Finding Latent Neighbors for Query Recommendation: a User-Controllable Scheme

Lin LI† and Masaru KITSUREGAWA††

† Department of Information and Communication Engineering, University of Tokyo, Japan
†† Institute of Industrial Science, University of Tokyo, Japan

Abstract Ambiguous queries and the large amount of unindexed information by search engines give rise to the problem of finding latent neighbors for query recommendation. In this paper, we give our definition for latent neighbors of a query, and then propose a novel two-phase algorithm to solve this problem, which takes the connectivity of the query-URL bipartite graph as input. In addition, due to the subjectivity of similarity measures, a user-controllable scheme is presented to bridge the gap between the determinacy of similarity values and the indeterminacy of users' information needs. The experiment results from two mixtures of data collections demonstrate the usefulness of our algorithm and the feasibility of our scheme.

Key words query recommendation, clustering, rank mechanism

1. Introduction

Today's Web search engines provide friendly user interfaces which allow users to specify queries simply as lists of terms. A main problem occurs for users because of this: properly specifying their information needs by using only a few terms in their queries. One reason is that the ambiguity of short queries will retrieve web pages which are not what users are searching for. On the other hand, users might fail to choose terms at the appropriate level of representation for their information needs, and worst, they are either unwilling or unable to invest in the query construction.

The utilization of query expansion techniques has been investigated to help users formulate better queries. However, these techniques just append terms to existing terms in the query. While these techniques help recall, they generally hurt precision at the top of the result lists. It is appealing to think that past queries may be a source of additional evidence to help future users. For some users who are not very familiar with a certain domain, we can suggest alternative queries that have been searched by previous similar users from which they may gradually refine their queries, and hence become expert users. Previous queries having common terms with the input query are naturally recommended as alternatives. However, it is possible that the queries can be identical or phrased differently with different terms but for the same information needs. As a concrete example, there are two queries, "irs" and "file taxes online". Although they have no terms in common, Similarity between the two queries can be induced from the the overlap of the two lists of search results (URLs) returned to them. We notice that this approach still has its limitations: 1) two related queries may output different URLs in the top search results; 2) the whole URLs for a query stored in a search engine are too huge to effectively handle online searches; 3) even though it is possible to quickly process the whole URLs, the Internet remains the fastest growing new medium of all time [11]. Therefore, we think of the problem: how to find such a latent relationship between two queries that may not share common terms and URLs, due to the limited, available information we have.

In addition, the fact which should not be neglected is the subjectivity of the similarity. Query-to-query similarity is not an absolutely invariant constant, especially for broad-topic queries. For instance, the query "apple" is related to both fruits and computers. Whether fruit-related or computer-related queries are recommended should consider the context in which users submit the query. Thus, if users are allowed to take part in the recommendation, the context of the query will be expressly indicated. In this paper, we propose a user-controllable scheme to solve these difficulties. Our algorithm uses only the connectivity of a query-URL bipartite graph as input, and ignores the contents. It has two major steps: extracting affinity subgraphs of queries from the bipartite graph and then hierarchically clustering queries. The monotonicity of the successive merge operations and flexibility in the Hierarchical Agglomerative Clustering (HAC) make it possible for users to control the searching process. The main contributions of this paper are

summarized as follows.

(1) We put forward a new problem of finding latent neighbors for query recommendation (Section 2).

(2) A novel two-phase algorithm is proposed to solve the problem, falling into graph theory and HAC techniques (Section 3).

(3) Under our scheme, experiment results prove that users can become an active partner from a passive acceptor in the process of query recommendation (Section 4).

Discussions of the related work and the conclusion are severally addressed in Section 5 and Section 6, respectively.

2. Problem Statement

Our goal is to find latent neighbors from query-URL data. In this section, we will describe the concept of latent neighbors.

2.1 Query-URL Data

The query-URL data inspire us to mining the latent neighbors of a query. In tradition Information Retrieval, the similarity between a pair of quires is apparently valued by the query contents and other information resources like document contents and user feedback. However, there are three reasons that move us to the deeper consideration. First two related queries may output different URLs. Second the whole URLs for a query stored in a search engine are too huge to handle effectively. The last reason is that no search engines have already indexed all the information in the Internet until now, as we described in Section introduction. We expect the algorithm addressed here will preliminarily solve the mining problem.

