Efficient Query Recommendations in the Long Tail via Center-Piece Subgraphs

Francesco Bonchi
Yahoo! Research, Spain
bonchi@yahoo-inc.com

Raffaele Perego
ISTI - CNR, Pisa, Italy
r.perego@isti.cnr.it

Fabrizio Silvestri
ISTI - CNR, Italy; and
Yahoo! Research, Spain
f.silvestri@isti.cnr.it

Hossein Vahabi
IMT, Lucca, Italy
hossein.vahabi@imtlucca.it

Rossano Venturini
Dept. of Computer Science,
University of Pisa
rossano@di.unipi.it

ABSTRACT

We present a recommendation method based on the well-known concept of *center-piece subgraph*, that allows for the *time/space efficient* generation of suggestions also for *rare*, i.e., long-tail queries. Our method is scalable with respect to both the size of datasets from which the model is computed and the heavy workloads that current web search engines have to deal with. Basically, we relate terms contained into queries with highly correlated queries in a query-flow graph. This enables a novel recommendation generation method able to produce recommendations for approximately 99% of the workload of a real-world search engine. The method is based on a graph having *term* nodes, *query* nodes, and two kinds of connections: term-query and query-query. The first connects a term to the queries in which it is contained, the second connects two query nodes if the likelihood that a user submits the second query after having issued the first one is sufficiently high. On such large graph we compute the center-piece subgraph induced by terms contained into queries and we reduce the cost of this computation using a novel and efficient method based on an *inverted index* representation of the model. We experiment our solution on two real-world query logs and we show that its effectiveness is comparable (and in some case better) than state-of-the-art methods for head-queries. More importantly, the quality of the recommendations generated remains very high also for long-tail queries, where other methods fail even to produce any suggestion. Finally, we extensively investigate scalability and efficiency issues and we show the viability of our method in real world search engines.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process, Query formulation

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.
Copyright 2012 ACM 978-1-4503-1472-5/12/08 ...$10.00.

General Terms

Algorithms, Experimentation, Measurement

Keywords

Query Recommender Systems, Web Search Effectiveness

1. INTRODUCTION

Much of the literature on query recommendation is devoted to propose novel "methods", or "models", that enhance *effectiveness*. Although this is, clearly, a fundamental issue, the other side of the coin, *efficiency*, has indeed been poorly addressed by the research community. The key idea behind query recommendation is that of exploiting the so-called "*wisdom of the crowds*", i.e., the knowledge mined from search engine query logs which store all the past interactions of users with the search system. For this reason query recommenders are more effective when the information need of the user is a popular one, i.e., the same query has been been frequently submitted by other users in the past. Using common wording, these queries are *head queries* indicating that they are usually appearing in the head of the power-law-like curve typical of query popularity distribution. In this paper we introduce a novel recommendation method based on computing the *center-piece subgraph* [14] on a large graph-model. Our solution presents several enhancements with respect to the state-of-the art. Firstly, the method is scalable and efficient. It is scalable as the generation of the model is easily parallelizable and the model itself can be stored in a compressed form. Furthermore, we represent the model in an inverted index and we show that several engineering practices used for inverted indexes are inherited by our model as well. The inverted index representation has several advantages as, for instance, the possibility of exploiting the existing index processing infrastructure of search engines with small, or no, modifications. The suggestions generation time, also, is comparable to that taken by the query processing phase. Thus, generating recommendations does not represent a bottleneck even when rare, uncached queries are processed. Even if the main goal of this paper is that of designing an efficient and implementable system, we show that the quality of its suggestions is comparable (and sometimes even better) with respect to state-of-the-art method, query flow graph. Interestingly, the quality of the suggestions produced by our system is stable, almost independently of the frequency of the query in the query log used to learn the model. This is a key property which does not hold for the query flow graph. Query flow graph is, indeed, not able to generate suggestions for previously unseen or rare queries.

More in details, our method is based on a graph-model that we dub TQGraph (*Term-Query Graph*). A TQGraph extends the well-known Query Flow Graph [3] by considering two distinct sets of nodes: *Term* and *Query* nodes. Arcs connect a term node to all the query nodes containing it, while arcs between two query nodes expresses the likelihood that a user submits the second query after having issued the first one. We design such a structure so that we are able to generate recommendations for a query by extracting the *center-piece subgraph* [14] associated with terms of the query itself. It is important to remark that since we do not require a query to be present in the TQGraph, but only its terms, our method is able in principle to provide recommendations even for a never-seen-before query.

It is this term-centric perspective that allows us to provide a framework enabling suggestions to be efficiently generated on the fly during query processing. Finally, let us resume the original contributions of this paper:

- A novel method for query recommendation based on center-piece graph computation over the TQGraph [1] model. Being term-centric, our center-piece-based method does not suffer from the problem of sparsity of queries, being able to generate suggestions for previously-unseen queries as far as terms have been previously seen. Empirical assessment confirms that our method is able to generate suggestions for the vast majority (i.e., 99%) of queries and with a quality that is comparable to (and in some cases better than) the state-of-the-art method based on Query Flow Graph.

- After having proved the effectiveness our method, we are faced with its major but only apparent drawback: any suggestion pass through the computation of the center-piece subgraph from query terms. Given the very large size of the underlying graph, this task is clearly unfeasible for any system claiming to be real-time. We overcome to this limitation by introducing a novel and efficient way to compute center-piece subgraphs. This comes at the small cost of storing precomputed vectors that need to be opportunely combined to obtain the final results. The data structure we use is *inverted list-based* and thus it is particularly suitable for web search environments.

- The inverted-list-based data structure is compressed by using a *lossy compression* method able to reduce by an average of 80% the space occupancy of the uncompressed data structures. Furthermore, caching is exploited at the term-level to enable scalable generation of query suggestions. Being term-based, caching is able to attain hit-ratios of approximately 95% with a reasonable footprint (i.e., few gigabytes of main memory.) Furthermore, since the compression method is lossy, we have evaluated through a user study the loss in terms of suggestion quality, and we have found that this loss is negligible. We can claim that our method to generate suggestions is very effective (and efficient) also when a very aggressive lossy compression strategy is applied.

It should be noted that the last two research results, beyond enabling our model to be real-time, represent a more general achievement for what concerns the computation of center-piece subgraphs in very large graphs.

The paper is organized as follows. Section 2 presents related research results. Section 3 details the TQGraph model and presents the methods used to compute suggestions. In Section 4 we assess the quality of the recommendations computed by our method and we compare our results with state-of-the-art QFG. Sections 5 and 6 present and experimentally evaluate our scalability and efficiency enhancing techniques. Finally, in Section 7, we discuss future work and conclude.

