The Application of Users' Collective Experience for Crafting Suitable Search Engine Query Recommendations

Faezeh Ensan[†], Ebrahim Bagheri[†], Mohsen Kahani[*]
[†] *University of New Brunswick, Fredericton, Canada.*
[*] *Ferdowsi University of Mashhad, Mashhad, Iran.*
faezeh.ensan@unb.ca, e.bagheri@unb.ca, kahani@um.ac.ir

Abstract

Search engines have turned into one of the most important services of the Web that are frequently visited by any user. They assist their users in finding appropriate information. Among the many challenging issues in the design of Web search engines that is mostly related to the design of an adaptive interface is recommending suitable query phrases to the end-users. This has two major benefits: firstly the users can more easily interact with the Web search engine and secondly get hints on what is more apt to look for in cases where they may not have any clue. In this paper, we propose a graph based query recommendation algorithm that sequentially recommends query terms to its users. The most important notion behind the design of the algorithm is that the past behavior of previous users of a search engine is mined and a multi-segmented graph is built. Recommendation is made based on the relative similarity of query terms, their frequency and conceptual closeness in the graph.

Index Terms— Search Engines, Query Recommendation, User Assistance

1. Introduction

The considerable growth in the number and diversity of the information available on the Web significantly increases the role of search engines. Although search engines can help users find appropriate websites, but it is the users themselves that affect the final results of the search engines by providing the initial search queries. Users usually have the tendency to use only a few query terms in their query phrases for searching their intended information. A study shows that the average query phrase length is about two words [1]. These short queries increase the chance of mismatch between the information that the user needs on the one hand, and the results that have been returned by search engines on the other hand. Another common problem is that users in some cases are not familiar with the domain they are searching or cannot find the appropriate vocabulary which correctly describes their intention. Therefore, not only are search engines unable to find correct and suitable results in such cases, but may also seriously perplex their users.

In order to overcome these problems and to enhance the performance of search engines, several methods have been suggested. Some techniques focus on proposing totally new query phrases different from those that have been entered by the user. The new query phrases are proposed so that it is more likely that the user reaches his required information through the employment of the proposed query phrase. Calculation of the similarity between different query phrases is the essential point in this approach. Similarity is mainly derived from mining search engine query logs. Wen et al. [2] propose a clustering method based on query logs. In their method, they consider similar queries as queries that have returned the same set of web pages which have been ultimately selected by the end users. In addition, they also consider the content of the web pages that have been clicked after a query to estimate their mutual relevance.

Baeza-Yates et al. [3] suggest a new clustering method for semantically clustering queries. In their method, a term-weight vector is assigned to each query, based on the terms available in the clicked URLs. The weight of each term in this vector has direct relation with the frequency of the term in that specific URL as well as the popularity of the URL in the set of all URLs which have been clicked for this query. The similarity of two queries is then calculated based on their term-weight vector and through the application of a cosine function.

Several query expansion techniques have also been developed in the recent years. Some of these methods make use of the content of the documents to select expansion terms, while others aim at mining past user queries to find appropriate query expansion terms. Methods that are based on global and local document analysis can be categorized into the first approach. The aim of global techniques is to inspect the whole document space and find out the relationship and association between all of its terms, and then expand any query phrase with the use of the obtained information [4]. Qiu and Frei [5] suggest a global query expansion method that makes use of similarity thesaurus for representing term association. Since global methods analyze all of the documents and detects the related terms, they guarantees more robustness; however, the speed and the required disk space in a huge document set where documents can frequently change, get eliminated or created can be a big drawback for such approach.

Alternatively, local document analysis methods only take into account a restricted number of documents. They use some of the top ranked documents to construct the thesaurus instead of processing the whole document space. Relevance feedback [6] is sort of local document analysis for determining the top ranked documents according to the web pages which are selected by the user. In order to overcome the fact that the users usually do not give correct and relevant feedback, blind feedback was suggested rather than the relevance feedback [7]. This type of feedback considers the *n* top ranked documents as relevant. Local Context Analysis [4], that is a technique for expanding users' query phrases, makes use of this type of feedback.

The other approach in query expansion focuses on the past queries that the users have entered into the system. Cui et al [8] propose a new scheme that assesses user query logs for extracting suitable terms. Actually, this method attempts to make use of both query log space and document space. Query terms are linked to document terms according to the choice of the user which has been logged at the time that the user had clicked on a particular document for each query. Normalized weight of a query term in a related document as well as the frequency of the pair of query-documents in the query log has a direct affect on the correlation between the terms of the document and the query. Based on this correlation, new query phrases can be analysed and appropriate terms can be recommended.

