Finding Similar Queries to Satisfy Searches based on Query Traces

Osmar R. Zaïane[1] Alexander Strilets[1]

[1] University of Alberta, Edmonton, Alberta, Canada T6G 2E8
{zaiane, astrilet}@cs.ualberta.ca

Abstract. Many agree that the relevancy of current search engine results needs significant improvement. On the other hand, it is also true that finding the appropriate query for the best search engine result is not a trivial task. Often, users try different queries until they are satisfied with the results. This paper presents a method for building a system for automatically suggesting similar queries when results for a query are not satisfactory. Assuming that every search query can be expressed differently and that other users with similar information needs could have already expressed it better, the system makes use of collaborative knowledge from different search engine users to recommend new ways of expressing the same information need. The approach is based on the notion of quasi-similarity between queries since full similarity with an unsatisfactory query would lead to disappointment. We present a model for search engine queries and a variety of quasi-similarity measures to retrieve relevant queries.

1 Introduction

Many commercial search engines boast the fact that they have already indexed hundreds of millions of web pages. While this achievement is surely remarkable, the large indexes without doubt compromise the precision of the search engines adding to the frustration of the common users. Many agree that the relevancy of current search engine results needs significant improvement. Search engines present users with an ordered and very long list of websites that are presumably relevant to the query specified based on criteria specific to each different search engine. Users typically consult the first ten, twenty or maybe thirty results returned and give up if relevant documents are not found among them. Results are normally ranked by relevance, which is calculated based mainly on the terms present in the query and not necessarily on the semantics or meaning of the query. Some Search engines like Google [2] use the notion of incoming and outgoing hyperlinks from documents containing the query terms to rank the relevant URLs [1]. It remains, however, that these lists are too long to browse. While users are at the mercy of the result ranking procedure of the search engine, they are also constrained by the expressiveness of the query interface, and often have difficulty articulating the precise query that

could lead to satisfactory results. It happens also that users may not exactly know what they are searching for and thus don't know how to effectively express it and end up selecting terms for the search engine query with a trial and error process. Indeed, often users try different queries until they are satisfied with the results. If the results are not satisfactory, they modify the query string and repeat the search process again. Many commercial search engines provide possibilities to narrow searches by either searching within search results or augmenting a query to help users narrow their search [8]. However, these query augmentations just append terms to the existing terms in the query. For example a search on AltaVista with the term "avocado" leads to the following suggestions: avocado trees, growing avocados, avocado recipes, avocado oil, avocado plant, etc. This is comparable to our first quasi-similarity measure presented below. We call this method the "Naïve approach" because it simply looks in the query trace for queries having terms similar to the terms in the current query. In other words, it considers only the terms in the queries and simply performs intersections (see below). This example, however, shows that it is conceivable that more than one user would send a search query for a similar need and it is possible that these queries are differently expressed. A study by Markatos shows that many queries sent to a search engine can be answered directly from cache because queries are often repeated (by presumably different users). The study reports that 20 to 30% of queries in an Excite query trace with 930 thousand queries, were repeated queries [6]. A similar study using AltaVista query logs demonstrated that each query was submitted on average four times. This average is for identical queries not taking into account upper/lower case, word permutations, etc. This justifies the assumption that when one user submits a query to a search engine, it is highly likely that another user already submitted a very similar query. The queries can be identical as found and reported by the studies mentioned above, or articulated differently with different terms but for the same information needs. This is the major argument to put forward the idea of using query collective memory to assist individual users in articulating their information needs differently by suggesting quasi-similar queries. We define quasi-similarity in the next section.