2. RELATED WORK

Hanghan *et al.* in [14] propose the Center-Piece Subgraph, as the subgraph that best captures the connections of a set of nodes in a graph. The computation of the Center-Piece Subgraph is based on the *Hadamard* (i.e., component-wise) product of a set of vectors, where each vector is obtained by doing a random walk with restart from a single node. Due to the long processing time of random walks with restart, the method is unfeasible for real-time application on large graph. The authors themselves propose a way to speed up the process by precomputing a $n \times n$ matrix, where n is the number of nodes in graph. This technique helps on small graphs, but is unfeasible for large graphs due to space requirements. This paper contributes with a novel way to compute efficiently center-piece sub-graphs on large graphs. The technique requires to precompute probability distributions and storing them in an inverted index that need to be opportunely processed to obtain center-pieces. Moreover, several optimizations, namely pruning, bucketing, and compression can be applied in order to reduce the memory footprint.

The techniques proposed during last years for query recommendation are very different, yet they have in common the exploitation of usage information recorded in query logs.

Baeza-Yates *et al.* [1] exploit click-through data on query logs as a way to provide recommendations. The method is based on the concept of *Cover Graph* (CG). A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs.

Among the proposals exploiting the chains of queries stored in query logs, [7] use an association rule mining algorithm to devise frequent query patterns. These patterns are inserted in a query relation graph which allows "concepts" (queries that are synonyms, specializations, generalizations, etc.) to be identified and suggested. Jones *et al.* introduce the notion of query substitution or query rewriting, and propose a solution for sponsored search [9]. Such solution relies on the fact that in about half sessions the user modifies a query with another which is closely related. Such pairs of reformulated queries are mined from the log and used for query suggestion.

Boldi *et al.* introduce the *Query Flow Graph* [3] (QFG) model, which is a Markov chain-based representation of a

[1] A preliminary presentation of the idea of the TQGraph, was given in a poster at WWW'11 by the same authors: http://www.www2011india.com/proceeding/companion/p15.pdf.

query log. A QFG is a directed graph in which nodes are queries, and an edge $e = (q_1, q_2)$ exists if at least a user has issued q_2 after q_1. Furthermore, e is weighed by the probability of a user to issue q_2 after q_1. Given a query q, suggestions are generated by means of random walks from q on the QFG [4]. It is worth noting that the "central" part of our TQGraph (i.e, the subgraph induced by only the query nodes, without the term nodes) corresponds to a QFG, whose construction is discussed in detail in the next section.

The importance of rare query classification and suggestion recently attracted a lot of attention. Generating suggestions for rare queries is in fact very difficult due to the lack of information in the query logs. As it was pointed out by Downey *et al.* [6] trough an analysis on search behaviors, rare queries are very important, and their effective satisfaction is very challenging for search engines. The authors also study transitions between rare and common queries highlighting the difference between the frequency of queries and their related information needs.

Mei *et al.* propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large scale bipartite graph [10]. The rationale of the method is to capture semantic consistency between the suggested queries and the original query. Empirical results on a query log from a real world search engine show that hitting time is effective to generate semantically consistent query suggestions. The authors show that the proposed method and its variations are able to boost long tail queries, and personalized query suggestion. Also a recent work by Yang *et al.* [12] proposes an optimal framework for rare-query suggestion leveraging on implicit feedbacks from users mined from the query logs.

Broder *et al.* [5] proposes an efficient and effective approach for matching ads against rare queries. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Xu and Xu [15] designes a way to learn similarity functions that are well suited for rare queries. The method leverages the knowledge extracted from past queries and build a locality sensitive hashing function thrugh which similarity of rare queries is estimated. Jain *et al.* [8] modifies the terms in rare queries in order to match more frequent queries in the query log.

All the previous proposals suffer from a main limitation which regards the granularity of the atomic items represented in the recommendation model. In the literature the granularity is always at the query level, and thus the suggestion algorithms can provide recommendations only to queries already seen in the past and present in the training log. This paper proposes a different solution in which the knowledge learned from the query log is coded at the granularity of the single terms present in the queries used for training. As a consequence, differently from competitors, our solution can generate suggestions even for queries not occurring in the training log and never submitted in the past. The only constraint is in fact on the presence in the training log of the terms used for expressing the query.

A totally orthogonal proposal to deal with query recommendations for long-tail queries has recently been presented by Szpektor *et al.* [13]. They propose to extract rules between *query templates* rather than individual query transitions, as currently done in session-based models. The method applies general rules learned from the log to rare queries fitting the rule. As an example, if the template $<$ *city* $>$ *hotels* $\rightarrow$ $<$ *city* $>$ *restaurants* holds strongly in the log, and *Montezuma* is recognized as a city in the rare query *Montezuma hotels*, then *Montezuma restaurants* is generated as possible query recommendation, even if it was not present it the training log.

Obviously this method extends the coverage of query recommendation to the long-tail from a different standpoint w.r.t. our proposal. While our method can not suggest *Montezuma restaurants* in the hypothetical case that the term *Montezuma* has never been seen before, the method based on query templates can only suggest recommendations for query for which templates exist. The two proposal are hence totally orthogonal and as such they could be potentially combined.

A second limitation common to some of the previous proposals is on efficiency. Query suggestions for most popular queries can be cached with query results themselves, but the time needed to generate relevant suggestions for queries that are not cached must necessarily be comparable to the query processing time. This need makes practically unusable all the methods requiring complex computations over large graphs, e.g., random walks over a huge QFG [4]. As discussed above, our novel solution to compute efficiently center-piece sub-graphs on large graphs solves this very important issue.

3. THE TQGRAPH MODEL

Let $Q = \langle q_1, \ldots, q_n \rangle$ be a query log, i.e., a set of queries each annotated with an anonymized *userid* and *timestamp* representing when the query has been submitted.

TQGraph is a digraph $G = (V, E)$ with vertices V and arcs E defined as follows. Let T be the set of all the distinct terms appearing in queries of Q. V contains a node for each term $t \in T$ and for each query $q \in Q$. In particular, let V_T be the set of *Term* nodes and V_Q be the set *Query* nodes, then $V = V_T \cup V_Q$. Likewise, E is the union of two different sets of arcs E_Q and E_{TQ}. Arcs in E_Q are defined as in QFG [3] and connect only nodes in V_Q. E_{TQ} contains arcs (t, q) where $t \in T$ is a term contained in query $q \in Q$. Finally, let $w : E \rightarrow (0..1]$ be a weighting function assigning to each arc $(u, v) \in E$ a value $w(u, v)$ defined as follows. For arcs $(t, q) \in E_{TQ}$, $\frac{w(t,q)=1}{d}$ where d is the number of distinct queries in which the t occurs, i.e., the number of outlinks of t. For arcs $(q, q') \in E_Q$ we follow QFG weighting scheme. In the original QFG paper, Boldi *et al.* [3] describe two distinct weighting schemes, namely *chaining probability* and *relative frequencies*. In the case of arc weighting it has been shown that *chaining probability* is the most effective scheme. Therefore, we resort to chaining probability for arcs in E_Q. To estimate such chaining probability, we extract for each arc $(q, q') \in E_Q$ a set of features. Such features are aggregated over all sessions in which queries q and q' appear consecutively and in this order. Finally, the chaining probability is computed by using logistic regression. Noisy arcs, i.e., arcs having a probability of being traversed lower than a minimum threshold value, are removed. In other words, query reformulations that are not likely to be made are not considered. For further details regarding the features and the model, we refer to the original work of Boldi et al. [3]. In particular, we have used the settings suggested in the original paper [3] for the values of the various parameters involved in QFG building.