In this paper, we propose a new query recommendation method. In our proposed method, the relationship among query terms is maintained by a graph. Query terms are nodes of this graph and an undirected link between two nodes represents the occurrence of the associated terms in a same query phrase. Links related to the terms that have frequently co-occurred have a higher weight. In fact, the hypothesis behind our method is that the terms which often come together in a same query phrase have conceptual similarity. Also for reducing the affect of immature phrases, the weight of each link between two query terms is reduced periodically.

In the next section, we will thoroughly explain our proposed query recommendation method which is based on a query graph. In Section 3, we describe our test setting, and implementation environment. After that we compare the obtained results from our proposed recommendation algorithm with a query expansion method. In the end, we conclude the paper with some final remarks.

2. The Proposed Algorithm

Most search engines maintain user query logs that consist of query phrases that their users' have entered. Some of the previous methods (e.g. [8]) have made use of this logged information in order to detect the relationship between query terms and web documents. This relationship is derived based on the target of the search engine result that has been selected by the user for any given query. In our approach, we exploit the query logs from a different point of view. We only focus on the query terms and the frequency of their co-occurrence. Through a graph structure, we maintain the relationship and depict the power of this relationship between each query term. In this graph, each node represents a query term. The links between the nodes of the graph represent the degree of co-occurrence of two query terms with each other.

As the users enter their query phrases into the search engine, the graph gets populated incrementally. Whenever a new query term is visited that does not exist in the graph a new node will be added, and it will be connected to the other nodes based on its occurrence (It is obvious that the results obtained from our method are dependant on the degree of query graph completeness. The more complete the log and the graph are, the better the recommendations will be). Since the graph may gradually evolve, it is regularly trimmed for three major reasons:

1. To prevent an enormous data structure and intractable memory consumption,
2. To decrease the impact of older query phrases on more recent queries,
3. Understand the present user interest's trend and focus.

As mentioned before, the terms that co-occur in previous queries have a higher chance of being semantically related and hence are more likely to appear in the same query. In our graph, we define a cost factor for each link that is inversely related to the number of co-occurrence of two related terms in the same query.

For recommending a new query term, if a user only enters one query term, according to our approach, the adjacent node to the current term with the minimum link cost would be an appropriate recommendation choice. But this process will be more complicated as the user enters more than one query term. In the following subsections we will explain the structure of the recommendation method in more detail.

Graph Construction

Every query phrase, *QP*, in a query log can be presented as:

$$< t_1, t_2, ,, t_q >; \quad 0 < q < NtQP \qquad (1)$$

t_i depicts the i^{th} term in the query phrase and NtQP is the Number of query terms in the Query Phrase. For every query term, t_i, in a query phrase, an associated node exists in the Graph *G*. If the node is not already present in *G* it will be inserted.

After all of the available query terms present in a newly entered query phrase have been added to *G*, a link will be drawn between any of the two query terms in that query phrase. This link is created regardless of the order of the terms in the query phrase. As an example, let's consider "Windows XP Microsoft" as a newly entered query phrase. Windows, XP, and Microsoft are the query terms that are available in the query phrase. Each of these three terms are added to *G* and are connected to each other through an undirected link with an initial weight ($W_{e,t}$: weight of edge *e* at time *t*) equal to *one*. In this case, since there are three query terms in the query phrase, and they should be connected to each other, (3*2)/2 = 3 links need to be created (In a general case for a query phrase with *n* query terms, *n(n-1)/2* links should be added to *G*).

In cases where the nodes related to the query terms and the edges between them already exist, they will be employed and therefore no other links or nodes are created. Here, the only alteration to *G* would be that the weight of the links would be incremented by one ($W_{e,t+1} = W_{e,t} + 1$).

We define an additional metric to evaluate the correlation between the query terms. For each edge, *e*, between two query terms, the related cost is defined as the inverse value of its weight at time *t* ($W_{e,t}$):

$$Cost_{e,t} = 1/W_{e,t} \qquad (2)$$

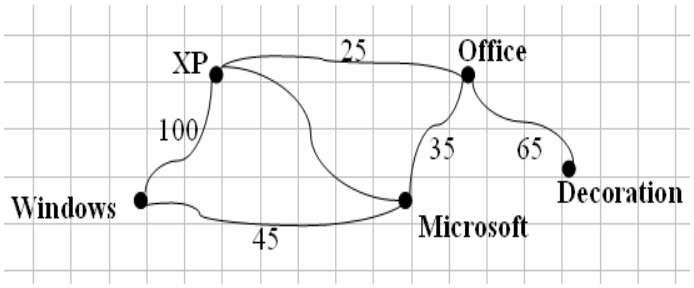


Figure 1. A Sample Graph Showing Five Different Query Terms Appearing in Different Query Phrases. For Example "XP" and "Office" Have Been Jointly Used in 25 Query Phrases.

