Hybrid Query Session and Content-Based Recommendations for Enhanced Search

Zhiyong Zhang and Olfa Nasraoui

***Abstract*— This paper presents a simple and intuitive method for mining search engine query logs to get fast query recommendations on a large scale industrial-strength search engine. In order to get a comprehensive solution, we combine two methods together. First, we study and model search engine users' *sequential* search behavior, and interpret this consecutive search behavior as client-side query refinement, that should form the basis for the search engine's own query refinement process. This query refinement process is exploited to learn useful relations and build fuzzy associative memories that help generate related queries via a fuzzy inference process. Second, we combine this method with a traditional *content* based similarity method to compensate for the high sparsity of real query log data, and more specifically, the shortness of most query sessions.**

I. Introduction

Providing related queries for search engine users can help them find the desired content more quickly. For this reason, many search engines started displaying related search keywords in the bottom of the result page.

Also, for some users who are not very familiar with a certain domain, we can use the queries that are used by previous similar searchers who may have gradually refined their query, hence turning into *expert searchers*, to help guide these novices in their search. That is, we can get query recommendations by mining the search engine query logs, which contain abundant information on past queries. Search engine query log mining is a special type of web usage mining, that can improve a search engine's query answer time and enhance its ranking quality. In this paper, we present a simple and yet well-rounded solution that has already been implemented with success as part of an *industrial-strength* search engine on one of the largest portals (*www.Sina.com*) in China. We further combine the *association or correlation-type* information with the *textual content* between queries in order to ***(i)*** improve coverage, and ***(ii)*** compensate for the short nature of the majority of search engine queries, which results in *sparse* data. Our association-type information between queries are computed in a simple and efficient incremental manner from the query logs, based on a one dimensional graph model, where nodes represent the queries, and edge weights represent a *soft* measure of relation or *consecutiveness* of the queries. A fuzzy relation matrix is built to store the relation between queries. Queries that are submitted immediately one after the other receive a maximal relation value, while this relation value is gradually dampened as the queries move farther apart from each other during the same search session. *Dampening* the relation values as queries become more distant in a session is not only a simple and intuitive way to capture query *relatedness*, but it also acts as a filter that helps decrease the influence of the imperfect session segmentation performed in the pre-processing phase (though this was based on relatively reliable cookies and timeout mechanisms), and thus alleviate problems that can result when even the *same* user *shifts their attention* during the *same* session. Moreover, the fuzzy relation values are accumulated incrementally with each new session to obtain a *global and scalable* model. We use the continuously updated query graph to form a *Fuzzy Associative Memory* [9] that can be used to combine all the uncertain associations between the queries in the historical query stream. In order to do this, we consider every input session to form a fuzzy set [17] over the domain of discourse consisting of all the individual queries. Note that this set is crisp at the input stage, since a user has either typed a particular query or not. However, the advantage of a fuzzy set model will become clear, once we proceed to the output of a Fuzzy Inference system that is designed to produce query recommendations from a sequence of queries. As will be discussed in Section II, much of the related work relies on the click-through data for query log clustering or generating query recommendations. But no work has considered one user's consecutive query behavior *alone and without click-through data*. In fact, one person's consecutive query behavior alone could provide abundant information which can be exploited for generating query recommendation or clustering the queries. Based on common sense, two consecutive queries generated by the same user may generally reflect this user's own query refinement or learning process. Also, using the query log without click-through data could discover clusters that cannot be identified by using the click-through data. And there is no work yet that *combines* the query *content* similarity with the user *query-behavior-related* similarity *together* to generate well-rounded query recommendations.

This paper tries to make a move in this direction as follows: ***(i)*** We present a method of utilizing a user's consecutive query behavior combined with content based similarity measurement for generating related queries. ***(ii)*** For the content based similarity analysis, since we do not record the clicked documents, we cannot get the term frequency of the query

Zhiyong Zhang is with Dept. of Computer Science and Engineering, University of Louisville, Louisville, KY 40292, USA (email: z0zhan13@louisville.edu). Part of this work was done while the first author was at Search Engine R&D Group, SINA Corporation, Beijing, China.

Olfa Nasraoui is with the Dept. of Computer Science and Engineering, University of Louisville, Louisville, KY 40292, USA (email: olfa.nasraoui@louisville.edu).