2.2 Definition

This new relationship described is discovered among queries which may have common terms and URLs, or even not, different from the concept of similar queries in the traditional Information Retrieval. To look into whether there exists this latent relationship, an example is illustrated in Figure 1(a). The number of common URLs between q_1 and q_2 is equal to that between q_2 and q_3 (i.e., 1), while there is no common URL between q_1 and q_3. We conclude that q_1 is similar to q_2, and that q_2 is similar to q_3. It is straightforward that q_1 is also similar to q_3, and vice versa. We think of some similarity between q_1 and q_3 which may be not the same value as that between q_1 and q_2. This observation inspires us to form the concept of latent neighbors. Now we give the definition in the context of graph theory:

Latent neighbors: Given a query graph like Figure 1(b), its vertices correspond to queries only. Its edges connect pairs of queries, and are weighted according to a distance measure. Latent neighbors of a query refer to queries from which there are paths to the targeted query. Under this definition, we generalize the concept of neighbors: the neighbors of a query include adjacent (e.g., q_1 and q_2 in Figure 1(b)) and un-adjacent (e.g., q_1 and q_3 in Figure 1(b)) vertices. Scores will be explained shortly. Both of the two kinds of vertices in the graph are latent as neighbors. In particular, we want to let users choose the nearer neighbors represented as an ordering from the latent neighbors under their demand.

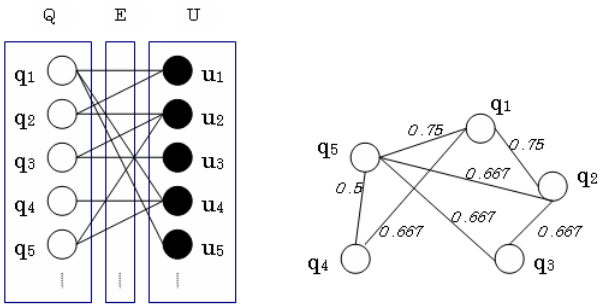

Figure 1 Query-URL Bipartite Graph (a) and Query Graph (b)

3. Proposed Algorithm

3.1 Preliminary

3.1.1 Bipartite Graph Model

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 independent sets, where Q (U) is the vertex set and E is the edge set of the graph. In this paper, we propose to recommend queries based on the inter-relationship of their corresponding URLs. The query-URL relationship can be intuitively represented as a bipartite graph and hence, it is used as our basic model where Q is a set of queries, U is a set of URLs, as shown in Figure 1(a). An edge e connects a query q and a URL u, if the URL u is returned by a search engine on a query q. Here we focus on the side of queries to recommend latent neighbors. 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.

3.1.2 HAC Strategies

HAC does not require a prespecified number of clusters because it 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 that contains all queries. Its output is a hierarchy, a structure that is more informative than the unstructured set of clusters in flat clustering. And most HAC algorithms find the optimal 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 should 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 $d_1 \leqq d_2 \leqq \cdots \leqq d_k$ must hold. A non-monotonic HAC strategy contains at least one inversion $d_i \geqq d_{i+1}$ and contradicts the fundamental assumption that the best possible merger is found at each step. Urged by the monotonic property, we think that queries which have the shortest distances will be merged first. At each remaining step in the hierarchy, the next closest pair of queries (or groups) should be merged. The sequence of merge operations scores the relevance of two queries and produces an ordering of neighbors for a specific query. The higher relevance score means the nearer neighbor is.