To decrease the value of older relationships between different query terms, the cost of each edge is incremented by a constant value (φ) periodically. Therefore for each period (τ) we have:

$$Cost_{e,t+\tau} = Cost_{e,t} + \varphi \qquad (3)$$

Concurrently, a trim function evaluates the cost values of each edge in *G*. The edges that fall below a certain threshold at a given time, *t*, (λ_t) will be regarded as redundant and will hence be discarded. Various threshold values can be selected for this purpose. It can be either a static value or be adjusted adaptively based on different circumstances. A good choice for the threshold would be the mean value of the edge costs with a tolerance value equal to the mean standard deviation which is based on the Chebyshev's theorem [10].

$$\lambda_t = \frac{\sum_{i=1}^{numberofedges} Cost_{i,t}}{numberofedges} + STD(Cost_{i,t}) \qquad (4)$$

$$\forall edge(e_i) \;\; if\, Cost_{e,t} > \lambda \;\; then \;\; erase(e_i) \qquad (5)$$

Having removed certain edges between various nodes, some of the nodes may end up dangling as separated disjoint entities that have no connections to any other node. The trim function also removes such nodes that represent the query terms that are not frequently used.

For the small graph shown in Figure 1, the number of edges is 5, and the aggregate cost of the edges is:

$$\frac{\sum_{i=1}^{numberofed\,ges} Cost_{i,t}}{numberofed\,ges} = \frac{\left(\frac{1}{100} + \frac{1}{45} + \frac{1}{25} + \frac{1}{35} + \frac{1}{65}\right)}{5} = 0.0232$$

Therefore, the adaptive threshold for eliminating redundant edges would be:

$$\lambda_t = 0.0232 + \mathrm{STD}(\mathrm{Cost}_{i,t}) = 0.0349$$

In this example, the edge between the query terms "XP" and "Office" would be removed since it is higher than the threshold (0.04 > 0.0349). The rest of the edges will be kept.

The final result of the graph construction process is a graph *G*, with a set of nodes that represent the most frequently used query items along with their strong correlations (co-occurrences higher than a given threshold) with other query terms. It should be noted that *G* may be a collection of *n* disjoint sub-graphs. This means that the query terms in each of these sub-graphs have close relationship with each other and show no closeness to the query terms used in the other sub-graphs.

Based on this graph construction scheme, we introduce a multi-depth query recommendation algorithm in the next sub-section. The most important concept behind the algorithm is that we believe that the aggregate behavior of all the users entering query phrases into a search engine conveys a correct semantic meaning.

Recommendation Algorithm

As we explained earlier in this paper, a constructed graph may consist of many different disjoint sub-graphs. Let's assume that the query term that the user has entered into the search engine is $Q=\{t_1,..t_q\}$. Each of the query terms t_i can belong to any of the n disjoint sub-graphs. For instance, n=1 represents a situation where all of the query terms belong only to one of the sub-graphs. Consider Sub-graph SG_i, $0<i<n$, and the query term t_j, $0<j<q$, for every adjacent neighbor nb_{ijk} to t_j in SG_i, we define a related Cumulative Distance Cost, CDC as:

$$CDC_{ijk} = \frac{\sum_{m=1}^{q} djsCst(nb_{ijk}, t_m) * djsLnt(b_{ijk}, t_m)}{\sum_{m-1}^{q} djsLnt(nb_{ijk}, t_m)} \quad (6)$$

Where djsWt(a,b) is a function that calculates the cost of the shortest path from node *a* to node *b* using the Dijkstra algorithm and djtLnt(a,b) determines the length of the shortest path that has been identified by the Dijkstra algorithm.

For all the adjacent nodes to any of the query terms, t_i, in the query phrase, one node for each SG_i with the minimum cumulative distance is specified. These nodes are added to a set named the Candidate set. Since there are at most *n* sub-graphs in the query graph, the size of the Candidate set can at most reach *n*. This occurs only when the user enters *n* different query terms in his query phrase that are present in *n* different sub-graphs.