Partial support for this work was provided by National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui.

term for assigning the weight factor when using the TF*IDF method. Hence, we use the search frequency (SF) instead. This proves to be a better replacement of term frequency (TF) since it also reflects the user's cumulative support factor. For the inverse document frequency (IDF), we rely on the whole web searching document index, which is available for use. Then we use what we termed an *SF*IDF* method for weighting each term in calculating the content-based similarity. To evaluate our method, we used three months worth of query logs from SINA's search engine to do off-line query mining. Based on the evaluations of independent editors on a query test set, our method was found to be effective for finding related queries, despite its simplicity. In addition to the subjective editors' rating, we also performed tests based on actual anonymous user search sessions. We organize the paper as follows: In Section II, we review related work, then in section III, we introduce the query log and session format. In section IV, we introduce our method for detecting related queries. In section V, we discuss our implementation details and experimental results. Finally, in section VI, we make our conclusions.

II. Related Work

The study of query logs is not a new topic. In [6] and [13], the query logs of *Excite* and *Altavista* were studied and analyzed, respectively, and some search patterns were reported. The analysis of the user sessions and query correlations encouraged further exploitation of these statistics. Beeferman and Berger [2] use a "content-ignorant" approach and a graph-based iterative clustering method to cluster both the URLs and queries, based on the users' query and click-through data. Clustering query logs can also help provide query recommendations. In [1], Baeza-Yates et al. presented methods to get query recommendation by utilizing the click-through data. In [3], Fonseca et al. used association rules for generating related queries. The idea of mining association rules to discover search engine related queries is interesting, and in a way is the most similar to our approach, though our approach is much simpler and takes into account the *order* of the queries in a session. Chen et al. [10] used fuzzy rules to adapt user queries in IR. In this work, fuzzy rules were extracted from the fuzzy clusters, describing a group of textual documents of interest to a user, and discovered by the Fuzzy C-Means clustering algorithm [12]. Since these rules can be used to characterize the semantic connections between keywords in a set of documents, they can be used to improve the user queries for better retrieval performance. The work presented in this paper differs from this approach since no clustering is used, and the rules are not based on the text documents. They are instead only based on the query stream itself and the notion of query sessions. This makes our approach fast and able to adapt continuously and dynamically to changing query inputs. Tajima and Takagi [14] propose a search engine which conceptually matches input keywords and text data. The conceptual matching is realized by context-dependent keyword expansion using conceptual fuzzy sets, that are realized using Hopfield Networks. The proposed technique is more inline with a search engine which can execute conceptual matching dealing with context-dependent word ambiguity, and the conceptual fuzzy sets are extracted from external sources that may have to be manually entered. In contrast, the method proposed in this paper tends to *automatically* learn fuzzy associative memories at the "query" level and does so directly from the query log. Miyata et al. [11] propose a query expansion method based on fuzzy abductive inference. However this system relies on a rigid representation of domain knowledge that is similar to WordNET for all the assumed concepts. In the work proposed in this paper, we make no such assumption about any pre-assumed concepts or domain knowledge. Instead, all of the fuzzy associative memory correlations are learned directly from the data (*the query log*), which means that the proposed approach can adapt to any search engine *regardless, for example, of language.*

III. Query Logs and Query Sessions

SINA is one of largest Internet Portals in China and has more than *one hundred million registered users*, and many *more* unregistered users. It provides online news and content services, mobile value-added services, community-based services, games, and Web search directory services, etc. We used the query logs generated from its web searching services (About 30G query logs in total for our evaluation). The query log file has the format shown in table 1. Each request contains a *CookieID* identifying the user, an IP address of the client machine, a time stamp, and the query. In case we can't get the cookieID, we use the user's IP address combined with a timeout for identifying one user. In our query session definition, in order to get as much information from a single user as we can, we define the query session as a sequence of queries input by one specific search engine user as long as the time interval between two consecutive queries falls within a threshold *t:* QuerySession = (query1, query2, query3,...), where each query is a set of keywords.

IV. Detecting Related Queries From Query Sessions

After extracting the query sessions, we will make an assumption that all the queries in the same query session bear some conceptual similarities. This is the same assumption made by methods that use association rules [3]. However, association rules cannot distinguish between the strength of association of two queries that have been submitted immediately one after the other in the same session, and two queries that may be separated by more other queries in between. Below, we first illustrate how we can model this association using an intuitive graph model. Then we explain how a theoretical and computational inference method can be achieved through the use of a collection of Fuzzy Associative Memories, each one modeling one of the historic query sessions. Note that as will be explained in section V-A, in the Indexing phase all rules are incrementally accumulated and indexed to form a summarized (cumulative over all sessions) set that is quickly accessible.