The idea of tapping into the collective knowledge of users, embodied as a set of search queries, is not new. Fitzpatrick et al. studied the effect of using past queries to improve automatic query expansions in the TREC (Text REtrieval Conference) environment [4]. The idea is that top documents returned by the query from a pool of documents are also top documents returned by similar queries and are good source for automatic query expansions. They compared the performance of this method against the unexpanded baseline queries and against the baseline queries expanded with top-document feedback. The authors present a query similarity metric that empirically derives a probability of relevance. They also introduce the notion of threshold to control on per query basis whether or not a query should be expanded. In a similar study, Glance describes the community search assistant, a software agent that recommends queries similar to the user query [5]. The similarity between queries is measured using the number of common URLs returned from submitting both queries to the search engine. The main contribution is the notion of

collaborative search using query traces in the web search engine context. However, if two queries have the same search results and the user is not satisfied with the result of one of them, the results of the second ought to be unsatisfactory. Thus, the suggested similar query is inadequate; hence, the notion of quasi-similarity of queries presented in the next section.
A query trace is basically a log containing previously submitted queries. This log is not enough to compute similarities between queries. In Section 2 we introduce Query Memory, a data structure that holds not only the collective query trace but also extra information pertaining to the queries that would help in measuring similarities between queries. We introduce our query quasi-similarity measures using the Query Memory in Section 3. In Section 4 we depict our prototypical implementation. Some examples are discussed in Section 5. Finally, Section 6 presents our conclusions.

2 Query Memory Data Structure

We have collected a large Query Memory from a popular meta-crawler and saved these queries locally in our database. A query in our view is not just a string, but a bag of words and associated to it is the list of documents that are returned by different query engines (via a meta-search-engine). Each document consists of a URL, a document title and a snippet (short text returned by the search engine). Each title and snippet is considered as a bag of words as well.
A Query Memory is a set of queries where each query is modeled as follows:
1- BagTerms: unordered set of terms (bag of words) from the query string;
2- Count: number of times the query was encountered in the query trace;
3- Ldate: last time encountered in the query trace;
4- Fdate: first time encountered in the query trace;
5- QResults: ordered list of URLs and titles returned when the query is submitted, in addition to the snippets (i.e. text that accompany URLs in the result page). The text is modeled as bags of words containing terms from the snippets and title as well as the words in the document path in the URL;
6- Rdate: date the Q*Results* was obtained. Notice that this is the date for the results and it is not necessarily related to *Fdate* and *Ldate*.

The words in *BagTerms* as well as the bag of words associated with the URLs in *QResults* are stemmed using Porter's algorithm [7] and filtered from stop-words. *QResults* elements are composed of: (1) *Rurl*: the URL of the result entry; (2) *Rtitle*: the title of the result entry; and (3) *Rsnippet*: bag of words from either the snippet accompanying the result entry or from the document itself pointed to by the URL.

Using the Query Memory model described above we propose different similarity measures that, given a user query, allow finding all other similar queries from our Query Memory. Notice that an exact similarity is not desired. If a user is unsatisfied

with search results and wants hints for queries, if these hinted queries are identical or give an identical result to the original query, the user would not be satisfied. Instead, we want to suggest queries that are close enough in terms of query, or queries that yield results that are comparable in content or description. We have tested different measures for quasi-similarity using either the terms in the query, the terms in the title or snippet of the search results, or the URL of the search results. Notice that we used the snippets returned by the search engines with the results to represent the document content instead of actually fetch the documents and retrieve the terms from their content. Fetching the documents would necessitate accessing the documents at their respective Web locations and parsing them, which would have added significant overhead. In theory, the real content of a document is the best representative of the document, but we noticed that using the snippets alone was sufficient and lead to acceptable results in a reasonable and practical time. Our model also keeps track of the timestamp when a query is first and last time encountered in the query trace. These timestamps are used in the ranking of similar queries, but they are also used to purge the Query Memory from old queries. It appears that some queries are cyclic (i.e. appearing and disappearing at different periods), or simply periodic (i.e. they appear very frequently but in a short time period, then never again). An illustrative example would be the queries: "terrorism", "El Quaeda" or "Afganistan". They appear after an major event, persist in popularity for some time then fade out when the general interest shifts somewhere else. This is also true with Olympics and other events. Currently, we did not implement the purging of old queries from the Query Memory. However, we use the timestamps for ranking results and introduce the notion of query aging.