3.1 Query suggestion method

Given our TQGraph G, the query suggestions for an incoming query q composed of terms $\{t_1, \ldots t_m\} \subseteq T$ are generated from G by extracting the center-piece subgraph [14]

	#queries	#terms	$\lvert V_Q \rvert$	$\lvert V_T \rvert$	dang.	$\lvert E_{TQ} \rvert$	$\lvert E_Q \rvert$	$\overline{d}_{TQ}$	$\overline{d}_Q$	#queries
										freq = 1
											#terms
											freq = 1

Yahoo!	580,797,850	1,343,988,549	28,763,637	6,261,105	14.5%	83,808,761	56,250,874	13.38	1.95	162,221,967	5,099,145
MSN	14,899,247	35,697,149	6,488,713	2,014,547	35.2%	19,740,312	5,051,843	9.79	0.77	4,992,180	1,633,729

Table 1: Statistics of the two query logs and TQGraphs. Total number of queries and terms. Number of query nodes and term nodes. Percentage of dangling nodes. Number of arcs from terms to queries and from queries to queries, and corresponding average degrees. Number of queries and terms with frequency 1.

starting from the m Term nodes corresponding to terms $t_1, \dots, t_m$. Given a directed graph and m of its nodes, the center-piece subgraph is informally defined as a small subgraph that best captures the connections between the m nodes. In our case the center-piece subgraph represent the set of queries that better represent terms of the original query q. It is important to point out, here, the following important fact: *in order to build a center-piece subgraph from q is not necessary that q is contained in the* TQGraph.

The center-piece subgraph for a query q composed of terms $\{t_1, \dots t_m\} \subseteq T$ is obtained by performing a Random Walk with Restart (RWR) from *each one* of the m term nodes corresponding to terms in q. The resulting m stationary distributions are then multiplied component-wise. More formally, given an incoming query $q = \{t_1, \dots, t_m\}$ we compute m RWRs from the m query-terms of q to obtain m vectors of stationary distribution $\mathbf{r}_{t_1}, \dots, \mathbf{r}_{t_m}$. Then, we compute the *Hadamard* (i.e., component-wise) product of the m vectors $\mathbf{r}_{t_1} \circ \mathbf{r}_{t_2} \circ \dots \circ \mathbf{r}_{t_m}$ to obtain the final scoring vector $\mathbf{r}_q$. Following the definition of Hadamard product, the i-th component of $\mathbf{r}_q$, i.e. $\mathbf{r}_q\,(i)$, is given by $\mathbf{r}_q\,(i) = \prod_{j=1}^{m} \mathbf{r}_{t_j}\,(i)$. Since each dimension of $\mathbf{r}$ corresponds to a query in Q, the TQGraph recommendation algorithm suggests the k queries having the k highest scores, where k is a parameter determining the maximum number of recommendations we want to show for each query. The reason for resorting to the product of the entries (namely, *Center-piece*) instead of their sum (namely, *Personalized PageRank*) is that we are interested in discovering queries that are "*strongly*" related to "*most of the terms*" in the starting query instead of queries that are highly related even to just few of them[2] It is also worth being remarked that, while Personalized PageRank could have been computed directly on the QFG, computing the center-piece subgraph related to the query terms can be done only on the TQGraph. Computing center-piece starting from the queries in QFGcontaining those terms, in fact, would prevent those queries themselves to be returned as suggestions. Obviously, this is not the case for the QFG-based model.

The following toy example shows how suggestions are computed using the center-piece-based TQGraph model. Consider a query log containing only two queries, q_1, q_2, made up from a vocabulary of three terms, t_1, t_2, t_3. Suppose that the RWR procedure described above leads to the following three stationary distributions: $\mathbf{r}_{t_1} = [0.9, 0.1]$; $\mathbf{r}_{t_2} = [0.3, 0.5]$; $\mathbf{r}_{t_3} = [0.09, 0.91]$. Finally, let k be equal to 1. According to this model, when a user submits a query containing terms (t_1, t_3) the system would compute the vector of scores $\mathbf{r}_q = [0.081, 0.091]$. Thus q_2, i.e., the top-1 center-piece subgraph, would be chosen as suggestion since, in this case, its score is greater than q_1 score.

4. EFFECTIVENESS

Ideally, a recommendation model has to produce high quality recommendations for the largest number of queries possible. The fraction of queries for which a method is able to generate recommendations, henceforth referred to as *coverage*, is of paramount importance in order to satisfy as much users requests as possible. It turns out that coverage is one of the major weak points of previously proposed solutions. As we shall see in the following our method is extremely robust w.r.t. this problem, and it is able to generate (useful) recommendations for a very large fraction of users' queries.

Unfortunately, defining and measuring quality of recommendations is not an easy task. It turns out to be, in fact, a subjective matter, usually measured on the basis of user-studies comparing a given method with some baselines. Following the prevailing custom, we will rely on an extensive user-study to assess effectiveness. Finally, to let the reader to appreciate the quality of our methods, we will discuss some anecdotal evidences on a small set of user-queries.

Query logs. We use two different query logs coming from two popular web search engines, namely Yahoo! and MSN.

- Yahoo! query log consists of approximately 600 millions of anonymized queries sampled from Yahoo! USA queries submitted within a short period of time in spring 2010. The TQGraph built on the Yahoo! log consists of $6,261,105$ term nodes and $28,763,637$ query nodes. The number of term-query arcs is $83,808,761$ whereas the number of arcs between query nodes is $56,250,874$.

- MSN is the Search Spring 2006 Query Log, released in the context of the 2009 Workshop on Web Search Click Data[3], containing approximately 15 millions queries from the USA search volume. The TQGraph built on this log consists of $2,014,547$ term nodes and $6,488,713$ query nodes, $19,740,312$ term-query arcs, and $5,051,843$ query-query arcs.

Table 1 reports some statistics on the two TQGraphs.

On coverage. Before describing recommendation effectiveness we discuss query coverage. As we have discussed before, the main advantage of our proposal over the state of the art, is the capability of providing useful recommendations also for "difficult" queries (i.e., rare or never-seen-before). Let us recall that, in order to produce recommendations for a given query, our method needs all the terms contained in the query to belong to the TQGraph. Or in other terms, our model fails to give a recommendation only when it receive a

[2] For the sake of completeness, a comparison of the suggestions produced by RWR from query nodes containing term queries and our TQGraph-based model is discussed in Section 4.