A. One-dimensional Graph-Model and Fuzzy Associative Memory based Modeling of the Consecutive Query Relations

As illustrated in Figure 1, suppose we have three consecutive queries (after pruning identical ones) in the same query session, the vertices in the graph represent the individual queries, while the arcs between them represent the rules or fuzzy relations that are generated by this session. We can

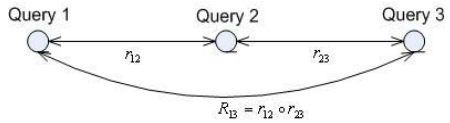

Fig. 1. Query graph and rules generated by one session

model the entire recommendation process, as shown in Fig 2, by the application of a Fuzzy Associative Memory inference system [9] that consists of the additive combination of a collection of rules that map a user's query to a recommended query. Rules are derived directly from fuzzy relation matrices which are composed together to form chained rules linking queries that appear together in the same session. The fuzzy relation matrices and their composition are defined using the same definition and notation in [8]. First, we define our universe of discourse as the set of all queries input by the users of the search system so far (i.e., the set of all distinct queries in the query log). Then we consider each query as a fuzzy set [17] defined over this domain of discourse. While a single user input query obviously defines a crisp singleton set with membership of 1 only for the matching query and zero for all other queries, the output of the query recommendation is a fuzzy set over the domain of discourse, with varying (possibly non-crisp) degrees of fuzzy memberships for each distinct query. These fuzzy memberships play the role of a *recommendation score*, which relative degrees will matter only for the purpose of *ranking* the final query recommendations, so that the top K recommendations are displayed to the user. Each rule/relation mapping one query to another query, is derived from one session by chaining several individual rules/relations that each map one of the historic queries to the query immediately following it in the same user session. Hence each new user session, say session Number s consisting of N_q consecutive queries, dynamically creates its own set of chained rules (call this a *macro-rule:* $R^s_{k,j} : q^s_k \Rightarrow q^s_j$), that associates the k^{th} query, q^s_k, in this session to any query q^s_j issued after the input query during the same session, ranging from query q^s_{k+1}, until the last query, $q^s_{N_q}$, of the session and each individual rule (call this *micro-rule:* $r_i : q^s_i \Rightarrow q^s_{i+1}$) within a session-chain maps a query to the query that comes immediately following it in the same session (i.e. its direct neighbor in the graph of Figure 1). Furthermore, when a new user query is submitted, it can activate any of the macro-rules if it matches the input of any of its micro-rules at any position within the chain sequence. Suppose that a given session from the query log contains the sub-sequence of queries $q^s_i, q^s_{i+1}, \cdots, q^s_j$. Then this will generate several micro-rules $r_k : q^s_k \Rightarrow q^s_{k+1}$ for $k = i, i+1, \cdots, j-1$, which are chained to form the macro-rules $R^s_{k,j} : q^s_k \Rightarrow q^s_j$. We chose to assign a constant strength ($\mu_{r_i}(q_i, q_{i+1}) = d$) to each of the micro-rules or relations r_i, and to further compose these micro-rules into a single macro-rule $R^s_{k,j} = r_k \circ r_{k+1} \circ \cdots \circ r_j$ with membership level $\mu_{R^s_{k,j}}$ obtained by a fuzzy composition (max-product correlation was used) of the individual relations r_i. Since individual relations map singleton queries, the composition will reduce to a multiplicative model as follows,

$$\mu_{R^s_{k,j}}(q_k, q_j) = \prod_{i=k}^{j-1} \mu_{r_i}(q_i, q_{i+1}) = \prod_{i=k}^{j-1} d = d^{(j-k)}$$

When the user submits a new query, q_k, the inference procedure applies all the macro-rules $R^s_{k,j}$, derived from all sessions s in the query logs, to derive the query recommendations, q_j, and associated scores, $\mu^k(q_j)$, as follows:

$$\mu^k(q_j) = \vee_{s=1}^{N_s}(R^s_{k,j} \wedge q_k)$$

where $\vee$ and $\wedge$ are suitable t-conorm/union (bounded sum in this case) and t-norm/intersection (product in this case) respectively. Since the recommendation scores $\mu^k(q_j)$ serve only for the purpose of ranking the recommended queries to the user, only their relative values are important (see Fig.3) and we simply use a regular sum instead of a bounded sum, as follows,

$$\mu^k(q_j) = \sum_{s=1}^{N_s} \mu_{R^s_{k,j}}(q_k, q_j) \cdot q_k \tag{1}$$

Eq. (1) is expected to capture the relation between queries submitted during the same search session. As will be illustrated below, our experiments confirm these intuitive expectations. For example, when we input "*River Flows Like Blood*", a novel written by a Chinese writer "*Haiyan*", we obtained not only the *writer* himself as its top-ranked related queries, but also obtained his *other popular novels* in the top-ranked recommendations.