The monotonic property is desirable, but not always true of various HAC strategies. Which strategies then, hold this property? Lance et al. [8], [9] answered our question that single-linkage, complete-linkage strategies are monotonic by definition. Moreover, they introduced a generalized recurrent formula including all special cases, defined as

$$d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} + \gamma |d_{hi} - d_{hj}|, \qquad (1)$$

where the parameters α_i, α_j, 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 $\alpha_i+\alpha_j+\beta \geqq 1$ and $\gamma=0$. Furthermore, they derive a flexible strategy by the quadruple constraint ($\alpha_i+\alpha_j+\beta=1$, $\alpha_i=\alpha_j=\alpha$, β, $\gamma=0$). This constraint suggests a set of monotonic strategies such that as α increases from 0 to 1, the hierarchy changes from an almost completely "chained" system to one with increasingly intense clustering. A given set of queries may now, by varying the parameters, be made to appear as sharply clustered as a user may desire.

3.2 Two-phase Algorithm

The advantages of HAC, however, come at the cost of lower efficiency. Its complexity is at least quadratic in the number of queries because of the distance matrix of all pairs of queries. Loading the entire matrix to the memory will speed up the clustering process. Moreover, one observation from our experiments suggests that the query-URL bipartite graph is an unconnected graph which may be subdivided into connected subgraphs, and the sizes of these subgraphs (clusters) are very skewed. These influence our algorithm to partition the original graph before applying the HAC stragegies. Thus we first extract connected components (also called affinity subgraphs here) by the disjoint-sets data structure [2], and then perform HAC only on the partition containing the query nodes. The pseudo code is described in Table 1.

Table 1 Two-phase Algorithm

Input: query-URL data in the form of (query, URL), a HAC strategy chosen by a user, and an input query.
Output: an ordering of latent neighbors of the input query.

Affinity Subgraph Extracting Phase(Off-Line):

1. If a query q appears with URL u, then place an edge in graph G between the corresponding vertices in Q and U;
2. On the basic initialization of the disjoint-sets structure [2], each vertex in G is in its own set;
3. The connected components are calculated based on the edges in G, and update the disjoint-sets structure when edge(q,u) is added into the graph;
4. Extract the connected components, also called affinity subgraphs here;

Hierarchical Clustering Phase(On-Line):

5. Find the affinity subgraph containing the input query;
6. Apply the selected HAC strategy on this affinity subgraph;
7. The successive merge operations on the input query naturally form an ordering of latent neighbors;
8. If the user is unsatisfied with these neighbors, selects another HAC strategy; Go to 6;
   Else end this searching process.

The Off-Line part extracts all the connected components (Line 2 $\sim$ 4) to form affinity subgraphs as shown in Figure 1(b), a kind of new subgraphs that we derive from the original graph. It is not bipartite, and its nodes correspond to queries only. The edges connect pairs of queries, and are weighted according to a distance score measured by

$$d(q_i, q_j) = 1 - \frac{U(q_i) \cap U(q_j)}{U(q_i) \cup U(q_j)}. \qquad (2)$$

where $q_i(q_j)$is a query, and $U(q_i)$ $(U(q_j))$ is the set of URLs returned by a search engine on query $q_i(q_j)$. The value of the defined distance between two queries lies in the range $[0 \cdots 1]$: 0 if they are exactly the same URLs, and 1 if they have no URLs in common. The distance matrix for each affinity subgraph consists of the distances of the pairs of queries.

Thus our first step has already completed the same clustering task as studied in [1] which exactly used single-linkage clustering to find connected components in the entire graph. The severe termination condition, however, caused the skewness of cluster sizes. Moreover, in the stage of query recommendation, from a cluster that contains query q, this study selected members that occurred most frequently in the whole clickthrough records. But the frequency is not a good descriptor of similarity because it does not consider the tar-

geted query. For some clusters with large sizes, choosing the nearer neighbors from a large number of candidates becomes an issue. The monotonic merge operations of HAC strategies figure out a way to solve it. On the second phase, our HAC strategies on each affinity subgraph (Line 6) are as follows.

Single-linkage clustering (HAC_S):
$d_{hk} = MIN[d(q_h, q_k)]$.

Group-average clustering (HAC_G):
$d_{hk} = \frac{\sum_{q_h \in n_h} \sum_{q_k \in n_k} d(q_h, q_k)}{n_h n_k}$.