3 Overview of different similarity measures

In this section we use the following notation: Q denotes the current query; Δ denotes the Query Memory database; Q.terms indicates the set of terms in the query; Q.Results is the set returned by a search with Q.Results[i] symbolizing the i^{th} entry in the list of results; Q.Results[i].*Title* and Q.Results[i].*Text* represent respectively the set of words extracted from the title and URL, and the set of words extracted from the snippet of the search result entry. We should point out that the notion of query results (Q.Results) is non-deterministic. In other words the same query submitted to the same search engine at a different time can, and usually will produce different results. This is because search engines constantly update their databases and indexes. Some search engines even rotate the order of returned results. For example submitting query "car" twice to the Metacrawler search engine will result in the following three top results respectively for the first and second submission: (1-"Factory Invoice Prices and Dealer Cost", 2-"Free New Car Quotes!", 3-"Car Buying Information") and (1-"Factory Invoice Prices and Dealer Cost", 2-"Shopping for a New Car? Try InvoiceDealers!", 3-"Classic Cars Photo Ads"). Top results retuned by the same search engine are quite different, even though the same query was submitted to the search engine within a few minutes of each other. In our Query Mem-

ory, we do not store all results returned by search engines, but only the first 200 results.

Using the notations described above we define the different quasi-similarity measures as follows:

1- Naïve query-based method:

$\forall \boldsymbol{q} \in \Delta \; / \; \boldsymbol{q}.\textbf{terms} \cap \boldsymbol{Q}.\textbf{terms} \neq \varnothing$

This is a method used by some existing search engines to recommend expending queries. It simply finds queries with common terms as the current query. There is no word semantics directly or indirectly involved. The results are ordered based on the values of *Count* and *Ldate* (see above). While it is very simplistic, it sometimes yields interesting results. For example "car parts" is found similar to "automotive parts" and "Honda parts". However, it wouldn't be able to find "salsa food" similar to "Mexican recopies". The next method, however, could.

2- Naïve simplified URL-based method:

$\forall \boldsymbol{q} \in \Delta \; / \; \boldsymbol{q}.\textbf{Results.URL} \cap \boldsymbol{Q}.\textbf{Results.URL} \neq \varnothing$

This is a method similar to the first one, except instead of looking for common words in the text of a query, we are looking for common URLs returned by the search engines. The reasoning is that if URLs returned by two queries at least have one URL in common, then these queries might be related. All the queries that are found to be similar are then sorted by the occurrence frequencies and are presented in that order. In other words the most common queries that have at least one common URL with the result of the user query are suggested to be similar. This particular method works well on a small dataset of collected queries but has a tendency of making bad suggestions more often than the next method.

3- Naïve URL-based method:

$$\forall \boldsymbol{q} \in \Delta \; / \; \theta_m < \frac{\left| q.\mathrm{Re}\, sults.URL \cap Q.\mathrm{Re}\, sults.URL \right|}{\left| Q.\mathrm{Re}\, sults.URL \right|} < \theta_M$$

This method considers the URL set in the search results of the queries. θ_m is a minimum threshold, while θ_M is a maximum threshold. We found that this method could yield interesting results depending upon the thresholds we set. If θ_M is too close to 100%, the queries become too similar. If θ_m is too small, completely different and irrelevant queries could be suggested. In the current prototype implementation we set θ_m to 0.2 and θ_M to 0.8. Method 3 is more specific and more flexible than method 2.

4- Query-Title-based method:

$\forall \boldsymbol{q} \in \Delta \; / \; \exists$ **i,** $\boldsymbol{q}.\textbf{Results[i].}\boldsymbol{Title} \cap \boldsymbol{Q}.\textbf{terms} \neq \varnothing$ **and** $\boldsymbol{q}.\textbf{Results[i]} \notin \boldsymbol{Q}.\textbf{Results}$

This method looks for queries that have in the titles (or URL path) of their results terms appearing in the original query. The idea is that queries that return results with titles related to the original query and the results were not retrieved by the original query could be indeed related to the original query.