B. Content Based Similarity Calculation

For the content-based similarity calculation, we use the cosine similarity.

$$Cosine(p, q) = \frac{\sum_{i=1}^{k}(cw_i(p)) \times (cw_i(q))}{\sqrt{\sum_{i=1}^{m} w_i^2(p)} \times \sqrt{\sum_{i=1}^{n} w_i^2(q)}} \tag{2}$$

where $cw_i(p)$ and $cw_i(q)$ are the weights of the i^th common keyword in the query p, and q respectively and $w_i(p)$ and $w_i(q)$ are the weights of the i^th keyword in the query p and q respectively. For weighting the query terms, instead of using TF*IDF, we use SF*IDF, which is the search frequency multiplied by the inverse document frequency. The reason why we use SF instead of TF is, considering the sparsity of the query terms in one query, TF in one query makes almost no difference. While using SF in its position is a kind of

voting mechanism that relies on most people's interests and judgements. SF is acquired from all search queries, while IDF is acquired from the whole document set.

C. Combination of the Consecutive Query Relation and Content-based Similarity

Using the above two methods, we can get two different types of query clusters. The two methods have their own advantages and they are complementary to each other. When using the first (consecutive relation-based) method, we can get clusters that reflect the users' consecutive search behavior. While when using the second, (content based) method, we can group together queries that have similar textual composition. To quickly combine them together, we used the merge-sort algorithm. For one specific query, we will get two clusters of related queries, which are sorted according to their cumulative relation levels. We then merge the consecutive based recommendations with the content based similarity together to produce the recommendation score

$$Rec(q_j) = \alpha \times \mu^k(q_j) + \beta \times Cosine(q_k, q_j) \qquad (3)$$

where α and β are the coefficients or weights assigned to consecutive-query-based relations and content-based relations respectively. In our experiments, we give them the same value. Fig. 3 illustrates the entire process.

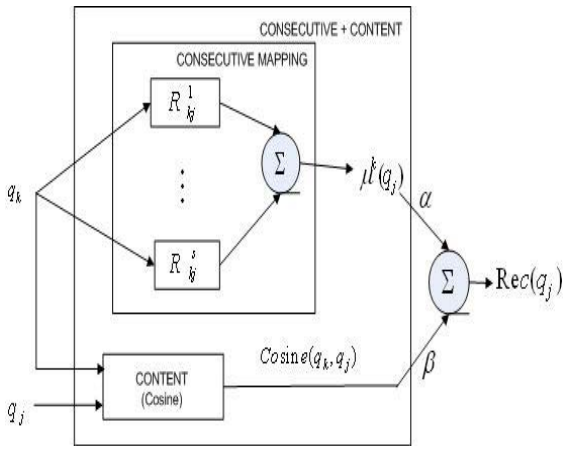

Fig. 2. Combining Consecutive Query Inference with Fuzzy Associative Memories and Content-based similarity to get each Recommendation

V. Implementation and Experimental Results

In our implementation, outlined in Figure 4, we follow the steps discussed below.

A. Six Steps for generating related queries

1) Step 1. Extracting Query Sessions: We isolate through *cookieId* the user sessions. According to our statistics, most people will accept cookies. So from almost every query log

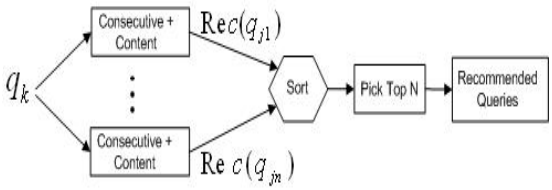

Fig. 3. Process Diagram showing The Combination of Query Recommendation with input query q_k