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

In the Appendix we give the proof that the HAC_G holds the monotonic property. Complete-linkage clustering is also monotonic, but it is not suitable in this situation as shown in Figure 1(a). $d(q_1, q_2) = 0.75$, $d(q_2, q_3) = 0.667$, and $d(q_1, q_3) = 1$. The complete-linkage clustering considers the distance between one group (query) and another group (query) to be equal to the greatest distance from any member of one group to any member of the other group, and then finds the closest (shortest distance) pair of groups and merges them into a single group. Therefore, q_1 and q_3 will not be in the same group until all the queries are clustered into a single group. But it is apparent that q_3 is a latent neighbor of q_1 for there is a path from q_1, q_2, to q_3.

We last specify the procedure of forming the recommended ordering of latent neighbors (Line 7, 8). In HAC, an input query and a candidate query will come together at a distance (d_{ic}) between the two groups which are being merged and contain the two queries, respectively. At a distance (d_i), the input query is merged with a group (query) the first time, and the candidate query is at a distance(d_c). Then, the distance score between the two queries is equal to $|d_i - d_{ic}| + |d_c - d_{ic}|$ which ranks each candidate (latent neighbor) in the affinity subgraph that includes the input query. The pseudo code of ranking is described in Table 2. In each iteration, the two most similar clusters are merged (Line 11 $\sim$ Line 13) and the rows and columns of the merger cluster i in D are updated (Line 14 $\sim$ Line 18). Ties in HAC are broken randomly. The clustering is stored as an N by 2 matrix in M, where N is the number of candidates. Row i of M 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. I indicates which clusters are still available to be merged. H stores the combination distances between merging clusters at the successive stages.

Table 2 HAC-based Rank Mechanism

Input: N candidates from the affinity graph of the input query, a HAC strategy chosen by a user

Output: An ordered list of related queries to the input query

Distance matrix
1. for k=1 to N
2. for l=1 to N
3. $D[k][l] = dis(q_k, q_l)$

Initialization
4. $H[N]$ (for combination distances)
5. $M[N][2]$ (for collecting merge sequences)
6. $O[N]$ (for the ordered list)
7. for k=1 to N
8. $I[k] = 1$ (keeps track of active cluster)

Clustering
9. for k=1 to N
10. Begin Loop
11. $(i, j) = argmin_{(i,j) l \neq m, I[i]=I[j]=1} D[i][j]$
12. M.append($< i, j >$)
13. H.append((i, j))
14. for h=1 to N
15. Begin Loop
16. $d_{hk} = \alpha_1 D[h][i] + \alpha_2 D[h][j] + \beta D[i][j] + \gamma |D[h][i] - D[h][j]|$
17. $D[i][h] = D[h][i] = d_{h(i,j)}$
18. End Loop
19. $I[j] = 0$ (deactivate cluster)
20. End Loop

Rank Mechanism
21. if $M[i][j] == -iq$
22. $d[iq] = H[i]$ (the first merge for iq)
23. for cq=1 to N ($cq \neq iq$)
24. Being Loop
25. if $M[i][j] == -cq$
26. $d[cq] = H[i]$ (the first merge for cq)
27. if row i of $M[N][2]$ are clusters that include cq and iq
28. $d[(iq, cq)] = H[i]$
29. $O[cq] = |d[iq] - d[(iq, cq)]| + |d[cq] - d[(iq, cq)]|$
30. if $O[cq] > 0.2$, delete it
31. End Loop
32. Sort $O[N]$ in increasing order

4. Experiments

4.1 Data Sets

In the experiments, we used two data sets: Mixture1 and Mixture2. The former is a mixture of Medline (1033 medical abstracts and 30 queries), Cranfield (1400 aeronautical system abstracts and 225 queries), and Cisi (1460 information retrieval abstracts and 111 queries) collections(注1). The latter is a mixture of 800 labelled queries of KDD cup 2005 dataset, and 100 queries selected from our previous

(注1) : These document sets can be downloaded from ftp://ftp.cs.cornel.edu/pub/smart.

experiments [10]. Given each of the total 900 queries, the top 20 URLs of search results were offered by Google API (http://code.google.com/apis/). We avoid significant preprocessing of URLs and queries except for mapping the query characters to lowercase. Clearly, however, even a minimal amount of preprocessing could help the clustering much. For instance, the query "costa & rica" contains a "&", which distinguishes it from the semantically identical "costa rica". We want to underscore the nature of the proposed algorithm.

