University of Michigan.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (1990). *Introduction to Algorithms*. New York: McGraw-Hill.

Cui, H., Wen, J.-R., Nie, J.-Y., Ma, W.-Y. (2003). Query expansion by mining user logs. *IEEE Transactions on Knowledge and Data Engineering, 15*, 829–839.

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

Fitzpatrick, L., & Dent, M. (1997). Automatic feedback using past queries: social searching? In *Proc. of the 20th annual international acm sigir conference on research and development in information retrieval (SIGIR'97), Philadelphia, PA, USA* (pp. 306–313). New York: ACM.

Glance, N.S. (2001). Community search assistant. In *Proc. of the 9th international conference on intelligent user interfaces (IUI'01), Santa Fe, NM, USA* (pp. 91–96). New York: ACM.

Hansen, M., & Shriver, E. (2001). Using navigation data to improve ir functions in the context of web search. In *Proc. of the 10th ACM CIKM international conference on information and knowledge management (CIKM'01), Atlanta, Georgia, USA* (pp. 135–142). New York: ACM.

Jansen, B.J., Spink, A., Bateman, J., and Saracevic, T. (1998). Real life information retrieval: a study of user queries on the web. *SIGIR Forum, 32*, 5–17.

Jeh, G., & Widom, J. (2002). Simrank: a measure of structural-context similarity. In *Proc. of the 8th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'02), Edmonton, Alberta, Canada* (pp. 538–543). New York: ACM.

Kleinberg, J.M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM, 46*, 604–632.

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

Lance, G.N., & Williams, W.T. (1967). A general theory of classificatory sorting strategies: 1. Hierarchical systems. *Computer Journal, 9*, 373–380.

Li, L., Otsuga, S., Kitsuregawa, M. (2010). Finding related search engine queries by web community based query enrichment. *World Wide Web, 13*, 121–142.

Li, L., Xu, G., Zhang, Y., Kitsuregawa, M. (2011). Random walk based rank aggregation to improving web search. *Knowledge-Based Systems, 24*, 943–951.

Lin, J. (1991). Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory, 37*, 145–151.

Manning, C.D., Raghavan, P., Schutze, H. (2008). *Introduction to information retrieval*. Cambridge: Cambridge University Press.

Otsuka, S., & Kitsuregawa, M. (2006). Clustering of search engine keywords using access logs. In *Proc. of the 17th international conference on database and expert systems applications (DEXA'06), Kraków, Poland* (pp. 842–852). Berlin: Springer.

Page, L., Brin, S., Motwani, R., Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Technical report. Stanford Digital Library Technologies Project.

Pereira, F.C.N., Tishby, N., Lee, L. (1993). Distributional clustering of english words. In *Proc. of the 30th annual meeting of the association for computational linguistics (ACL'93), Columbus, Ohio, USA* (pp. 183–190). Menlo Park: Association for Computational Linguistics.

Qixia, J., & Maosong, S. (2011). Fast query recommendation by search. In *Proc. of the twenty-fifth AAAI conference on artificial intelligence (AAAI'11), San Francisco, California, USA* (pp. 1192–1197). Menlo Park: AAAI Press.

Raghavan, V.V., & Sever, H. (1995). On the reuse of past optimal queries. In *Proc. of the 18th annual international ACM SIGIR conference on research and development in information retrieval (SIGIR'95), Seattle, Washington, USA* (pp. 344–350). New York: ACM.

Rege, M., Dong, M., Fotouhi, F. (2006). Co-clustering documents and words using bipartite isoperimetric graph partitioning. In *Proc. of the 6th IEEE international conference on data mining (ICDM'06), Hong Kong, China* (pp. 532–541). Los Alamitos: IEEE Computer Society.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *JASIS, 41*, 288–297.

Siegel, S., & Castellan, N.J. (1988). *Nonparametric statistics for the behavioral sciences* (2nd ed.). New York: McGraw-Hill.

Sun, J., Qu, H., Chakrabarti, D., Faloutsos, C. (2005). Neighborhood formation and anomaly detection in bipartite graphs. In *Proc. of the 5th IEEE international conference on data mining (ICDM'05), Houston, Texas, USA* (pp. 418–425). Los Alamitos: IEEE Computer Society.

Sun, R., Ong, C.-H., Chua, T.-S. (2006). Mining dependency relations for query expansion in passage retrieval. In *Proc. of the 29th annual international ACM SIGIR conference on research and development in information retrieval (SIGIR'06), Seattle, Washington, USA* (pp. 382–389). New York: ACM.

Voorhees, E.M. (1994). Query expansion using lexical-semantic relations. In *Proc. of the 17th annual international ACM-SIGIR conference on research and development in information retrieval (SIGIR'94), Dublin, Ireland* (pp. 61–69). New York: ACM.

Vuong, N., & Tru, C. (2010). Ontology-Based query expansion with latently related named entities for semantic text search. *Advances in Intelligent Information and Database Systems, 283*, 41–52.

Wen, J.-R., Nie, J.-Y., Zhang, H. (2002). Query clustering using user logs. *ACM Transactions on Information Systems, 20*, 59–81.

Xiaohui, Y., Jiafeng, G., Xueqi, C. (2011). Context-aware query recommendation by learning high-order relation in query logs. In *Proc. of the 20th ACM conference on information and knowledge management (CIKM'11), Glasgow, United Kingdom* (pp. 2073–2076). New York: ACM.

Xu, J., & Croft, W.B. (1996). Query expansion using local and global document analysis. In *Proc. of the 19th annual international acm sigir conference on research and development in information retrieval (SIGIR'96), Zurich, Switzerland* (pp. 4–11). New York: ACM.

Xu, J., & Croft, W.B. (2000). Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems, 18*, 79–112.

Yunlong, M., Hongfei, L., Yuan, L. (2011). Selecting related terms in query-logs using two-stage SimRank. In *Proc. of the 20th ACM conference on information and knowledge management (CIKM'11), Glasgow, United Kingdom* (pp. 1969–1972). New York: ACM.

Zha, H., He, X., Ding, C.H.Q., Gu, M., Simon, H.D. (2001). Bipartite graph partitioning and data clustering. In *Proc. of the 10th ACM CIKM international conference on information and knowledge management (CIKM'01), Atlanta, Georgia, USA* (pp. 25–32). New York: ACM.

Zhu, Y., & Gruenwald, L. (2005). Query expansion using web access log files. In *Proc. of the 16th international conference on database and expert systems applications (DEXA'05), Copenhagen, Denmark* (pp. 686–695). Berlin: Springer.