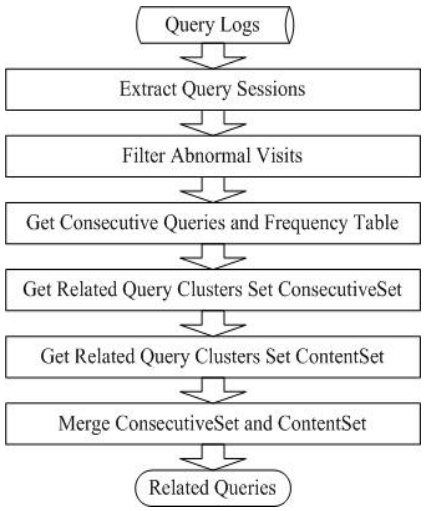

Fig. 4. Steps for generating related queries

entry, we could get the *cookieID*. Also in this step , we remove queries which are too long, blank or illegal. By *illegal*, we mean some queries that can't be decoded into normal Big5, GB2312 Chinese or English queries.

2) Step 2. Filtering robot and abnormal visits: Most commercial search engine servers receive many daily abnormal and robot/crawler visits. We need to filter these queries in the data preparation phase in order to get a cleaner data sample for further analysis. To recognize such abnormal visits, we rely on the following three heuristics.

i) *One user with too many sessions*: After sampling from our user track log, we found that if the number of queries exceeds 30 for the same user, then these visits may be deemed abnormal.

ii) *One user with too many repeated queries*: If user's queries are more than two times the total distinct queries, we identify the user's visit as abnormal and prune such sessions.

iii) *One user with too many close visits*: If a user's close

visits are too numerous in one day, we also consider that user's visits as abnormal.

3) Step 3. Obtaining the consecutive queries and frequency table: After filtering abnormal visits, we begin to get what we defined as *consecutive visits* (each visit generates a query). For consecutive visits, we define a valid consecutive interval, such as between t_1 and t_2. If the time between two visits falls outside this valid consecutive interval, that is, if the interval is too short or too long, we will no longer consider them as consecutive visits. We set the t_1 lower limit here again to avoid some abnormal visits, since a normal visit may require some time at least to explore the returned pages. A series of consecutive visits by the same user within the valid time interval form *one user session*. Notice that one specific user may have several sessions in a day, each one possibly targeting a different topic. Since these query sessions are temporally divided, we do not consider them to be in the same session although they may be generated from the same user.

Moreover, in order to avoid having too many consecutive visits in one session, and for processing simplicity, we set an upper limit *U* for storing the number of queries in one session.

At the same time, we use our cleaned query sessions to record the search frequency of each distinct query and sort them in descending order according to the queries' visit frequency. This frequency table is later used for generating content based related query clusters as in [1].

*4) Step 4. Obtaining the consecutive-based related query clusters set (*ConsecutiveSet*):*

Indexing Related Queries: After logging the consecutive queries in the users' search sessions, we begin to extract related queries offline. For each related query pair, we calculate the fuzzy membership $\mu^k(q_j)$ using (1), and store it in a related query table that maps each pair of queries q_i and q_j to their membership $\mu^k(q_j)$. This calculation is performed in an incremental manner as new sessions are logged, by accumulating the membership $\mu^k(q_j)$ whenever the related queries occur in a new session, and storing them in an *SQL* database, indexed by the key values k and j. To capture the symmetry of the related queries, the entries for (k, j) are identical to (j, k).

Adapting to Evolving Related Queries: In order to adapt to new queries, while at the same time gradually forgetting old queries, (and in the process also avoiding a potential overflow), we use a limited active sliding window, for storing and processing the query logs, that spans an active period such as 100 days, where we keep both the fuzzy memberships $\mu^k(q_j)$ that reflects the *overall cumulative* query logs during the most recent active period, as well as the fuzzy memberships $\mu^k(q_j)$ that reflects the sub-cumulative query logs received in *each day* during the past period. Every time that a related query pair falls outside of the active window period, such as more than 100 days ago, we deduct its single-day contribution to the membership value from the total accumulated value. At the same time, the contributions of the most recent/current queries keep getting accumulated as usual. In this way, the cumulative version of the fuzzy membership $\mu^k(q_j)$ will always reflect the related queries according to the most recent period, therefore evolving to the dynamic nature of user searches.

Filtering Top Queries: In practice, we have also found that exploiting the consecutive and content based relations between queries is not sufficient because of the exceptional spamming effect of extremely popular queries. For example, the query "Film download" may be the top (most frequent) search query regardless of the topic being searched, and it may spam the query relation memberships accumulated. If too many people submit query B after searching with query A, we may either consider query B as A's related query, or consider query B as a very popular query, (from now on referred to as *top queries*). Therefore, before extracting the related queries, we prune all the top queries for the given day to reduce their spamming effect. For this purpose, we get everyday's top *N* queries, and remove their related pairs from the cumulative memberships.