4.2 Experiment1 on Mixture1 Data Set

To investigate the existence of the latent relationship and the possibility of forming the affinity subgraph, a preliminary experiment was carried out. On the first phase of our algorithm, the three collections in Mixture1 showed differently. In the Medline collection, no affinity subgraph of queries was mined. The reason is that the related abstracts of any pair of queries had not been overlapped during the initialization of distance matrices. For the Cisi collection, the result was exactly opposite to that of the Medline collection. A dominating affinity subgraph included 76 queries of the total 111 queries, and the rest of the queries were unadjacent. Due to page limitations, we only illustrate the results of the Cranfield collection in Figure 2. The sizes of the extracted affinity subgraphs in it were somewhat skewed, but were still good for our task. On the second phase, the various HAC strategies were operated on the Cranfield collection. See Figure 3 and Figure 4 to observe the results of an affinity subgraph (cluster) pointed by an arrow in Figure 2. The structures of the cluster changed in accordance with the variable α. In brief, our preliminary experiment confirms the assumption addressed in the beginning of this paragraph, and that the user-controllable scheme is applicable with the help of the parameters in Equation 1.

4.3 Experiment2 on Mixture2 Data Set

In this section, we report three of our cluster results selected from the different strategies we performed: "food history", "piano moving", and "aquarium". Table 3, Table 4, and Table 5 do not contain all latent neighbors, but a top ranked selection. The relevance scores of queries in the brackets to an input query are tied. "Mexican recipes" and "Irish food recipes" were the nearest neighbors to the query "food history", for it is easy to understand that the food culture involves the characteristics of the nationality. We further checked the two returned lists of URLs on the two neighbors, and found there was no overlap. However, they shared common URLs with the query "food recipes", which connected them. (recalling that our algorithm only considers the connectivity of a graph, the common term "recipes" is useless here). This example verifies our assumption as well. In Table 4, although the query "piano moving" shared the common term and URLs with "moving" and "home moving", two companies with the home moving service were the winners. Here we repeat that the winner is nondeterministic because of the different information needs of users. Our approach, however, can bridge the gap between the determinacy of similarity values and the indeterminacy of users' needs. The last example in Table 5 showed changes of neighbors through different HAC strategies. Under $\alpha = 0.3$ the top was "city tampa", while "clearwater beach Florida" became the nearest one by the single-linkage clustering. Another observation was that the clustering turned more intense, and more pairs of neighbors were constructed, when the value of α increased. Our future research will mine other features of this new relationship.

Table 3 The top nearer neighbors of the query "food history"

HAC Strategy	Query: food history
HAC_F($\geqq$ 0.6), HAC_S	(Mexican recipes, Irish food recipes), chicken recipe website, (food recipes, all recipe.com)
HAC_F($\leqq$ 0.5), HAC_G	Irish food recipes, Mexican recipes, chicken recipe website,(food recipes, all recipe.com)

Table 4 The top nearer neighbors of the query "piano moving"

HAC Strategy	Query: piano moving
HAC_F($\geqq$ 0.24), HAC_S, HAC_G	(allied van line, united van line), (moving, home moving)
HAC_F($\leqq$ 0.23)	united van line, allied van line, (moving, home moving)

Table 5 The top nearer neighbors of the query "aquarium"