[3] http://research.microsoft.com/en-us/um/people/nickcr/wscd09/

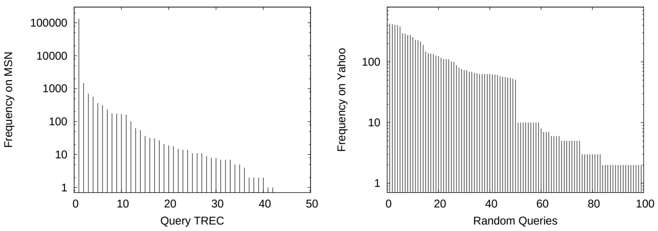

Figure 1: Frequency in the corresponding log of all the queries in the two testbeds.

never-seen-before term. While unique queries are quite frequent, unique terms are not. For instance, out of the 580.8 million of queries contained in the Yahoo! query log about 162.2 million of them are unique, in the same log the number of unique term is, instead, 5.1 million (see Table 1). Therefore, on this query log the coverage of our model is more than 99%, while the maximum coverage of QFG is 73% (i.e. the percentage of repeated queries).

User-study. Having pointed out the almost perfect coverage of our method, we next focus on evaluating the quality of the recommendations produced, evaluated by conducting a user-study. As a baseline for comparison we used recommendations provided by the state-of-the-art method QFG. To produce recommendations from the QFG we follow the method presented in [4]: recommendations are based on the probability of being at a certain node after performing a random walk over the QFG. This random walk starts at the node corresponding to the input query. At each step, the random walker either remains in the same node with probability 0.9, or follows one of the out-links with probability 0.1. in the latter case the links are followed with probability proportional to $w(i, j)$.

The user study was conducted on two different sets of queries. The first one is the composed by the 50 queries of the standard TREC Web diversification track testbed[4] that we use for the MSN query log. The second is a set of 100 queries randomly chosen from the Yahoo! query log. Figure 1 reports the distribution of the frequency of the queries in the two sets in the corresponding query log. We can observe that 8 queries in the TREC testbed do not appear at all in the MSN query log. The assessment was conducted by a group of 10 assessors (researchers not working on related topics). To reduce the load on our assessors we conducted only two, out of the four (2 query logs × 2 testbeds) possible user-studies. In particular we pair the Yahoo! set of queries up with the TQGraph and QFG models built over the same log (obviously the queries were randomly drawn by a portion of the log not used to build the suggestion models). Whereas, we pair TREC queries up with the models built on the MSN query log. In fact, the period from which TREC topics come, is close to the period in which MSN queries were collected.

We generated the top-5 recommendations for each query by using both the QFG and the TQGraph-based method with different parameters setting. Using a web interface each assessor was presented a random query followed by the list of all the different recommendations produced. Recommendations were presented shuffled, in order for the assessors to not be able to distinguish which system produced them. We give assessors the possibility to observe the search engine results for the original query and the recommended query that was being evaluated. The assessors were asked to rate a recommendation using one of the following scores: *useful*, *somewhat useful*, and *not useful*[5].

TREC on MSN	useful	somewhat	not useful
$\alpha = 0.9$	57%	16%	27%
$\alpha = 0.5$	32%	13%	55%
$\alpha = 0.1$	22%	12%	66%

100 queries on Yahoo!	useful	somewhat	not useful
$\alpha = 0.9$	48%	11%	41%
$\alpha = 0.5$	41%	20%	39%
$\alpha = 0.1$	37%	20%	43%

Table 2: TQGraph effectiveness on the two different set of queries and query logs, by varying α.

TREC on MSN	useful	somewhat	not useful
TQGraph $\alpha = 0.9$	57%	16%	27%
QFG	50%	9%	41%

100 queries on Yahoo!	useful	somewhat	not useful
TQGraph $\alpha = 0.9$	48%	11%	41%
QFG	23%	10%	67%

Table 3: TQGraph and QFG effectiveness on the two different testbeds.

In first instance we evaluate the impact of the α parameter (we recall here that α is the restart value of the RWR, from each term of the query to be recommended). Table 2 shows the effectiveness of our method when varying α among three different values $\alpha = 0.1, 0.5, 0.9$. Results show that the best quality is achieved when $\alpha = 0.9$. Table 3 reports the results of the user study comparing effectiveness of the TQGraph-based and QFG-based recommendations. TQGraph-based recommendations are globally of higher quality than QFG-based ones. We further investigate (see Table 4) the effectiveness of our method with respect to three different classes of MSN queries: unseen, dangling, and others. A query is unseen if it does not appear in the training query log. A query is dangling if its corresponding node in the QFG has no outgoing edges. The remaining queries belong to the class others. We observe that QFG is unable to provide suggestions for queries in the first two classes while our method provides

TREC on MSN (unseen)	useful	somewhat	not useful
TQGraph $\alpha = 0.9$	46%	10%	44%
QFG	0%	0%	100%

TREC on MSN (dangling)	useful	somewhat	not useful
TQGraph $\alpha = 0.9$	60%	30%	10%
QFG	0%	0%	100%

TREC on MSN (others)	useful	somewhat	not useful
TQGraph $\alpha = 0.9$	59%	17%	24%
QFG	61%	13%	26%

Table 4: User study results on unseen, dangling and others queries.

[4] http://trec.nist.gov/data/web09.html

[5] The following very broad instruction was given to assessors: *A useful recommendation is a query such that, if the user submits it to the search engine, it provides new results that were not available using the original query, and that agree with the inferred user intent of the original query.* Of course there is a great deal of subjectivity in this assessment as the original intent is not known by the assessors.

suggestions of high quality. The two methods have almost the same quality for the third class of queries. A similar behavior has been observed in Yahoo! query log. We do not report the results due to space limitations.[6]

Anecdotal evidence. We next show an example of query recommendations. The query *"lower heart rate"* is one among the eight from the TREC testbed that does not appear at all in the MSN query log. Below we report the top 5 recommendations both using our TQGraph model and using RWR[7].

Query: lower heart rate

TQGraph **Suggestions**	**RWR Suggestions**
things to lower heart rate	broken heart
lower heart rate through exercise	prime rate
accelerated heart rate and pregnant	exchange rate
web md	bank rate
heart problems	currency exchange rate

We can observe that all the top-5 suggestions can be considered pertinent to the initial topic. Moreover, even if this is not an objective in this paper, they present some *diversity*: the first two are *how-to* queries, while the last three are queries related to finding information w.r.t. possible problems (with one very specific for pregnant women). The most interesting recommendation is probably *"web md"*, which makes perfect sense (WebMD.com is a web site devoted to provide health and medical news and information), and has a large edit distance from the original query. Recommendations produced by RWR are not relevant due to the effect we point out in Section 3.

5. EFFICIENCY

Since query suggestions have to be served online, a query recommender must compute them efficiently, possibly in real-time. In this section we introduce some novel techniques allowing the efficient generation of recommendations at query time. Such techniques are peculiar of the TQGraph as they can only be achieved thanks to the term-centric perspective.