*5) Step 5. Obtaining the content-based related query clusters' set (*ContentSet*):* For content-based similarity calculation, if we compute the similarity for every two pairs (without repetition) in the frequency table, the complexity will be $O(n^2))$, which entails a high computational cost. In order to reduce this cost, we made the following simplification and adjustment. We set a visit frequency threshold *V* for processing a specific day. After we collect the visit frequency of all queries in a day's session, we only calculate the similarity of the queries whose visit frequency is higher than *V*.

6) Step 6. Merging the Clusters in ConsecutiveSet and ContentSet: In step four and five, we get two different cluster sets, which are named *ConsecutiveSet* and *ContentSet*. To get the final related query set, we need to merge them by using the formula developed in section 5.4.

B. Offline mining and Online Recommendation

The above process is conducted offline. After getting the final related query table, we use some further optimization, which includes pruning pornography or objectionable queries, to generate the applicable data set. Note that we do this pruning in order to avoid giving unintended offensive recommendations to regular users and not as in a censoring measure. We extract the related queries that have sufficiently high similarity value or only recommend the top *K* related queries to the user. The real-time recommendation part is implemented online. The online recommendations are rather efficient, since we only need to use a hash table to find the related queries and then present them to the user.

C. Experimental Results

In the following, we present our experimental results by processing about 3 months worth of query logs from SINA' search engine. Table 3 lists a sample of our related search keyword examples, where we have translated the Chinese queries into English.

For more examples, please visit *iask.com*, where the query recommendations are displayed in the bottom of the main frame.

Table 3. Recommendation Sample

QUERY	RECOMMENDATIONS
great wall	great wall computer;great wall broad-band;great wall motors; great wall pictures;ten-thousand-mile great wall;southern-great-wall cup;great wall securities;great wall cooperations;great wall lubricant;Badaling great wall
lilac	lilac download;lilac lyrics;lilac FLASH;lilac guitar music; story of lilac;lilac MP3;Tanglei lilac;lilac songs;lilac blossom; ≪lilac≫
jeans	jeans beauty;jeans picture;beauty jeans;tight jeans;jeans brands; LEVIS jeans;tight jeans beauty pictures;tight jeans beauty; LEE jeans;jeans beauty pictures
Fragrance Hill	Fragrance Hill park;Fragrance Hill red autumnal leaves; Fragrance Hill Hotel;Beijing Fragrance Hill;Fragrance Hill Plant arboretum;Honolulu;Fragrance Hill maps;Beijing Fragrance Hill Park;Fragrance Hill weightloss;Fragrance Hill sage
shopping guide	LIFE STYLE;Beijing shopping guide;Shanghai shopping guide;LIFE STYLE newspaper;online shopping; OLAY shopping guide;Hongkong shopping guide;shopping; home appliances;air-conditioners
super market	super market management;carrefour supermarket; supermarketers;wal-mart supermarket;supermarket promotion plans;supermarket good shelves;Quanzhou information supermarket;Fresh Flowers supermarket;supermarket management policies;Watsons supermarket

Below, we discuss how we evaluated the query recommendation results. We use the traditional precision and coverage method in information retrieval. Moreover, we used two evaluation methods to obtain both quantitative and qualitative validation.

*1) **Anonymous Session Coverage Evaluation Method And Results**:* Generally, coverage is defined as **the percentage of correct recommendations relative to the true query set**. Suppose we have a new user session which is not processed in our training phase and have the following queries:

NewSession = query1, query2, query3, query4, query5, query6.

After the user inputs query1, if all the remaining queries are given by our recommendation system as related queries, we consider the coverage to be 100%. And if we only recommend query2 and query3, then we consider that the coverage is only 40%, etc. Note that there is no reasonable or intuitive basis to evaluate precision in this context, because of the nature of the query recommendation task. In other words, if spurious queries are part of the recommendations, then this will not bother the user unless all of them are spurious. This is because the user will benefit even if a single query is promising, and will likely not be overloaded, since we only present up to 10 recommendations, (that usually span a modest space of only 1 to 3 lines). For this evaluation, we randomly chose 50 testing sessions, each of which containing two queries or more. See Table 6 for some real session samples. These sessions were extracted from two days' query logs from April 2005, so they have no overlap with the query logs used in the training set. The latter contained the query logs for September, October, and November. We did our test based on a maximum of $k = 10$ recommendations per query. We then counted the number of recommendations that matched the hidden queries of one session. Table 4 shows the testing results.

