A New Query Suggestion Algorithm for Taxonomy-based Search Engines

Roberto Zanon[1], Simone Albertini[2], Moreno Carullo[1] and Ignazio Gallo[2]
[1]*7Pixel S.r.l., Binasco (MI), Italy*
[2]*Dipartimento di Scienze Teoriche e Applicate, University of Insubria, Varese, Italy*

Keywords: Query Suggestion, Query Log, Query Session, User Experience.

Abstract: The objective of this work is the realization of an algorithm to provide a query suggestion feature in order to support the search engine of a commercial web site. Starting from web server logs, our solution creates a model analyzing the queries submitted by the users. Given a submitted query, the system searches the most adequate queries to suggest. Our method implements an already known session based proposal enriching it by exploiting specific information available in the current context: the category the user is browsing on the web site and a solution to overcome the limits of a pure session based approach considering also similarity between queries. Quantitative and qualitative experiments show that the proposed model is suitable in terms of resources employed and user's satisfaction degree.

1 INTRODUCTION

Search engines have assumed increasing importance for the Internet users, becoming the main point of reference to access any other service or information on the web. The users expect that these research systems would provide increasing assistance and simplicity allowing the user to reach his goal. For this reason all the main search engines are going to introduce additional services in order to support the user such as, automatic correction of the query, the suggestion of related queries and the suggestion of related multimedia content.

Several research fields related to the study of the user behavior grew, starting from the query analysis (Mat-Hassan M., 2005), the classification of particular types of queries (Ortiz-Cordova A., 2012) to the definition of similarity and correlation relations, and so on. This is a difficult problem, because the queries contain unstructured data; thus, they often are very short or equivocal for a direct use. For example, in an online store we can exploit the navigation path followed by an user in order to infer its interests and to enhance its experience providing suggestions that can be used for presenting products the user may be interested in. With *query suggestion* we mean the task of proposing a set of different possible alternative search texts to a user who submitted a query, so that they could help him reaching what he is looking for.

The field of *Web Usage Mining* studies techniques for gathering information to profile the users, for example analyzing the web server log and the application level data for each user session (Srivastava and Cooley, 2000; Pierrakos et al., 2003). Such information allows to create algorithms able to predict the need and the desires of users just analyzing their behavior and trying to reduce that information to an already known behavior pattern.

The objective of this work is the realization of an algorithm for providing a query suggestion feature in order to support the search engine of a commercial web site (Shoppydoo, 2012). The system design is based on models already proposed by the literature (Boldi et al., 2008; Cao et al., 2008). From the literature we notice that there are two main approaches: *document-based* (Baeza-yates et al., 2004) and *session-based* (Boldi et al., 2008). The first approach exploits the URLs the user clicks after having subtimmed a query while the other is based on the consecutiveness of the queries submitted within each user session. The most studied and promising approach for developing a query suggestion system is the session based one (M.P. Kato, 2011). This approach states that when a user types a query q, then the system should suggest the queries that previous users submitted after having typed the same query q. This method is justified by the behavior of the users: when the first query they submit do not lead to the expected results, they tend to restate it following well known logical schemes like "reformulation for generalization", "reformulation for specialization" or "equivalent reformulation" (Boldi et al., 2009).

Zanon R., Albertini S., Carullo M. and Gallo I..
A New Query Suggestion Algorithm for Taxonomy-based Search Engines.
DOI: 10.5220/0004108001510156
In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval* (KDIR-2012), pages 151-156
ISBN: 978-989-8565-29-7

Another aspect a query suggestion system could follow is the exploitation of the context the user is actually browsing. So it is possible to classify the systems in *context aware*, like (Cao et al., 2008), and *non context aware*. We can also say an algorithm is *incremental* or *non incremental*, depending on whether it modifies its internal structure as it is given a new query or it has a fixed set-up and may be only queried (Broccolo et al., 2010).

2 PROPOSED METHOD

Each query suggestion algorithm is composed by two main modules: a background module which holds and manages the data structures and an online process which provides the suggestions to the front end application and makes use of the underlying data structures.

Starting from the web server logs, the proposed solution creates the first module in the following manner:

1. *Pre-processing*. It parses the web server logs in order to extract the queries and the related information.
2. *Creation of the sessions*. A set of sessions is built, that is a set of lists of queries performed by the same user within a certain amount of time.
3. *Construction of the model*. The algorithm builds the data structures for representing the sessions in order to efficiently mine all the information needed for the suggestions.
4. *Pruning of the data structures*. The algorithm must reduce the dimension of the data structures.