We recall that given an incoming query q, the generation of suggestions requires to compute RWRs on the TQGraph from the nodes associated with the terms occurring in the query. For each term t, the stationary probability distribution resulting from the RWR is represented by a vector $\mathbf{r}_t$ that scores queries in Q according to the probability of reaching them in a random walk on the TQGraph starting from t. As discussed in Section 3, given an incoming query $q = \{t_1, \ldots, t_m\}$, our recommender system returns the k queries having the largest probabilities in the Hadamard product $\prod_{i=1}^{m} \mathbf{r}_{t_i}$.

Before introducing our optimized solution we want to point out the major drawbacks of the two trivial approaches that can be used for computing suggestions using our model.

The most trivial approach consists in simply computing the RWR on the TQGraph for each term t_i in the incoming query as it arrives, and in multiplying the resulting stationary distributions.

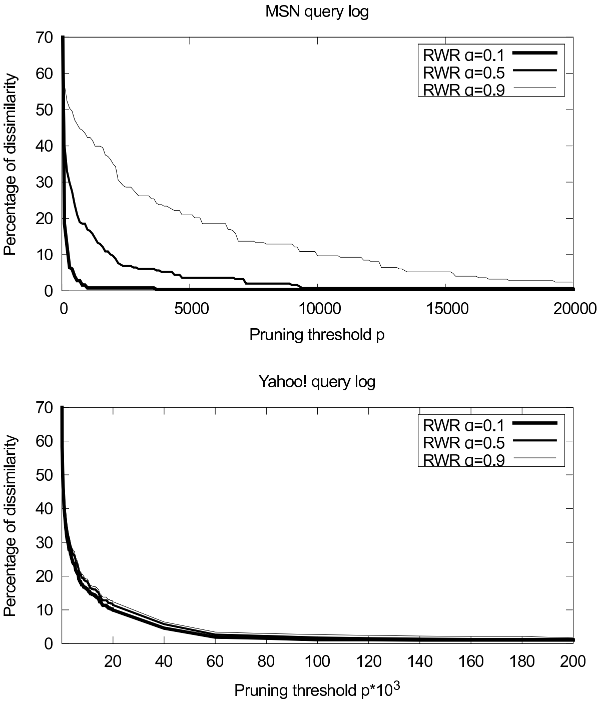

Figure 2: Dissimilarity (in percentage) for the top-5 suggestions as a function of the pruning threshold p, measured on the MSN (top) and Yahoo! (bottom) query logs. The curves refer to different values of the parameter α used in the RWR.

The second (less) trivial approach, instead, provides to store the precomputed stationary distributions for all the terms appearing in T. To improve efficiency, we can resort to use an index on which the stationary distribution of the random walks for terms in T are stored as lists of postings, where each posting is given by the identifier of the query (queryID), along with its probability. Recommendations for an incoming query are then computed by processing the posting lists associated with the terms composing the query.

Both approaches suffer from crucial time or space inefficiencies. The former approach requires $\Omega(m \cdot (|E| + |Q|))$ time for each suggestion, thus making it unusable in any online recommender system. As far as the latter approach is concerned, it has its main drawback in the space occupancy. Indeed, the algorithm for computing recommendations is significantly simpler and its time complexity is lower (i.e., $O(m \cdot |Q|)$). Storing all the stationary distributions requires to store $|T|$ different vectors of $|Q|$ entries each (namely, the ith entry of each vector is the probability of the i-th query in the stationary distribution of a term). The space required to store these $|T| \cdot |Q|$ entries is unfortunately prohibitive even for quite small query logs. For example, we notice that using such a approach for the TQGraph built over the relatively small MSN query log would ask to store a total of 13×10^{12} entries which is clearly not feasible in any real system.

In the following we show how the two above-mentioned drawbacks can be avoided by using three different optimizations, namely *pruning, bucketing*, and *compression*. The

[6] We observe that results in Table 3 and Table 4 are consistent because we considered as *not useful* the cases in which a method is not able to provide any suggestion.

[7] RWR corresponds to summing the entries of score vectors instead of computing the Hadamard product.

goal is to sensibly reduce space requirements, thus making our query suggestion method viable.

Pruning Lists. In order to reduce the space occupancy of the latter approach we consider to prune unnecessary entries in each list. The idea is to store only the probabilities of the top-p entries of each stationary distribution, where p is a user defined threshold. In this way we require to store p entries per term instead of $|Q|$.[8] The total number of entries to be stored becomes thus $p \cdot |T|$, with a large saving in space occupancy when $p \ll |Q|$. Obviously, this pruning phase comes at the price of introducing errors in the scores computed by the recommender. Assume that the k suggestions for a query $q = \{t_1, t_2, \dots, t_m\}$ are the queries $q_1, q_2, \dots, q_k$. The pruning phase introduces an error whenever one of these top-k queries has been pruned in the list of at least one of the terms $t_i \in q$.

We evaluated experimentally the effects of pruning, and report in Figure 2 the results of these tests. In particular we measured the average dissimilarities for the top-5 suggestions returned before and after pruning, by varying the number p of entries maintained for each term. Given two set A and B of size k, the **dissimilarity** among A and B is defined as $1 - \frac{|A \cap B|}{k}$. This experiment has been repeated by varying also the parameter α used in the RWR. In both cases the largest loss is obtained for RWR $\alpha = 0.9$. This is due to the fact that with this value of restart the most probable queries for a term are more likely to differ from the ones of other terms. This increases the chances for a query to be evicted from the list of at least a term of the user query. However, the experiments conducted show that relatively small values of p lead to a average similarity larger than 95% between the sets of top-5 results.

For example, it is possible to preserve the correctness of 97,6% of the top-5 results by setting $p = 20,000$ on the MSN query log. Note that on MSN 20,000 queries account for only 0.67% of the total number of queries $|Q|$ present in the log. For Yahoo! query log we can instead preserve 95.40% of the results by setting $p = 100,000$. In this case the number of non pruned queries is just 0.34% of $|Q|$. These figures show that about 97% of the queries in each list can be safely pruned away without remarkably affecting the quality of results. This phenomenon can be explained by observing that usually the terms in the user queries are highly related to each other. Thus, it should be not surprising that relevant queries have relatively high probabilities in the lists of all these terms.

Table 5 shows some figures related to the percentage of dissimilarity measured for some values of p on the two query logs. The average dissimilarity after pruning for the top-5 suggestions returned for all the queries in our testbeds, is computed by considering only those recommendations that have been deemed to be sufficiently good by assessors in the user study (namely, recommendations having been classified as *useful* or *somewhat useful* by at least an assessor). Dissimilarity, in percent, due to pruning for the whole lists of the top-5 recommendations is indicated between parenthesis.