HAC Strategy	Query: aquarium
HAC_S	clearwater beach Florida, (tampa fla, city tampa), st.petersburg
HAC_F(=0.5)	clearwater beach Florida, city tampa, tampa fla, st.petersburg
HAC_F(=0.1, 0.2)	tampa fla, city tampa, tampa florida, tampa bay florida,tampa fla
HAC_F(=0.3)	city tampa, tampa fla, (tampa florida, tampa bay florida), st.petersburg
HAC_F(=0.4)	clearwater beach Florida, (tampa florida, tampa bay florida), city tampa, tampa fla, clearwater beach Florida, st.petersburg
HAC_F(=0.6)	clearwater beach Florida, st.petersburg, city tampa, tampa fla
HAC_F(=0.7, 0.8)	(clearwater beach Florida, st.petersburg), city tampa, tampa fla, (tampa florida, tampa bay florida), (st petersburg, saint petersburg)
HAC_F(=0.9)	(clearwater beach Florida, st.petersburg), (city tampa, tampa fla), (tampa florida, tampa bay florida), (st petersburg, saint petersburg)

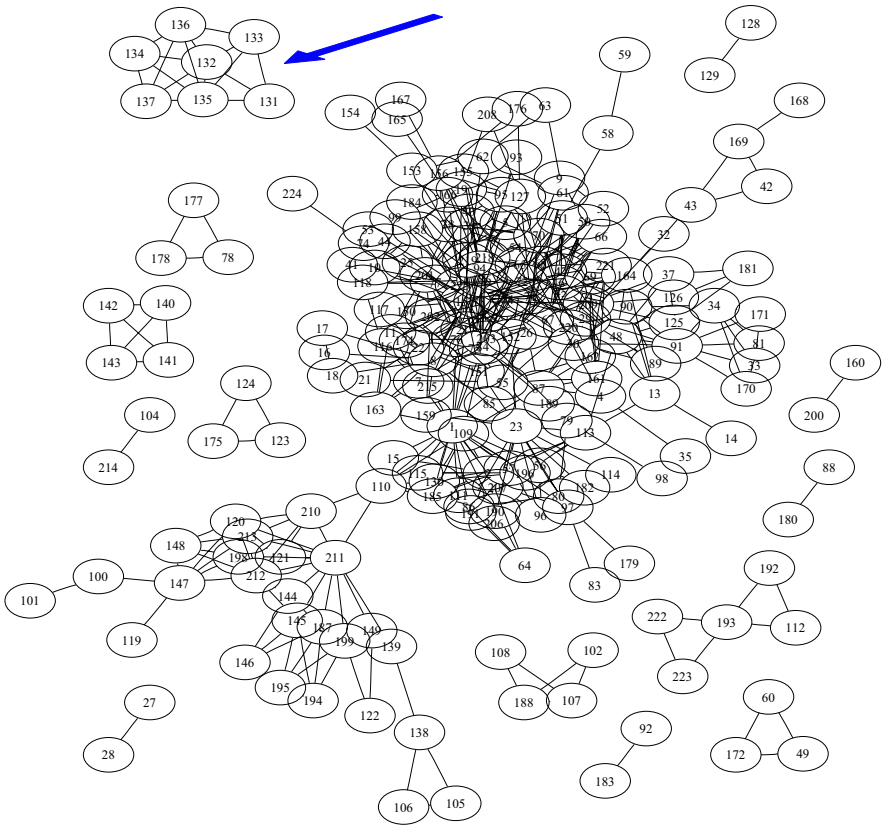

Figure 2 Affinity Subgraphs from the Cranfield collection

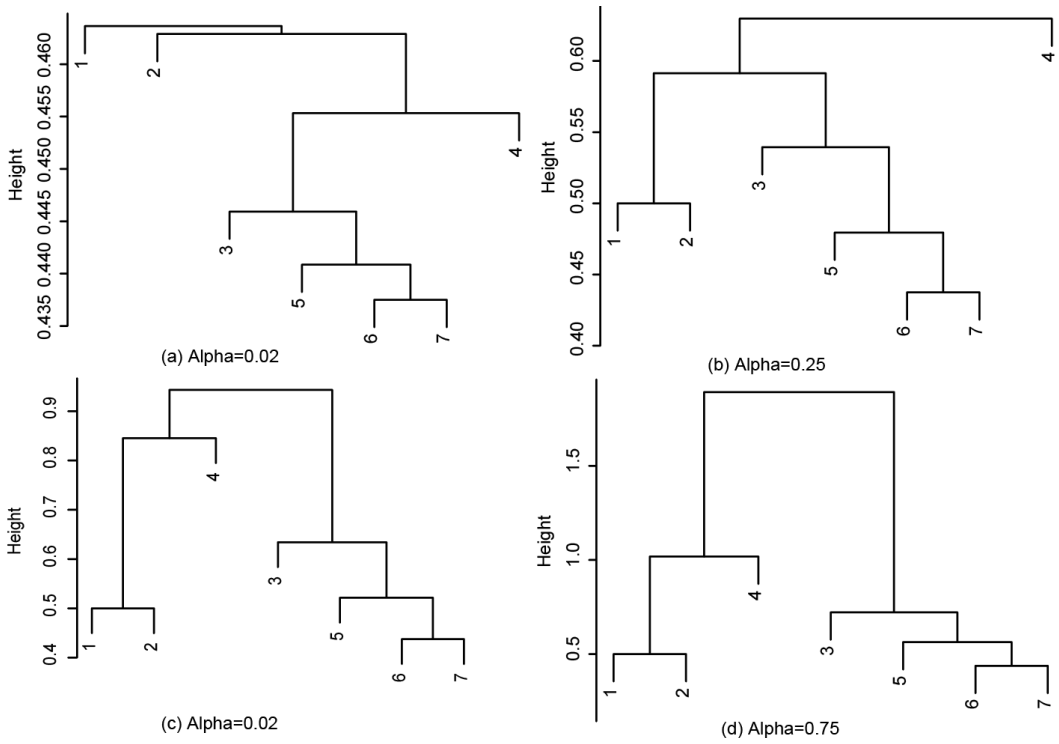

Figure 3 Flexible Strategy on the Cranfield collection: $\alpha = 0.02, 0.25, 0.5, 0.75$ in (a), (b), (c), (d), respectively