The second module of the algorithm inspects the data structures previously generated applying a ranking function, returning the suggested queries given a query q.

(Baeza-yates, 2007) presents an analysis of the processes that could be used to generate the data model which represents the queries and their relations. In the proposed work we followed an approach inspired by the *word graph* and *session graph*. The development took into account that the system must be generic in order to have the possibility to use it also in other websites but, on the other hand, as much specific as we can to exploit all the available information from the available query logs.

The proposed solution is inspired by (Boldi et al., 2008) but with some important differences. The first difference is that we are in a context where the objects to query are grouped into categories: the goal is to provide a query suggestion system for a website that lists commercial products sold by several merchants. So we have an implicit taxonomy and it is possible to exploit additional information considering the user is given the possibility to select a category to browse within and to submit queries for only that category. The category information is optional but it is a powerful tip for the system in order to select the suggested queries: it allows our algorithm to distinguish queries that would be syntactically indistinguishable. However, the concept of categories is also present in fields different from the price comparison: for example, a similar information is exploited by *Yahoo! directories*[1]. Another important difference is that, in addition to suggest the queries that belongs to the subtree which starts with the given query, it selects also the queries that follow the queries with the search string similar to the input search text. The similarity among strings is not usually considered sufficient because of the short length and ambiguity of the queries: anyway in this setting it is possible to adopt this approach because the majority of the queries is associated with a category known in advance, which gives a semantic contribution to the process.

Following the classification given by (Broccolo et al., 2010), we can consider our algorithm as an *incremental session-based non context-aware* approach.

2.1 Creation of the Sessions

The process starts analyzing the log files from the web server. The fields that are considered when filtering the log are the *query string* and the *category id*, extracted from the URL parameters; the *user id* if available or the IP address; the *timestamp*. Log entries different from searches or originated by bots are not considered.

A logical session is a sequence of queries with the same user id and where for each pair of queries, they were submitted not far more than thirty minutes.

Our system exploits the session information in order to create a query graph which models the consecutiveness relation of the queries with, in addition, another graph on the same set of queries which models the similarity among the search texts. An index is mainteined in order to provide fast access to the nodes of the graph.

2.2 Creation of the Query Graph and the Index

The query graph is similar to the *query flow graph*

[1] http://dir.yahoo.com/

Algorithm 1: Building of the query graph and the indexes.

Require: set of *sessions*
Ensure: query index I_q and category index I_c

```
I_q ← new HashTable {query index}
I_c ← new HashTable {category index}
for all session ∈ sessions do
    for all consecutive couples of queries (q_1, q_2) ∈ session do
        query_node ← add or get node if already exists from I_q given (h(q_1), q_1)
        add next node q_2 to query_node
        I_t ← I_c(h(q_1.category)) if exists or a new hashTable
        for all term ∈ q_1.search_text do
            add (term, q_1) to I_t if the entry don't exists
        end for
    end for
end for
return I_q, I_c
```

presented in (Boldi et al., 2008). It is represented by an adjacency list, where each node is a unique couple <*category, query string*>. Two nodes are linked by an edge if the two queries appear at least in a session consecutively and the weight on the edges are integer values which mean the number of times the two queries appear consecutively in a session. An hash table is used in order to efficiently access the list, with a hash function on the couples <*category, query string*> as key. The indexes allows to trace back to a node of the graph starting from the couple <*category, term*>, where *term* is a word of the query. It is realized using two hash tables: a category hash table where the key is a function on the category id and the value is another hash table nested in the first one. This is a hash table of terms where the values are sets of queries which contains that term.

Algorithm 1 shows the pseudocode of the procedure for building the indexes and the query graph. It takes linear time in the number of the queries. Concerning this, we can notice that all the operations on the hash tables with the hash function h need constant time and the inner loop on the terms in the search text can be assumed upper bounded because, as we can see in section 3, the average number of terms per query is 2 and in the 99% of the queries, they contain less than 6 terms.

After having constructed the graph, it is recommended to prune it removing all the edges which have a low weight or the nodes with a small amount of occurrences for essentially two reasons: it should remove useless information that could undermine the quality of the results and for performance issues as logs always grow, so will do the data structures.

2.3 Creation of the Similarity Graph

A typical problem encountered by *session based* systems, which exploits query logs, is the high percentage of single queries or couples of queries that are present together in the logs.

Algorithm 2: Search for similar queries.