As far as the space occupancy is concerned, we recall that the list of each term is formed by a pair of values, for each of the p most probable queries, i.e., the queryID and its probability. We represent each list in the form of a posting list. Firstly, we sort pairs by increasing queryIDs. Then, we encode differences between consecutive queryIDs by resorting to the well-known Elias' Delta coding (see [11] and references therein for more details on integers encoding methods). Finally, we encode the probability of each query in a fixed-length field of 8 bytes. The average number of bits required to store each pair ranges from 69 to 74 bits depending on the value of p. Complete experimental results are reported in parenthesis in Table 6. We observe that the larger the value of p, the denser are the lists for each term. This implies that gaps between queries IDs become smaller and, thus, each pair is more effectively encodable.

MSN query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5,000	1.18 (0.40)	5.31 (3.60)	13.66 (20.80)
10,000	1.18 (0.40)	1.77 (0.80)	7.10 (9.60)
15,000	1.18 (0.40)	1.77 (0.80)	6.01 (5.20)
20,000	1.18 (0.40)	1.77 (0.80)	3.28 (2.40)
100,000	1.18 (0.40)	0.88 (0.80)	0.00 (0.00)
200,000	1.18 (0.40)	0.00 (0.80)	0.00 (0.00)

Yahoo! query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5,000	21.75 (31.40)	24.26 (31.80)	25.08 (31.60)
10,000	15.09 (25, 00)	18.69 (26.20)	18.31 (25.40)
15,000	11.93 (20.80)	15.74 (22.40)	14.58 (21.20)
20,000	11.23 (18.20)	13.77 (20.00)	13.22 (20.00)
100,000	1.05 (1.80)	2.30 (3.00)	2.37 (4.60)
200,000	1.05 (1.60)	1.64 (2.20)	1.36 (2.60)

Table 5: Average dissimilarity (in percentage) between the sets of top-5 suggestions computed with or without pruning as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis.

MSN query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5,000	19.21 (73.63)	17.34 (73.98)	15.81 (73.71)
10,000	18.32 (73.31)	17.15 (73.44)	16.08 (73.44)
15,000	17.99 (73.16)	17.17 (73.16)	16.23 (73.32)
20,000	17.82 (73.03)	17.23 (72.96)	16.33 (73.21)
100,000	15.84(71.39)	16.09(71.28)	16.34(71.15)
200,000	15.43(70.22)	15.36(70.15)	15.21(70.05)

Yahoo! query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5,000	16.25 (72.17)	15.26 (72.26)	13.67 (72.33)
10,000	15.72 (71.43)	14.99 (71.54)	13.79 (71.63)
15,000	15.54 (71.17)	14.96 (71.29)	13.99 (71.42)
20,000	15.59 (71.03)	15.11 (71.17)	14.27 (71.32)
100,000	16.55 (70.47)	16.42 (70.67)	16.16 (70.87)
200,000	16.30 (69.94)	16.23 (70.12)	16.09 (70.30)

Table 6: Average bits per entry for our bucketing method ($\epsilon = 0.95$) and the baseline (between parenthesis) by varying p and α.

Approximating Probabilities. The above method allow to build a index of terms' RWRs, where the pruned list of queries for each term is coded as a postings list. The size of this index is used as baseline to assess the effectiveness of our more sophisticated solution that relies on approximations. We observed that the previous method spends most of its space in storing the probability $\mathbf{r}_t(q)$ for each query q

[8] The probability of a pruned query is assumed to be 0.

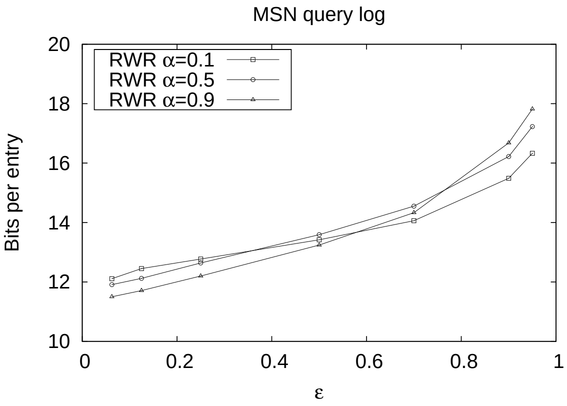

Figure 3: Average bits per entry on the MSN query log as a function of ϵ ($p = 20,000$).

in the stationary distribution $\mathbf{r}_t$. The idea is thus that of approximating each probability by a bucketing schema in a way that still preserves roughly indications of its magnitude. We start by choosing a parameter ϵ which is a real value in $(0, 1)$. We divide the query IDs in the list of a particular term t in buckets based on their probabilities. Let s be the smallest probability in the list. We have l buckets B_0, B_1, ..., B_{l-1} where $\epsilon^{l-1} \leq s < \epsilon^l$. The i-th bucket B_i contains the IDs of the queries whose probabilities are in the range $[\epsilon^i, \epsilon^{i+1})$. The approximated probability $\hat{\mathbf{r}}_t(q)$ of a query q in bucket B_i is approximated with ϵ^i. This organization in buckets is particularly suitable for compression. In our scheme we sort queries IDs in each buckets, then we encode gaps between consecutive queries in the same bucket. Finally, we encode the index and the cardinality of each bucket. For simplicity, all the values have been encoded by resorting to Elias' Delta coding. Different choices are possible since the literature offers a great variety of different solutions for these aims [11]. According to our schema, the decoding is very simple: IDs of queries are obtained by decompressing each bucket, while their probabilities are set to be equal to ϵ^i where i is the index of the corresponding bucket.

By resorting to this bucketing technique, we are able to achieve high levels of compression as shown in Figure 3. In this figure we report the average number of bits per entry required by our method with $p = 20,000$, as a function of the values of ϵ and α. The number of bits per entry ranges between 11 and 19. These figures are reported for the MSN query log only since a very similar behavior was observed on the Yahoo! query log. Notice that the smaller the value of ϵ, the smaller is the average number of bits per entry. This expected effect is due to the fact that with smaller values of ϵ we obtain fewer different buckets which are more dense. Thus, query IDs become more compressible. Table 6 compares the average number of bits per entry required by our schema ($\epsilon = 0.95$), with those required by the baseline (pruned lists without bucketing). The comparison was made by using both the MSN and Yahoo! query logs by varying p for three different values of α in the RWR. The number of bits per entry for the baseline are reported between parenthesis. The table shows that our scheme is much more space-efficient than the baseline (namely, each entry requires roughly 4 times less space).