5. Related Work

The idea of exploiting the collaborative knowledge of users, embodied as a set of search queries, is not new. Glance et al. [5] introduced a software agent that collected queries from previous users, constructed a query graph, and recommended

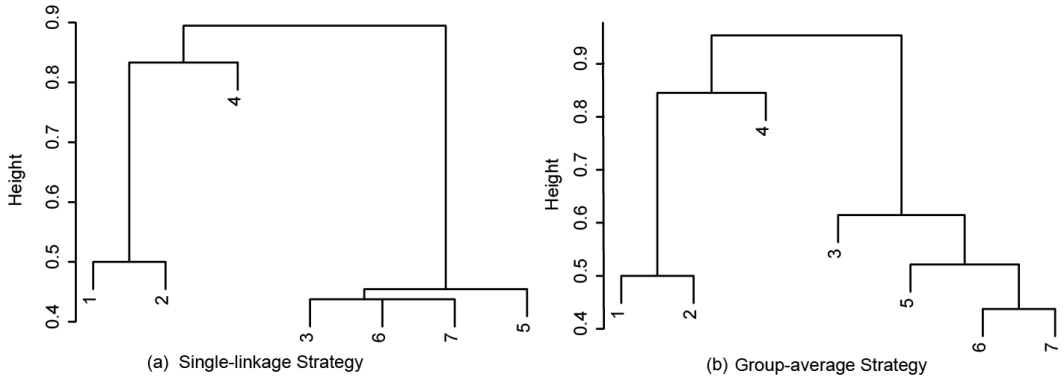

Figure 4 (a) Single-linkage and (b) Group-Average Strategy on the Cranfield collection

related queries. Query clustering is a less explored problem than web page clustering, but it is a principle alternative for query recommendation. Wen et al. [14] suggested that clustering similar queries could provide effective assistance for human editors in discovering new FAQs. Hansen et al. [6] distilled the search-related navigation information from proxy logs for clustering queries. Traditional studies [6], [12], [14] deemed that the similarity between two queries is determined by the query or web page contents, the overlap of search result lists retrieved by a search engine or clicked search results by searchers, and so on.