There are four options for recommending a proper query item to the user based on the calculated cumulative distance cost values of the adjacent nodes to the entered query items:

1. The first option is to rank order all of the members of the Candidate set in an incremental fashion based on their cumulative distance cost value. The member with the minimum CDC value (See Equation 7) is regarded as the most appropriate choice for recommendation

$$Candidate = Min_{i=1}^{n}(CDC(Candidate(t_i))) \quad (7)$$

2. It is also possible to rank order the members of the Candidate set based on their CDC value. All of the members of the Candidate set are then recommended to the user in an ordered way. The user would then have the option to select from among these recommendations.

3. Regardless of the Candidate set, we can order all of the adjacent nodes to any of the entered query terms based on their CDC value. The major difference between this method and the one previously introduced is that in this method more than one query term may be recommended to the user from the same sub-graph; while in the previous method, every sub-graph has only one representative.

4. In order to value the sub-graphs with more adjacent nodes to the entered query terms, we can divide the cumulative distance cost of each query item by the number of the query items in that sub-graph . In this way, the sub-graphs that accommodates more query items entered by the user will receive more attention. The Weighted Cumulative Distance Cost (WCDC) is a measure designed for this purpose and is calculated as follows:

$$WCDC_{ijk} = \frac{\sum_{m=1}^{q} djsCst(nb_{ijk}, t_m) * djsLnt(b_{ijk}, t_m)}{|member(QP, SG_i)| \sum_{m-1}^{q} djsLnt(nb_{ijk}, t_m)} \quad (8)$$

In Equation 8, |member (QP, SG_i)| denotes the number of query terms of the query phrase *QP* in sub-graph SG_i. In our experiments, we have used this method for recommending query terms to the end users.

To clarify how the recommendation algorithm actually performs, we go through a short example. Let's suppose that a user has entered "Office Microsoft Information" into the search engine as his intended query phrase. Based on our sample graph (Figure 2), we have the CDC and WCDC values shown in Table 1.

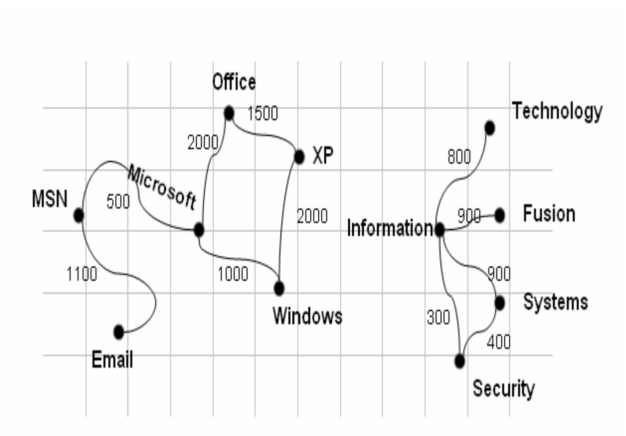


Figure 2. A Sample Graph Representing the "Office Microsoft Information" Query Phrase and its Adjacent Nodes

The recommendations made for any of the four approaches to query term recommendation would respectively be:

1. The Candidate set would be the nodes from each sub-graph with the lowest CDC. Therefore, the Candidate set would be:
   Candidate set = {XP, Systems, Fusion}
   The member of the Candidate set with the minimum CDC will be selected as the term to be recommended to the end user, which will be "XP".
2. In the second method, the members of the Candidate set are ranked based their CDC value. This will result in the recommendation of an ordered list of 1. XP, 2. Fusion, 3. Systems.

3. The third method ranks the query items regardless of their existence in the Candidate set based on their CDC value. Therefore, the recommendation list is: 1. XP, 2. Windows, 3. Fusion, 4. Systems, 5. Technology, 6. MSN, 7. Security.
4. Finally, in the fourth approach that we have also selected in our implementation, the query terms in the Candidate set are rank ordered based on their WCDC value. The final output of this method would hence be 1. XP, 2. Fusion, 3. Systems.

It is clear that the results of this example may not be so much rational since the graph that we have used for our example is too small and does not represent actual edge weights. In the following section, we will discuss the results obtained from an authentic implementation of our proposed recommender algorithm. The results will also be compared with that of another technique.