Clearly, our bucketing scheme may introduce approximations on the probabilities of each query. However, by construction, we are able to precisely bound the highest possible level of approximation. The approximated probability of any query q is in fact at most ϵ^{-1} times larger than its real probability (namely, $\mathbf{r}_t(q) \leq \hat{\mathbf{r}}_t(q) < \epsilon^{-1} \cdot \mathbf{r}_t(q)$). Thus, the larger is the value of ϵ, the better are the guarantees on the resulting approximations. Since recommendations are computed by using these approximated probabilities, there may exist differences between the top-k queries suggested by resorting to real probabilities and the ones obtained with approximated probabilities. In other words, a query that is among the top-k with real probabilities may be replaced by another query when we resort to approximations. However, we can prove that this event happens only if the two queries have a very close product of (real) probabilities. More formally, let $q = \{t_1, t_2, \ldots, t_m\}$ be the user query and let q' and q'' be any pair of candidate queries. Assume that

$$\mathbf{r}_{q'} = \prod_{i=1}^{m} \mathbf{r}_{t_i}(q') > \prod_{i=1}^{m} \mathbf{r}_{t_i}(q'') = \mathbf{r}''_q.$$

Thus, q' precedes q'' in the ranking for query q computed via real probabilities and, thus, q' should be preferred to q''. The relative order in the ranking between q' and q'' computed via the approximated probabilities differs if[9]

$$\hat{\mathbf{r}}_{q'} = \prod_{i=1}^{m} \hat{\mathbf{r}}_{t_i}(q') \leq \prod_{i=1}^{m} \hat{\mathbf{r}}_{t_i}(q'') = \hat{\mathbf{r}}_{q''}.$$

Therefore, a change in the relative order of these two queries is possible only if their products of probabilities are too close. More precisely, since $\mathbf{r}_{q'} < \hat{\mathbf{r}}_{q'}$, $\hat{\mathbf{r}}_{q'} \leq \epsilon^{-m}\mathbf{r}_{q''}$ and $\mathbf{r}_{q'} > \mathbf{r}_{q''}$, the relative order between q' and q'' cannot change whenever $\mathbf{r}_{q'} > \epsilon^{-m}\mathbf{r}_{q''}$. The quantity ϵ^{-m} is sufficiently small since the number of terms m in the input query is usually a value between 2 and 3. For example, the average number of terms per query in MSN query log is $m = 2.40$. Thus, it suffices that $\hat{\mathbf{r}}(q') > 1.131 \times \hat{\mathbf{r}}(q'')$ in order to preserve the relative order between q and q'' with $\epsilon = 0.95$.

Table 7 shows the percentage of dissimilarity achieved with pruning and bucketing with respect to the original results computing considering the whole lists. Once more dissimilarity is computed for the top-5 results restricted to those recommendations that have been considered sufficiently good in the user study discussed in Section 4. At the first glance, we can see that the percentage of dissimilarity is remarkable. For example, a percentage ranging between 47.12% and 28.14% of the useful suggestions generated using the whole lists built from the Yahoo! query log are lost due to pruning and approximation. We observe however, that we are measuring exact differences in sets of results. This measure might not actually capture the global quality of the set of suggestions provided. It could happen in fact (and we will see that it often actually happens), that a good query suggested by using the whole lists is evicted by the set of suggestions due to pruning and bucketing to make place to a different query of comparable quality. The differences in the probabilities among the queries retrieved which are close to the fifth position, are in fact so small that our approximation could swap two queries having a very similar probability. This has a negligible effect on the overall quality

[9] In case of tails between products of probabilities one of the queries is preferred arbitrarily.

MSN query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5, 000	18.48 (8.47)	22.58 (14.11)	42.04 (40.32)
10, 000	17.39 (8.47)	20.97 (12.50)	39.49 (30.65)
15, 000	17.39 (8.47)	20.16 (12.10)	36.31 (25.40)
20, 000	17.39 (8.06)	18.55 (11.29)	33.12 (22.18)
100, 000	17.39 (8.06)	18.55 (11.29)	32.48 (21.77)
200, 000	17.39 (8.06)	18.55 (11.29)	32.48 (21.77)

Yahoo! query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
5, 000	33.75 (40.30)	37.87 (42.22)	45.11 (47.12)
10, 000	27.76 (34.12)	32.84 (36.89)	40.23 (42.22)
15, 000	26.18 (31.13)	31.36 (34.33)	38.22 (39.23)
20, 000	23.97 (27.72)	28.70 (31.56)	37.07 (38.38)
100, 000	19.24 (17.48)	21.89 (20.26)	31.32 (28.78)
200, 000	19.24 (17.70)	21.30 (19.62)	31.90 (28.14)

Table 7: Average dissimilarity (in percentage) between the sets of top-5 suggestions computed by resorting or not to bucketing (with $\epsilon = 0.95$) as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis.

MSN query log

p	useful	somewhat	not useful
5, 000	56%	17%	27%
20, 000	55%	15%	30%
200, 000	55%	15%	30%

Yahoo! query log

p	useful	somewhat	not useful
5, 000	46%	29%	25%
20, 000	47%	29%	24%
200, 000	46%	28%	26%

Table 8: Effectiveness of the suggestions provided with pruning and bucketing as a function of p for $\epsilon = 0.95$ and $\alpha = 0.9$.

of the suggestions provided, but it accounts for a 20% error according to our metrics. In order to verify this hypothesis, we thus conducted a new user study to evaluate the recommendations generated with the index exploiting pruning ($p = 5,000; 20,000; 200,000$) and bucketing ($\epsilon = 0.95$).

Table 8 reports the results of this new user study, conducted exactly as discussed in Section 4. By comparing the figures reported in Table 3 and 8, we can see that the quality of recommendations as judged by our assessors does not change remarkably. The experiment confirms our hypothesis: *even if some of the lowest-ranked top-5 recommendations computed on the whole RWRs lists are lacking in the set of recommendations generated by using the pruned and approximated index, they are in most cases replaced with queries of similar quality even according to human judgements.*

6. SCALING UP SUGGESTION BUILDING

To further improve query suggestion response time in the case even the pruned and compressed index discussed above does not fit into the main memory of the computer used for generating suggestion, we can exploit caching to improve throughput and scalability. It is in fact worth remarking that while query popularity changes significantly over time, the usage of terms in queries presents a higher temporal locality [2].

As we have discussed in the previous section, our index is accessed by query terms. For each term t occurring in queries of the log, we have a posting list consisting of p pairs of query IDs and (approximated) steady-state probabilities of reaching such queries by performing a RWR from t. At recommendation-generation time, the lists corresponding to each term occurring in the incoming query are retrieved and their probabilities multiplied. Therefore, in order to speed-up the recommendation scoring phase we can adopt a cache to keep in memory a "working set" of "likely-to-be-used" lists. Each entry of the cache stores p bucketed queryIDs, and is accessed by using the associated term as the key. The cache can be managed with a simple "Least Recently Used" (LRU) policy consisting in replacing, when needed, the oldest list in the cache.

Experiments. In order to assess empirically the benefits of adopting such a caching mechanism, we consider two portions of the query logs not used to learn the TQGraph model. We extract from such portions the queries and we sort them by timestamp. From each query we parse the terms and we build the stream of term requests by keeping the order induced by query timestamps. Each time a term t appears in the stream, we check if the cache contains the associated list. If not we count a cache *miss* and we store t and the associated posting list in the cache, possibly evicting another entry according to the LRU policy. Three different values of $p = 5,000, 20,000, 200,000$ are considered in the experiments, while the relative average bits per entry b are set as reported in Table 6. As in the previous experiments the RWRs are computed by setting $\alpha = 0.9$. The number of entries fitting in a cache of s bits is thus given by $s/(p \cdot b)$.