Require: input query q, number of queries to return k, category index I_c
Ensure: list of similar queries $similar_q$

```
I_t ← I_c[category(q)] {term index for the category}
similar_q ← ∅
for all term ∈ terms(q) do
    if I_t contains term then
        append all the queries from I_t[h(term)] to similar_q
    end if
end for
sort similar_q by similarity with q.
return the first k queries in similar_q
```

The problem of the sparsity of the queries emerges: given an input query, it is likely that the system will have few or no information about it. This is a typical issue of *session based* systems which exploits the query logs. In order to solve it the proposed algorithm also takes into account the similar queries already processed by the system. We say two queries are similar if they belong to the same category and have a similar search text. In order to measure the similarity between search texts the algorithm makes use of the Jaccard similarity coefficient (Tan et al., 2005) on the set of words of the search text, not considering the stopwords.

The procedure for computing the similarity measure is shown in Algorithm 2. It obtains the term hash table I_t for the category associated with the input query q and, for each *term* in the query, it adds the set of queries having this term to the set of all similar queries. Then, this list $similar_q$ is sorted by similarity in respect to the input query calculating the Jaccard similarity measure. Finally the algorithm returns the first k queries in that list. The complexity of this algorithm depends on the number of similar queries N_s. It needs $O(N_s \cdot \log(N_s))$ as it is the time for sorting the similar queries.

In order to avoid running the Algorithm 2 for each input query in the online phase, the system build a graph on the same set of nodes of the query graph defining new non oriented edges which represents the similarity relations among queries. This is very close to the index adopted by (Cao et al., 2008) for finding the queries given a query represented as a vector in the selected url space. It allows to look for the queries similar to the input search text when it is already present in the graph: this situation happens about half the time. Defining N_p as the number of nodes in the graph (after the pruning) and N_s the average number of similar queries per query, the similarity graph building process takes $O(N_p \cdot N_s \cdot \log(N_s))$.

Algorithm 3: Search for related queries.

Require: input query q, max number of recommended queries m, query and category indexes I_q, I_c
Ensure: set of queries *related*

```
related ← ∅
if I_q contains q then
    similar ← similar queries for I_q[q]
else
    similar = find_similar(I_c, q, k) {Algorithm 2}
end if
for all q' ∈ similar do
    related ← related ∪ next queries of q'
end for
sort related by the ranking function r
return the first m entries in related
```

2.4 Online Query Suggestion

Algorithm 3 shows how the online phase works. Given an input query q it acts as follow.

1. Search for the queries similar to q. The algorithm looks for the node $\hat{\mathbf{N}}_q$ in the query graph, either if it exists or not. If it exists, the algorithm selects the queries in the children nodes of $\hat{\mathbf{N}}_q$.
2. It selects all the nodes $\hat{\mathbf{N}}_s$ which represent a the queries similar to q, following the edge of the similarity graph. It selects the queries from the nodes next to $\hat{\mathbf{N}}_s$ and add them to a set as candidate queries for the suggestion.
3. The set of candidate queries is ordered by a ranking function r and the first m queries are returned. The adopted ranking function sums the normalized weights of the link in the similarity graph and the weight on the link in the query graph that allowed to get to it. In case of equality, the queries are ordered by the number of occurrences in the query logs.

In the worst case the complexity of Algorithm 3 depends on the call to Algorithm 2, that is *find_similar* in the listing. This call takes $O(N_s \cdot \log(N_s))$. The number of queries *related* can be considered constant as it is $k \cdot N_n$, where N_n is the number of subsequent nodes for a node in the query graph and k is the maximum number of similar queries for each node. Since the graph is sparse, N_n can be considered constant, so the time for ordering the queries in the *related* set is constant too.

3 EXPERIMENTS

An evaluation conducted on two datasets with different characteristics is fundamental for verifying the generic nature of the proposed system. The two datasets have the following differences:

1. *Number of queries per day*. *Shoppydoo* has about 200.000 queries per day, while *Trovaprezzi* about one million.
2. *Typologies of queries*. The users of *Shoppydoo* are usually more specialized, so the queries are more focused into some categories and they are more correct and precise. On the other hand, *Trovaprezzi* is for the most used by inexperienced users, so the queries are more equally distributed among several categories. Anyway, these queries are sometimes "wrong" as they contain words with no meaning or lead to no results.
3. *Session identifier*. On *Shoppydoo* the users are identified by IP address, while on *Trovaprezzi* by HTTP session ID. In the first case it is more difficult to obtain accurate user sessions because the queries from the same IP can be from several different users.
4. *Length of the sessions*. On *Shoppydoo*, a session last 2,5 queries in average, while 3 queries on *Trovaprezzi*.