Table 1: The Related CDC and WCDC for each of Query Term's Adjacent Nodes

Adjacent Nodes	Query terms				CDC	WCDC
	Microsoft		office			
	djsCst	*djsLnt*	*djsCst*	*djsLnt*		
Windows	0.001	1	0.0012	2	0.0011	0.00055
MSN	0.002	1	0.0025	2	0.0023	0.00115
XP	0.0012	2	0.0006	1	0.001	0.0005
	Information					
	djsCst		*djsLnt*			
Technology	0.0013		1		0.0013	0.0013
Fusion	0.0011		1		0.0011	0.0011
Security	0.0033		1		0.0033	0.0033
Systems	0.0011		1		0.0011	0.0011

3. Performance Evaluation

In this section, we will describe our test environment, the implementation of our proposed algorithm and compare and discuss our obtained results. We make use of the AltaVista search engine query log for the purpose of our analysis. Our query log processing and graph construction phase has detected 43058 query terms from a subset of the AltaVista query log. The constructed graph comprises 145601 edges between its nodes.

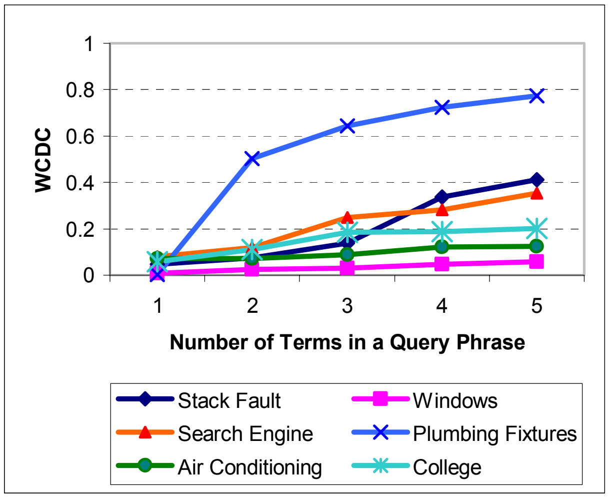
For computerizing the graph structure and running the base graph algorithms over our dataset, a java package for graph manipulation called JUNG [9] has been used. JUNG is a java package that allows programmers to model, analyze, and visualize their data in the form of graphs.
To evaluate our algorithm, we have selected ten of the most frequently used query phrases in the query log. Studies have shown that users are likely to enter queries with at most two query terms. Therefore the terms that have been selected either contain one or two query terms. Table 2 shows the list of our initial query phrases.

Table 2: The Initial Query Phrases Used for Evaluating the Proposed Algorithm.

1.	Windows	2.	Playstation
3.	Search engine	4.	Advance camera
5.	Plumbing fixtures	6.	Air conditioning
7.	Jedi outcast	8.	Stack fault
9.	e-commerce	10.	College

To evaluate the performance of our proposed algorithm we employ two major criteria namely: the weighted cumulative distance cost, and the precision of the results obtained from the search engine after employing the recommendation algorithm. The WCDC factor depicts the usefulness degree of the recommended query term. The higher the value of WCDC for a recommended query term is, the less worthy it is. This is due to the fact that as the value of WCDC increases, their conceptual distances from the query terms entered by the user also increase. Therefore, since our recommendation algorithm aims at providing the users with the most relevant query terms, terms with high WCDC values are not reasonable. Based on our experiments, we have inferred the fact that as the number of recommendations increase, the WCDC value of the query terms also increases, and therefore their effectiveness decreases. To show this point, we have used the ten query phrases introduced in this section. For each query term we use our recommendation algorithm and make a suggestion. The WCDC value of the new query term is then calculated and recorded. Now with the employment of the new query phrase (which is the result of the concatenation of the old query phrase and the new recommended query term), we make another recommendation, and again the WCDC value is calculated. This process has been repeated five times for ten different query phrases. As it can be seen in Figure 3, the WCDC values of the recommended query terms increases as the length of the query increases. This fact shows that the users need to be cautious in selecting or accepting the recommended query terms, since this will highly affect the final information that they will receive from the search engine which is a natural result of narrowing the search query phrase.

Based on this observation, we conclude that the shorter query phrases that have been gradually created based on the recommendation algorithm have a higher chance of reaching users' requirements and intentions. Therefore, if the user employs too many recommendations, he will start getting misled.