Results. Figure 4 shows the percentage of cache misses measured on our LRU cache over the total number of requests. The first obvious observation is that bigger caches correspond to a smaller number of cache misses. Indeed, the cache miss curve has an asymptotic trend allowing us to estimate the most reasonable cache size for each query log.

For instance, in the case of the MSN query log, when p is equal to 5, 000 and the average bit per entry is 15.81, we are able to obtain a cache miss ratio of 7.21% when the cache has a size of 4GB. On the other hand, by doubling the size of the cache (from 4GB to 8GB) we obtain a miss ratio of 6.20%, i.e., a decrease of just the 1.01%, a small gain if compared to the higher cost of allocating a bigger cache size.

Similar results can be observed in the case of the Yahoo! query log (> 1 billion terms) where almost 90% of the recommendations can be computed (when $p = 5,000$, and $\epsilon = 0.95$.) in memory by using a "small" 8GB cache. These figures strengthen further our proposal that results to be very scalable as it allows the fast generation of recommendations by using an in-memory approach for the vast majority of queries.

7. CONCLUSIONS AND FUTURE WORK

In this paper we shifted the common query-centric perspective over the problem of query recommendation, and by taking a term-centric perspective we achieved two important results. First a novel query recommendation method able to generate relevant query suggestions also for rare or previously unseen queries. Second, a framework to make our emthod extremely efficient and scalable, thanks to the use

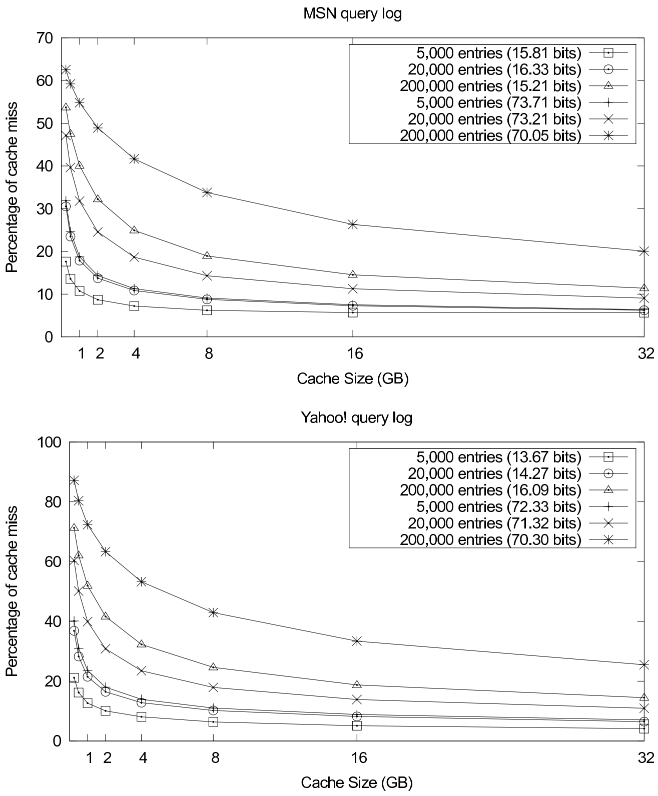

Figure 4: Miss ratio of our cache as a function of its size for different values of p. Results obtained on both the two query logs MSN (top) and Yahoo! (down) are reported.

of a optimized index data structure.

The proposed method is based on a graph model dubbed TQGraph, which has two sets of nodes: *Query* nodes, which are connected among them on the basis of the query reformulations observed in a historical query log, and *Term* nodes, which have only outgoing links pointing at the nodes corresponding to the queries in which the terms occur. Given a TQGraph, the suggestions for a incoming query are generated by performing RWRs starting from the nodes associated with query terms. The method we propose to efficiently compute on-the-fly query suggestions by exploiting a pruned and approximated index represents a more general contribution in the field of efficient computation of approximated random walks with restart, in particular of center-piece subgraphs. An accurate experimental evaluation was conducted to assess effectiveness and efficiency of our proposal. We firstly observed that the distribution of query terms allows our solution to provide suggestions for about the 99% of the queries encountered. The quality of TQGraph-based recommendations were judged higher than QFG-based ones in the user study conducted. Furthermore, we showed that the optimization techniques adopted are very effective, and safeguard the quality of suggestions provided also when aggressive pruning and lossy compression strategies are applied. Finally, a simple caching technique was proposed to enable scalable and fast in-memory generation of TQGraph-based recommendations.

There are various ideas that worth further investigation in order to improve our solution. Firstly, in our model the stationary distributions of the terms in the input query are multiplied together without considering their relative importance. Essentially, we are implicitly assuming that all the terms forming a query are equally important. Obviously, this is not the case. Thus, one could think to weight each vector based on the importance of the corresponding term. For example, we could increase the importance of the terms that better characterize the query at hand. This could be done by resorting to scoring mechanisms that recall known measures like TF/IDF.

Another aspect worth further exploration is the tuning of the length of the pruned lists and of the approximation level. In all the experiments conducted the values of these two parameters were equal for all the terms. Thus, we are again implicitly assuming that all the terms have the same importance. Instead, one should reserve more space for the lists of more important terms.

Acknowledgement

This research has been partially funded by the EU CIP PSP-BPN ASSETS Project, Grant Agreement no. 250527, and InGeoCloudS Project, Contract No 297300.

8. REFERENCES

[1] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proc. KDD'07*.

[2] R. A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *TWEB*, 2(4), 2008.

[3] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *Proc. CIKM'08*.

[4] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. Query reformulation mining: models, patterns, and applications. *Inf. Retr.*, 14(3):257–289, 2011.

[5] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. In *Proc. WWW'09*.

[6] D. Downey, S. Dumais, and E. Horvitz. Heads and tails: studies of web search with common and rare queries. In *Proc. SIGIR'07*.

[7] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *Proc. CIKM'05*.

[8] A. Jain, U. Ozertem, and E. Velipasaoglu. Synthesizing high utility suggestions for rare web search queries. In *Proc. SIGIR'11*.

[9] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proc. WWW'06*.

[10] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proc. CIKM'08*.

[11] F. Silvestri and R. Venturini. VSEncoding: efficient coding and fast decoding of integer lists via dynamic programming. In *Proc. CIKM'10*.

[12] Y. Song and L.-w. He. Optimal rare query suggestion with implicit user feedback. In *Proc. WWW'10*.

[13] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *Proc. WWW'11*.

[14] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *Proc. KDD'06*.

[15] J. Xu and G. Xu. Learning similarity function for rare queries. In *Proc. WSDM'11*.