5- Query-Content-based method:

$\forall q \in \Delta$ / $\exists$ **i,** ***q*.Results[i].*Text*** $\cap$ ***Q*.terms** $\neq \varnothing$ **and** ***q*.Results[i]** $\notin$ ***Q*.Results**
The idea is similar to the previous method except that the snippets of the results from the candidate queries are considered instead of the titles.

6- Common query title method:

$\forall q \in \Delta$ / $\exists$ **i,j,** ***q*.Results[i].*Title*** $\cap$ ***Q*.Result[j].*Title*** $\neq \varnothing$
The objective in this method is to compare the terms in the titles returned by the original query with the terms in the titles returned by the candidate queries. No comparison of the query terms per se is done. The main idea of this method is that if two queries return results such that title words in the results returned are similar (there are some common words), then queries are also similar.

7- Common query text method:

$\forall q \in \Delta$ / $\exists$ **i,j,** ***q*.Results[i].*Text*** $\cap$ ***Q*.Result[j].*Text*** $\neq \varnothing$
This method is similar to the previous one except that the snippets are compared instead of the titles. The last two methods are particularly good at finding queries that are syntactically different but semantically similar. Our experiments showed that on a small set of processed queries this method performs better than the previous one, however it is in an order of magnitude more computationally intensive than the previous one. Selecting the content of pages instead of the snippets returned by the search engines would make the computation is more time-consuming.

For all these methods, results are sorted. The ranking of the suggested queries is based on either the *Count* and *Fdate*, or the number of common URLs. Only a limited number of quasi-similar queries are presented to the user. Notice that we can easily combine these measures, in particular 3 and 4, or 5 and 6. The last two similarity measures are particularly computationally expensive and their time complexity is proportional to the size of the collected database. We are considering new indexes to reduce the time it takes to recommend queries with measure 6 or 7.

4 Prototype Implementation Overview

Figure 1 illustrates the general architecture of our prototype constructing the Query Memory. A query collector collects queries from a popular metacrawler and stores them in a query database with a timestamp. Query processors submit these queries to other search engines to collect results and store these results in the Query Memory with updated counts and timestamps. Different search engines are used to harvest search results and the original metacrawler is not used to submit queries in order to avoid affecting the query trace of the metacrawler, and thus avoid compromising the counts of the queries in our Query Memory by processing collected queries.

We have implemented our prototype on NetBSD, an operating system allowing large disk partitions, and MySQL database. The query database collected quickly reached more that 2 Gigabytes, the largest disk partition allowed by Linux. The

collection and processing of the queries is implemented with Python scripting language, while the similarity engine is in C^{++}. Figure 2 shows an overview of the query recommendation phase.

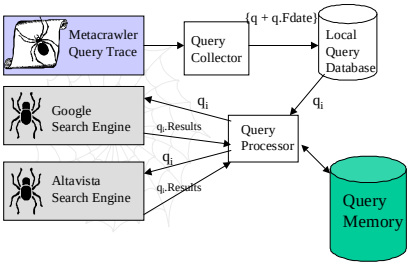

Fig. 1. Constructing the Query Memory

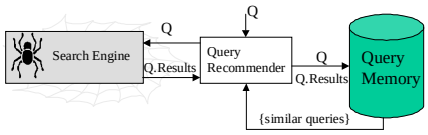

Fig. 2. Query quasi-similarity engine.

Our system consists of three major parts: a web query collector, a web query processor (Figure 1) and a quasi-similarity engine (Figure 2) that finds similar queries among previously collected queries to the one specified by the user. The query collector simply collects queries submitted to the Metacrawler search engine[1] and puts them in the query queue along with the timestamp when these queries were obtained. The Metacrawler search engine provides its query trace by displaying at regular times a list of currently submitted search queries[2]. The query collector agent is simply a Python script that at regular time intervals consults the Metacrawler query trace provided and parses it to extract the query strings. The query processor verifies for each query if it already exists in the Query Memory, as described above. If the query already exists in the Query Memory, then it just updates the relevant counters and timestamps. Otherwise, the query is submitted to the Google search engine and AltaVista search engine to harvest the corresponding search results. These results are then parsed, processed, and added in the Query Memory.