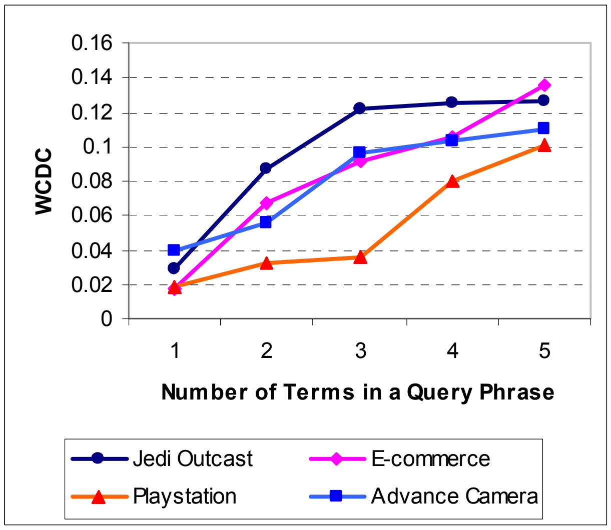


Figure 3. the Related WCDC Values of Each Query Phrase

The other factor that we have employed for evaluating our proposed approach is precision. Precision defines the degree of noise (inappropriate retrieved web pages from the search engine) in the list of returned web pages after sending a query that has been made through our recommendation algorithm. To calculate the degree of usefulness of the returned documents, we have used the TF-IDF metric to calculate the relevance of the returned web page to the submitted query phrase. Table 3 shows the precision of our proposed algorithm compared with both the Local Context Analysis query expansion method [4], and the state where no recommendation or expansion was employed. The smoothing factor of the LCA algorithm has been set to 0.1. The top 50 results returned from the search engine have been used for query expansion in LCA. We have used Google as the base search engine for performing our queries. The first twelve web pages returned from each of the approaches have been used to calculate the precision value. The examinations show that the results obtained from our proposed query recommendation algorithm outperforms the query expansion method (See Table [3]).

Our proposed method also has the advantage that allows the user to see his options and choices and guides him through the query phrase construction process which is not the case in a typical query expansion algorithm.

Table 3. To calculate precision, we have assumed the documents with a TF-IDF value higher than a specific threshold as relevant. The threshold (T) has been adaptively calculated based on the average TF-IDF of all returned documents in the baseline method.

Threshold	Baseline	LCA	Proposed Model
T	0.358333	0.391667 (9%)	0.441667 (23%)
T + 0.05	0.216667	0.183333 (-15%)	0.266667 (23%)
T - 0.05	0.630952	0.619048 (-1%)	0.690476 (9%)
Average	0.401984	0.398016 (-2.3%)	0.46627 (18.3%)

4. Conclusions

In this paper, we have proposed a query recommendation algorithm. The algorithm creates a graph of query terms based on the degree of relevance of the query terms derived from their co-occurrence in different query phrases. As the user enters query terms into the search engine, our proposed algorithm traverses the graph to find the most relevant query terms that can be recommended to the user. The suitability and relevance of each query term has been calculated based on a factor named weighted cumulative distance cost. We have compared the performance of our proposed algorithm with two different states: 1. where the user does not use any sort of assistance, 2. where the user employs a query expansion algorithm. The results obtained from these comparisons have been shown throughout the paper. These results depict reasonable performance for our proposed query recommendation algorithm.

5. References

1. BJ Jansen, A Spink, J Bateman, T Saracevic; Real life information retrieval: a study of user queries on the Web, ACM SIGIR Forum; Volume 32 , Issue 1; Pages: 5 - 17 ; 1998

2. J. Wen, J. Nie, and H. Zhang; Clustering user queries of a search engine, WWW10, May 1-5, 2001, Hong Kong.

3. R Baeza-Yates, C Hurtado, M Mendoza; Query Recommendation using Query Logs in Search Engines, In *Workshop in Web Clustering*, Greece, 2004. Current Trends in Database Technology - EDBT 2004 Workshops, LNCS 3268, Springer.

4. J Xu, WB Croft ; Query Expansion Using Local and Global Document Analysis, In Proceedings of the 19th International Conference on Research and Development in Information Retrieval, pages 4-l 1, 1996.

5. Y Qiu and H Frei; Concept Based Query Expansion. In proceeding ACM SIGIR Conference, pages 160-169, 1993.

6. G. Salton, C. Buckley; Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science, 41,* 288-297, 1990.

7. C. Buckley, G. Salton, J. Allen and A. Singhal; Automatic query expansion using SMART: TREC-3. In: D. K. Harman (ed.), The Third Text Retrieval conference (TREC-3). U.S. Department of Commerce, 1995.

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

9. J. O'Madadhain, D. fisher, P. Smyth, S. White, Y. Boey; Analysis and Visualization of Network Data using JUNG. Available online at http://jung.sourceforge.net/doc/ index.html, February 2005.

10. R. E., Walpole, Elementary Statistical Concepts. Macmillan, New York, 2nd edition, 1983.