Table 4. Anonymous Session Coverage Result

session length	num of sessions	n_1	n_2	average coverage
2-query session	147	35	29	21.8%
3-query session	37	23	19	28.4%
4-query session	14	5	9	16.7%
6-query session	2	2	0	10%
total average coverage(200 sessions/473 queries)				22.3%

In Table 4, n_1 is the number of remaining queries in the session that fall into our recommendation set for the *first* query in the same session; while n_2 is the number of remaining queries in the session that fall into our recommendation set for the *second* query in the same session. From Table 4, we can see that the coverage is not very high(but this is typical of all query recommendation systems). If we increase the maximum number of the recommendations, to say 50 top ranked recommendations, this coverage would increase. However, this would not be practical. In just the same way that a search engine user would only have patience to view the first one or two pages of the search results, he/she would only have enough patience to scan a few top ranked query recommendations.

These considerations cause us to speculate that the evaluation measurements suffer from the following limitations. First, as discussed above, depending on the number of query recommendations, the precision and coverage may vary. Second, these query recommendations are not exactly "personalized" query recommendations, but rather "collaborative" recommendations. That is, our methods take into account the "cumulative" effects of a community of searchers, and there is no personalization for one specific user. For all users, we recommend the same recommendation set as long as they input the same query. Besides, in the above example, it may not be fair to consider the coverage for the 6-query session to be 40% just because the query recommendation set for the first query contains only query2 and query3, since if the user accepted our previous recommendations (query2 or query3), then the query recommendation set for query 2 and query3 may contain query4, query5, or query6. In this case, the *effective* coverage would be perceived to be higher than the one that we computed in our experiments. We need to improve our methods to evaluate testing in a real search scenario by taking into account the recommendations during the entire span of a search session. Measuring the practical users' clickthrough rate of the related queries would be more objective, but it also suffers from the same problem of not accounting for the accumulating effects.

*2) **Editors Rating Evaluation Method And Results**:* Search engine users' subjective perceptions may be more likely to indicate the success or failure of a search engine. For this reason, we also used editors' ratings to evaluate our query recommendations. In this method, we count our editors' likelihood of clicking on the related queries. We scale our editor's likelihood of clicking one of the query recommendations from 0 to 5. If all three editors rate their likelihood of clicking one of the query recommendations as 5, then this would indicate that during practical search circumstances, all of them would click one of the query recommendations, and we can thus consider our recommendations to have high coverage.

We should note that, the quality of the query recommendation is sometimes subjective and dependent on one user's domain knowledge of a specific field. Some closely-related queries may seem totally-unrelated for other users who have no such knowledge. This is the reason why we chose three independent editors.

The precision is also listed in the right column of Table 5.

For precision, we also rely on our editors judgement. We also scale the editors' comments from 0 to 5. When one editor rates a query's recommendations as 5, we consider that all the 10 query recommendations are closely related to the original query, while 0 means that none of the recommendations are related to the original query. To evaluate our method, we tested one hundred queries. These queries were selected by our editors according to several criteria. These criteria include that they have high visit frequency and that they cover as many different fields as possible. Using these queries, we generate the queries' recommendations. For each query, we ask our editors to rank the recommended query results from 5 to 0, where 5 means very good and 0 means bad. For precision, we ask them to review whether the related queries are really related to the original query. For coverage, we ask them whether they would click on the query recommendations in a real search scenario. Table 4 lists the average rating results from these evaluations.

Table 5. Editor Evaluation Result

Ranking	coverage(among 100)	precision(among 100)
very good(5)	35	29
good(4)	33	42
marginally good(3)	13	11
bad(2)	9	8
very bad(1)	7	7
nothing(0)	3	3
average rating	3.71	3.69
standard deviation	1.37	1.32

In Table 5, the numbers mark how many query recommendations are ranked in the specific rank among the one hundred queries. For example, in the second column, which is for coverage, 35 queries were rated as very good, 33 were rated as good, 13 were rated as marginally good, etc. And in the third column, which is for precision, 29 were rated by the editors as very good, 41 were rated as good, etc. From Table 5, we can see that among our editors' evaluations on coverage and precision, about 70 percent are very good or good.