For all the experiments we used a system with 2,4Ghz 32 bit CPU with 4Gb of RAM.

By analyzing the logs we noticed that about the 50% of the queries has a search text which appears only once, while the 6% appears two times. The amount of queries that are unique or that have few occurrences justifies the employment of methods to find similar queries in order to compute the suggestions.

The queries are very short and tend to be composed by two words. The 99% of all the queries have less then 6 words. This characteristic allowed us to consider constant the number of words per query while presenting the algorithms in Section 2.

In order to analyze the complexity of the algorithm we considered a collection of queries that goes from the queries submitted in a day to the set of queries submitted within eight days for *Shoppydoo* (240k to 1,8m queries), and a set of queries up to two days for *Trovaprezzi* (up to 2,4m queries).

3.1 Temporal Complexity

Figure 1 shows the time needed for building the query graph and for pruning it. In this experiment we choose to eliminate the links in the query graph that have unitary weight, that is the links between queries appear as consecutive in the sessions only once. The same graph shows the time necessary to create the similarity graph. The maximum number of links per node is set to 16, that is the double of the number of suggested

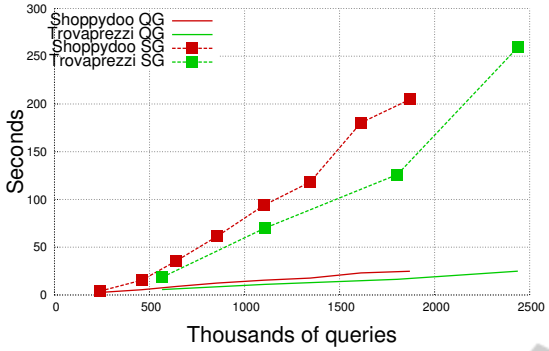

Figure 1: Time needed to build the query graph (QG) along with the indexes for the two datasets and the time needed for the similarity graph (SG) on both the datasets.

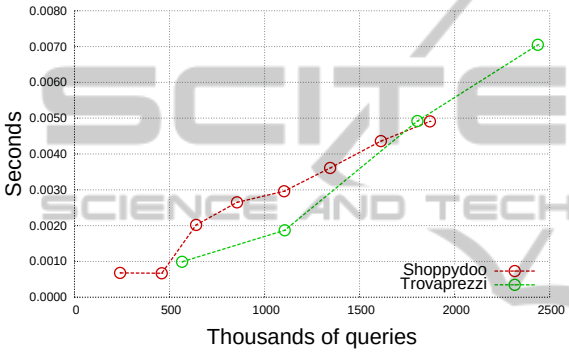

Figure 2: Online time necessary to generate the suggestions varying the number of analyzed queries.

queries the online phase would return in a reasonable setup.

Finally, in Figure 2 we can see the average time taken by the online phase. This value has been calculated using 500 different queries that do not belongs to the set of queries used to build the underlying model. Even if we built the graph with an increased number of queries it would be possible to maintain constant this times with a pruning or with a simplification of Algorithm 2 used to find the similar queries, for example modifying it to look for them only in the similarity graph and not searching for the similar queries if we do not find the entry in the query index as reported in Algorithm 3.

3.2 Quality Evaluation

In order to evaluate the results of the query suggestion algorithm we adopted two metrics similar to what it is possible to find in literature: the *coverage*, which indicates for how many input queries the algorithm returns at least a minimum amount of suggestions, and the *quality*, which denotes how many suggestions which are useful to the user we obtain. It was not possible to confront the proposed solution with the evaluations available in literature because we do not use a unique dataset. Thus, we chose to use different parameters for the evaluation for example the number of required suggestions in order to consider the results satisfactory, or which input query to employ in the quality tests. Regarding this last issue, our tests use random queries taken from the set of all the queries available from the web server log and do not select, for example, the most frequent ones.

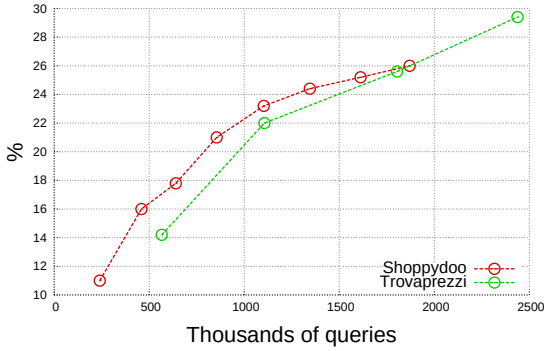

Figure 3: Coverage as the number of queries increases.