The quasi-similarity engine is triggered whenever a suggestion for queries is requested. The user specifies which similarity measure he or she wants to use to find similar queries. The similarity engine uses previously collected queries stored in the Query Memory to return suggested similar queries. We implemented a basic web-based interface that allows the submission of a web search query. The submitted query is sent to Metacrawler and the search results are displayed. If the user is unsatisfied with the results a recommendation for similar queries is requested and similar queries are displayed based on the similarity measure chosen. Because of lack of space we are not showing a snapshot of our prototypical user interface implemented. The entire web page of the prototype is divided into 4 frames. The top frame allows the user to specify a query, which is sent to the Metacrawler search engine. The

[1] MetaCrawler: http://www.metacrawler.com
[2] MetaSpy: http://www.metaspy.com/spymagic/Spy?filter=false

results returned are displayed integrally in the results frame (second from the bottom). Note that Metacrawler was arbitrarily selected for our prototype. Other search engines could be used or the choice could be given to the user. If the user is not satisfied with the results, he or she can request similar queries after selecting a quasi-similarity measure in the second frame. Suggested new queries are displayed in the bottom frame along with hyperlinks that would automatically submit the query if clicked, thus allowing an interactive process of query refinement.

5 Discussion on Some Experimental Results

This section presents some examples of similar queries using our measures of quasi-similarity for illustration purposes. We have implemented in our prototype the quasi-similarity measures 1, 2, 3, 6 and 7 as presented in Section 3. Using the interface described in Section 4, we submitted many queries and checked the suggested similar queries based on all implemented measures. While the similarity measures are different, it was common that the recommended queries with the different measures were alike. Moreover, it was difficult to find a winner (i.e. the similarity measure with the best recommendation) because: (1) for each different query we experimented with, a different winner could be proclaimed; (2) the search results and the recommendation are nondetermininstic (i.e. we can expect different results for the same query submitted twice); and (3) the validation is very subjective. The results of the query recommender, however, are very encouraging. Below we present two of our results randomly selected from the different experiments we performed: "salsa food" and "computer prices". Tables 1 and 2 do not contain all recommended queries but a top ranked selection. We found indeed queries such as "hot girls", latina porn", or "BBC News" as matches to the "salsa food" query for example. The match, however, is understandable. First these queries were submitted by someone, and thus were in the query trace. Second, there is a connection between "salsa", "hot" and "latina" indirectly discovered using our quasi-similarity measures by intersecting the search engine results. The "BBC News" match, on the other hand, is more difficult to explain. It is possible that at the time the "BBC News" query was processed, the results could have contained links to recipes, hence the intersection with salsa food. This also demonstrates the difficulty to use precision and recall as validating measures, given the subjectivity of the similarity.

Query:	"**salsa food**"
Measure 1:	Food Recipes, Japanese Food, genetically engineered food, food pictures
Measure 2:	recipes, Mexican Recipes, All Recipe.com, french cooking
Measure 3:	recipes, Mexican Recipes, All Recipe.com, french cooking
Measure 6:	junkfood, foods history, black pepper health benefit, Food Cooperative
Measure 7:	peruvian dry rubs, green curry recipes, black pepper health benefit, australian foods

Table 1. Recommended quasi-similar queries for the query "salsa food"

Query:	"**computer prices**"
Measure 1:	car prices, computer hardware, computer dictionary, airline ticket prices, computer parts
Measure 2:	computer memory, consumer price index, toms hardware, cheap computer upgrades, Hardware Stores
Measure 3:	price search, price compare, pricewatch.com, toms hardware, low cost computers, Wholesale Computer Pricewatch
Measure 6:	buying laptops, IBM Aptiva, online trading, ps one memory card, Shopping
Measure 7:	buying laptops, bid software, DVD Player & Review, christmas online shopping

Table 2. Recommended quasi-similar queries for the query "computer prices"