VI. Conclusions and Future Work

We presented a simple and intuitive solution that has already been implemented with success as part of an *industrial-strength* search engine on one of the largest portals (*www.sina.com*) in China. We further combine the *Consecutiveness Relation* with the *textual similarity* to extract relations between submitted queries in order to ***(i)*** improve coverage, and ***(ii)*** compensate for the short nature of the majority of search engine queries, which results in *sparse* data. Our association-type information between queries are computed in a simple and efficient incremental manner from the query logs, based on a one dimensional graph model, where nodes represent the queries, and edge weights represent a *soft* measure of *consecutiveness* of the queries. This means that queries that are submitted immediately one after the other contribute with the maximal membership in the cumulative query-pair relation matrix, while this membership is gradually dampened as the queries move farther apart from each other during the same search session. *Dampening* the weights as queries become more distant in a session is not only a simple and intuitive way to capture query *relatedness*, but it also acts as a filter that helps decrease the influence of the imperfect session segmentation performed in the pre-processing phase (though this was based on relatively reliable cookies and timeout mechanisms), and thus alleviate problems that can result when even the *same* user *shifts their attention* during the *same* session. Moreover, these weights are accumulated incrementally with each new session to obtain a *global, adaptive and scalable* model. Furthermore, our method adapts to the dynamic searching behavior of the users. We used the continuously updated query graph to form a *Fuzzy Associative Memory* [9] to combine all the uncertain associations between the queries in the historical query stream.

Our work benefits from *combining* both the query *content* similarity with the search session *query-pair relations together* to generate well-rounded query recommendations. More work needs to be done to eliminate, the crawlers' influence on web query log mining, and to more accurately model the *evolving* nature of users' queries.

VII. Acknowledgments

This work is partially supported by a National Science Foundation CAREER Award IIS-0133948 to Olfa Nasraoui.

References

[1] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *International Workshop on Clustering Information over the Web (ClustWeb, in conjunction with EDBT), Creete, Greece, March (to apper in LNCS).*, 2004.

[2] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. *Proceedings of ACM SIGKDD International Conference, pages 407–415.*, 2000.

[3] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani. Using association rules to discover search engines related queries. In *LA-WEB: 66-71.*, 2003.

[4] H.Cui, J.Wen, J.Nie, and M. W. Probabilistic query expansion using query logs. *WWW2002, May 7-11, Honolulu, Hawaii.*, 2002.

[5] H.Cui, J.Wen, J.Nie, and W.Ma. Query expansion by mining user logs. *IEEE Transaction on Knowledge and Data Engineering, Vol. 15, No. 4, July/August*, 2003.

[6] B. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval:a study of user queries on the web. *SIGIR Forum, Vol.32. No.1.pp. 5-17*, 1998.

[7] T. Joachims. Optimizing search engines using clickthrough data. *Proceedings of Knowledge Discovery in Databases.*, 2002.

[8] G. Klir and T. Folger. *Fuzy Sets, Uncertainty, and Information*. Prentice Hall, New Jersey, 1988.

[9] B. Kosko. *Neural Networks and Fuzzy Systems - A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, New Jersey, 1992.

[10] D. Kraft, J. Chen, M. Martin-Bautista, and M. Vila. Textual information retrieval with user profiles using fuzzy clustering and inferencing. In P. Szczepaniak, J. Segovia, J. Kacprzyk, and L. Zadeh, editors, *Intelligent Exploration of the Web*. Physica-Verlag, 2002.

[11] Y. Miyata, T. Furuhashi, and Y. Uchikawa. Query expansion using fuzzy abductive inference for creative thinking support system. *Trans. IEE of Japan*, 119-C:368–372, 1999.

[12] E. H. Ruspini. A new approach to clustering. *Information and Control*, 15-1:22–32, 1969.

[13] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum, Vol.33. No.3*, 1999.

[14] M. Tajima and T. Takagi. Query expansion using conceptual fuzzy sets for search engine. In *IEEE Fuzzy Sustems Conf*, pages 1303–1308, 2001.

[15] J. Wen, J. Nie, and H. Zhang. Clustering user queries of a search engine. *WWW10, May 1-5, 2001, Hong Kong*.

[16] J. Wen, J. Nie, and H. Zhang. Query clustering using user logs. *ACM Transactions on Information Systems, 20(1), pages 59 – 81.*, 2002.

[17] L. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.