On the other hand, the query-URL relationship can be represented by a bipartite graph as modelled in Section 3.1. From a graph theoretical point of view, SimRank [7] measured the object-to-object relationship by scoring recursively the similarity of their related objects. Sun et al. [13] employed random walk with restarts and graph partitioning to solve two problems, neighborhood formation and anomaly detection. Beeferman et al. [1] viewed the click through data as a bipartite graph, and utilized an iterative, agglomerative clustering algorithm to the vertices of this graph for clustering queries, and URLs respectively. Our problem is also close to the co-clustering problem in a more general way [3], [4].

The above studies either select several members from a cluster the query belongs to, or compute the query-to-query similarities to return the queries with higher scores to a specific query. Though the latter is more reasonable that the former, our algorithm is separated from them. Users are able to dynamically manage the structure of HAC, and thus find the latent neighbors of a query in terms of their current information needs.

6. Conclusion

In this paper, latent neighbors of a query are defined as queries between which and the target query there exists paths connecting them. We propose a user-controllable scheme to solve the problem of finding such neighbors from query-URL data set. It is realized by an algorithm that has two major steps. Under the first phase, we extract a set of query clusters from the given bipartite graph. Under the second phase, a flexible HAC strategy plus single-linkage and group-average clusterings are employed to each affinity subgraph. Our results are very encouraging: many similar queries without sharing any common terms and URLs have been grouped. In addition, under the flexible strategy in HAC, users are able to pick up different neighbors through a parameter adjustment. This study demonstrates the usefulness of graph connectivity and the feasibility of our scheme to help users optimally represent their information needs for search engines.

Our work is still ongoing, and there is certainly room for further improvement: 1) the interface of the query recommendation should be visualized, pushing our work into application stage; 2) a user study will be put into practice to evaluate the performance of our scheme; 3) using large data sets will further test the scalability and efficiency of our algorithm.

Appendix

Proof. In [8] a strategy with the constraint ($\alpha_i+\alpha_j+\beta \geqq 1$ and γ=0) has monotonicity. For Group-average clustering, the condition $n_k = n_i + n_j$ gives:

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

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.

References

[1] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *Proc. of the Sixth ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining (KDD'00)*, pages 407–416, Boston, MA, USA, 2000.

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

[3] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proc. of the Seventh ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD'01)*, pages 269–274, San Francisco, CA, USA, 2001.

[4] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. of the Ninth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD'03)*, pages 89–98, Washington, DC, USA, 2003.

[5] N. S. Glance. Community search assistant. In *Proc. of the 2001 Int'l Conf. on Intelligent User Interfaces (IUI'01)*, pages 91–96, Santa Fe, NM, USA, 2001.

[6] M. Hansen and E. Shriver. Using navigation data to improve IR functions in the context of web search. In *Proc. of the 2001 ACM CIKM Int'l Conf. on Information and Knowledge Management (CIKM'01)*, pages 135–142, Atlanta, Georgia, USA, 2001.

[7] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proc. of the Eighth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD'02)*, pages 538–543, Edmonton, Alberta, Canada, 2002.

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

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

[10] L. Li, Z. Yang, B. Wang, and M. Kitsuregawa. Dynamic adaptation strategies for long-term and short-term user profile to personalize search. In *Proc. of the joint Conf. of the 9th Asia-Pacific Web Conf. and the 8th Int'l Conf. on Web-Age Information Management*, Huang Shan, China, to appear, 2007.

[11] Lyman, Peter, and H. R. Varian. "How Much Information?" Retrieved from http://www.sims.berkeley.edu/how-much-info-2003 on 2006.

[12] S. Otsuka and M. Kitsuregawa. Clustering of search engine keywords using access logs. In *Proc. of Database and Expert Systems Applications, Int'l Conf. (DEXA'06)*, pages 842–852, Kraków, Poland, 2006.

[13] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Proc. of the 5th IEEE Int'l Conf. on Data Mining (ICDM'05)*, pages 418–425, Houston, Texas, USA, 2005.

[14] J.-R. Wen, J.-Y. Nie, and H. Zhang. Clustering user queries of a search engine. In *Proc. of the Tenth Int'l World Wide Web Conf. (WWW'01)*, pages 162–168, Hong Kong, China, 2001.