While measure 1 is very simplistic and matches only queries with terms in the original query, some of the results are nevertheless relevant to some extent. Measures 2 and 3, in the case of "salsa food", returned almost the same results; which is to be expected, since we do not have many queries processed in our Query Memory database yet. Therefore we set the threshold parameter θ_m to fairly a low level of 0.2. Measures 6 and 7 are looking for the common words in query titles or snippets and are capable of suggesting more "diverse" queries. The later measure, however, seems to make more relevant suggestions than the former one. This should not be a big surprise, since it looks for common words in snippets that are more descriptive than the title pages. On the down side, it is computationally very expensive.

There are two major observations from our experiments with our proposed quasi-similarity measures. First, the more queries we process, the richer the Query Memory is, and the better the recommendations are. While we have collected more than half a million different queries from the Metacrawler search engine, we have processed and harvested the results of only about 70,000 queries (which constitutes the Query Memory). Submitting queries to the search engine and harvesting result is very computationally and network intensive process. Ideally, if the query recommender is on the search engine side, not only the query trace would be immediately available, but also the inverted indexes of the search engine would also be available avoiding the submission of queries for results harvesting. Also because of the limitations of the MySQL database and the file systems file size problems, it would be better to directly process queries collected and populate the Query Memory rather than storing collected queries in a large queue. Only queries in the Query Memory can be used in the similarity measures. The second observation is that it seems difficult to select the best method. Apparently some combination of different methods with weights assigned might produce better results than every single similarity taken

alone. This is another open problem that we will be addressing in our future research.

6 Conclusion and Remarks

We have implemented a recommender system to suggest similar queries for search engine users when they are not satisfied with the result of an original query and would like help in expanding or improving the search query terms. We proposed some quasi-similarity measures to ensure that the suggested queries are not too similar to the unsatisfactory original query but analogous enough to be suitable. We have experimented with all these measures isolated or combined, and have noticed that it is very difficult to find an absolute winner. Each quasi-similarity measure could indeed generate very good results depending upon the query. In some cases what we expected to be the winner actually produced irrelevant queries. We have thus kept all the measures and left it to the user to select and experiment with the desired methods. It is important to notice that since the Query Memory is updated in real time by adding new queries, and since some queries could be removed if considered too old based on *Ldate*, the recommendation is never deterministic. In other words, the system, for the same original query, could suggest different similar queries at different times. In the current work we are also investigating the use of time constraints to limit the queries to those submitted in a time range. This is useful because queries could be time sensitive, appearing frequently at one point than disappearing completely later. We are also investigating the use of Latent Semantic Indexing [3] for finding relevant associations between titles or text returned in the search engine results. Finally, we are analysing the possibility to validate these quasi-similarity measures using precision and recall, and studying the effect of purging old queries from the Query Memory on the precision and recall.

References

1. Junghoo Cho, Hector Garcia-Molina and Lawrence Page, Efficient Crawling Through URL Ordering, 7th International Conference on WWW, Brisbane, Australia, 1998

2. Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, 7th International Conference on WWW, Brisbane, Australia, 1998

3. S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer and R. Harshman, Indexing by Latent Semantic Analysis, in Journal of the American Society of Information Science, 1990

4. Fitzpatrick, Larry and Dent, Mei 1997 Automatic Feedback Using Past Queries: Social Searching? In Proeedings of SIGIR'97, Philadelphia, PA.

5. Natalie S. Glance, Community search assistant, Workshop on Artificial Intelligence for Web Search, In conjunction with the AAAI conference, Austin, Texas, USA, 2000

6. Evangelos P. Markatos. On Caching Search Engine Results. In Technical Report 241, ICSFORTH, January 1999. available at: http://archvlsi.ics.forth.gr/html_pages/TR241/

7. M. F. Porter, An algorithm for suffix stripping, Program, pp 130-137, vol 14, n 3, 1980.
8. Danny Sullivan, Search Assistance Features: Related Searches, 2001 available at: http://www.searchenginewatch.com/facts/assistance.html