For the evaluation of the *coverage* we used 500 queries selected randomly from the logs. A set of suggested queries is considered sufficient if it contains at least eight suggestions. We chose this number because the main search engines display a number of suggestions close to the chosen one. For instance, Google and Bing display eight suggestions while Yahoo from three up to ten.

The employed hardware for the following evaluations is the same described in the header of this section with, in addition, a 64 bit Intel Xeon 2,66Ghz system with 8Gb of RAM used to perform the offline phase, that is the building of the data structures.

The experiments conducted led to the results presented in Figure 3. Another experiment was run using the queries for five days from *Trovaprezzi*, reaching a *coverage* of 37,2%.

In the end, we evaluate the *quality* of the queries that are suggested by the proposed algorithm. The evaluation is performed by humans using a web application designed for this purpose. The user who utilizes the application is asked if he considers the suggestions related to the original query he submitted or not. Thirty people were involved in this task and we collected about 150 opinions. The 70,1% of these expressed a positive judgment about the correlation of the suggested queries with the original one and its usefulness.

The underlying model was built using five days of queries from the log of *Trovaprezzi*, that is about 6 million queries.

The proposed system returns bad results espe-

cially if it is given long and inaccurate queries. As the length of the query increase, the system is not able to find equal or at least very similar queries in the graph, so the suggested queries are too generic in respect to the original intent of the user or they lacks of correlation.

Taking a look to the queries that led to good suggestions, we noticed they are manly specific product names, product types and brands. For this kind of queries the system is able to reformulate the search texts for *specialization*, *equivalent reformulation* and *parallel movement*.

The web application devised to evaluate the quality has also been employed for measuring the response times of the query suggestion system, logging the time taken for generating the page with the suggestions. The average time has been 0,0059 seconds, which allows to employ the system in an online real time environment.

4 CONCLUSIONS

The initial objective was the realization of a solution in order to enhance the search feature in an web application for price comparison implementing a query suggestion system. We realized a system that could take advantage from all the available data about the queries submitted to the web sites, while keeping a generic approach as much as possible, in order to allow the proposed solution to be applicable even on different search engines.

The implemented system is considered satisfactory in respect to the requirements we had set. This is confirmed by the experiments where, given 6 millions queries from a web site logs, the users consider the suggestions good, measuring a quality of 70% and a coverage of 37%, which are the queries which lead to at least eight suggestions.

Thus the performance are good, as the system in the online phase need about 1,3Gb of memory and it responds with a latency less then one hundredth of second.

The most promising possible future developments involve two aspects of the system. Firstly, the improvement of the ranking function, adding more parameters to consider clicks and relations among suggested queries and click-through rates, thus considering a linear combination of more factors rather than just adding the normalized weights from the graphs. Secondly, the definition of different similarity measures in place of the Jaccard index.

REFERENCES

Baeza-yates, R. A. (2007). *Graphs from Search Engine Queries*.

Baeza-yates, R. A., Hurtado, C. A., and Mendoza, M. (2004). *Query Recommendation Using Query Logs in Search Engines*.

Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., and Vigna, S. (2008). The query-flow graph: model and applications. In *International Conference on Information and Knowledge Management*, pages 609–618.

Boldi, P., Bonchi, F., Castillo, C., and Vigna, S. (2009). From "dango" to "japanese cakes": Query reformulation models and patterns. In *Web Intelligence*, pages 183–190.

Broccolo, D., Frieder, O., Nardini, F. M., Perego, R., and Silvestri, F. (2010). *Incremental Algorithms for Effective and Efficient Query Recommendation*.

Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., and Li, H. (2008). Context-aware query suggestion by mining click-through and session data. In *Knowledge Discovery and Data Mining*, pages 875–883.

Mat-Hassan M., L. M. (2005). Associating search and navigation behavior through log analysis. *Journal of the American Society for Information Science and Technology*, 56(9):913–934.

M.P. Kato, T. Sakai, K. T. (2011). Query session data vs. clickthrough data as query suggestion resources. In *ECIR 2011 Workshop on Information Retrieval Over Query Sessions*.

Ortiz-Cordova A., J. B. (2012). Classifying web search queries to identify high revenue generating customers. *Journal of the American Society for Information Science and Technology*. cited By (since 1996) 0; Article in Press.

Pierrakos, D., Paliouras, G., Papatheodorou, C., and Spyropoulos, C. D. (2003). Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13:311–372. 10.1023/A:1026238916441.

Shoppydoo (2012). http://www.shoppydoo.it.

Srivastava, J. and Cooley, R. (2000). Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1:12–23.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison Wesley.