KB-Enabled Query Recommendation for Long-Tail Queries

Zhipeng Huang, Bogdan Cautis, Reynold Cheng, Yudian Zheng

KB-Enabled Query Recommendation for Long-Tail Queries

Zhipeng Huang, Bogdan Cautis †, Reynold Cheng, Yudian Zheng
The University of Hong Kong, †Huawei Noah's Ark Lab, Hong Kong
{zphuang, ckcheng, ydzheng2}@cs.hku.hk, †cautis.bogdan@huawei.com

ABSTRACT

In recent years, query recommendation algorithms have been designed to provide related queries for search engine users. Most of these solutions, which perform extensive analysis of users' search history (or *query logs*), are largely insufficient for *long-tail queries* that rarely appear in query logs. To handle such queries, we study a new solution, which makes use of a knowledge base (or *KB*), such as YAGO and Freebase. A KB is a rich information source that describes how real-world entities are connected. We extract entities from a query, and use these entities to explore new ones in the KB. Those discovered entities are then used to suggest new queries to the user. As shown in our experiments, our approach provides better recommendation results for long-tail queries than existing solutions.

1. INTRODUCTION

Keyword search, which allows a user to express her query with a number of keywords, has become a fundamental tool in Web search engines. In the last decade, significant improvements have been made to improve the accuracy of keyword search [10]. Recently, the topic of *query recommendation*, which is closely related to keyword search, has attracted a lot of interest from research and industry communities. Besides displaying the "*classic ten blue result links*" from her keyword search, a search engine may suggest alternative formulations of the query, which can be more articulated, focused, and interesting to the user. Providing accurate query recommendations while the user is typing her query, almost instantaneously, can be extremely beneficial, in terms of enhancing the user experience and providing guidance to the retrieval process [11].

Existing work on query recommendation often involves the analysis of *query logs*, which contain a variety of information about previous keyword search activities (e.g., the query contents, the webpages selected by users, and click-through rates) [1, 2, 23]. These query logs are often used to construct a graph, which represents the relationship among queries, terms, and webpages; often, an importance weight for each edge is also computed. For example, in [4], the weight between two query nodes is proportional to the number of times the two queries appear in the same session. While these works have been shown to be useful for query recommendation, most of them do not focus on *long-tail queries*, i.e., queries that rarely appear in query logs. The fact is that long-tail queries exist in abundance, and they cannot be ignored. In our experiments, 32% of 20 million queries appear three or fewer times in an extended query log from a commercial search engine. Because a long-tail query has a low occurrence frequency (and statistical significance), the accuracy of query recommendation solutions can be affected. We have tested the query-flow graph (QFG) method [4] and the term-query-flow graph (TQGRAPH) one [6], two well-known query recommendation algorithms, on long-tail queries. As we will discuss in our experimental results, they are far from accurate, reflecting that there is room for improvement in the recommendation process.

In this paper, we propose an effective recommendation algorithm, called KB-QREC, for long-tail queries. Its main idea is to utilize a *knowledge base* (or *KB*) in the query suggestion process. A KB, such as YAGO [19] and Freebase [5], is a rich information source that describes the intricate relationships among real-world entities. Our solution uses a KB to assist the query recommendation process, alleviating the low-frequency problem of long-tail queries. Let us consider the following query:

q_1: *akira kurosawa influence george lucas*

Through an entity-linking process [22], *Akira_Kurosawa* and *George_Lucas* are identified to be the entities in the KB. Then, KB-QREC explores entities in the KB that are conceptually related to them. For example, *Hidden_Fortress* is a film directed by *Akira_Kurosawa*, and *Star_Wars* is one directed by *George_Lucas*. The relationships among these entities can be expressed by a *meta path* [16]:

$$P_1 : director \xrightarrow{directed} film,$$

where *director*, *film*, and *directed* are the types of nodes and links, respectively, in a KB. Correspondingly, the *meta path instance* *Akira_Kurosawa* $\xrightarrow{directed}$ *Hidden_Fortress* exists in the KB. The meta-path P_1 can also describe the relationship between entities *George_Lucas* and *Star_Wars*.

Next, KB-QREC uses the two discovered entities to suggest for q_1 the following alternative query:

q_2: *hidden fortress star wars comparison*

In other words, KB-QREC extracts the entities of a query, discovers more entities from the underlying KB, and uses those found entities to suggest queries to a user.

Challenges and solutions. In the process illustrated above, we have to address a few challenges. First, given the entities found in a query (e.g., *Akira_Kurosawa* and *George_Lucas* in q_1), how do we find other entities from the KB? As discussed before, *meta*

ISBN 0000.
DOI: 00000

paths, such as P_1, can be used in this process. However, there can be a huge number of meta paths connecting two KB nodes [16], hence we need to identify the useful ones given a large number of entity pairs. Second, after these entities are discovered, we need to perform query recommendation based on them. We examine how KB-QREC can be designed to tackle these challenges, giving fast and accurate query recommendation. We have also tested our solutions on real datasets. We found that KB-QREC can rightly suggest queries that are otherwise absent from other existing approaches.

The paper is organized as follows. We discuss related works in Section 2. Some preliminary information is given in Section 3. We present KB-QREC in Section 4. Our experimental results are discussed in Section 5. We conclude in Section 6.

2. RELATED WORKS

There is a rich body of research on mining the query terms, click-through data, and the logical user sessions in order to extract useful patterns and similarity measures – be it syntactical, semantical, or behavioral – for alternative query formulations in Web search engines. At their core, the techniques boil down to computing similarity between query instances, often using as an intermediary step various graphs involving queries, pages, users, and terms.

The short paper of [25] was among the first to suggest mining from query logs the sequential search behavior, and to combine it with content-based similarity. In [2], the authors introduce the concept of *cover-graph*, a bipartite graph between queries and Web pages, where links indicate corresponding clicks. In [8], another method using search *short cuts* was proposed.

The studies of [4, 6] present similar style graph-based methods, in which the flow patterns are exploited for query recommendation, using the *query-flow graph* and the *term-query-flow graph*, respectively. The former technique builds a graph over queries, in which links model the transition likelihood in query sessions. The latter technique extends this idea by adding to the graph also the query terms, in this way being able to recommend even for queries that may not explicitly appear in the graph (are not "covered"). In both works, the selection of the top-k query recommendations is done by performing random walks in the graph. In [9], the authors rely on a click-through bipartite graph but also consider the context of the query, its immediately preceding queries, in order to better identify suggestions at query time via suffix-trees.

Besides accuracy, two significant challenges for all these works are *efficiency* and *coverage*. Regarding efficiency, naturally, query recommendation in a Web search engine should trigger within *typing latency*, in order to disrupt as little as possible the user experience; one well-accepted upper-bound on the recommendation latency is $100ms$ and this threshold may be hardly met by techniques that must perform random walks in a large graph. This is why a lot of effort has been put into smart indexing and pre-fetching policies, the approximation of random walks, etc. Regarding coverage, which captures how often the recommendation engine can provide meaningful query suggestions, the existing techniques still underperform on the so called *long-tail* queries, those which are not very frequent and have scarce support for recommendations. Indeed, long-tail queries are generally handled poorly by the state-of-the-art approaches, such as the ones of [4, 6], simply because little to no evidence is available in the historical records for them.

Very few works have considered the integration of semantics, in its most common and rich form, the one of a KB, as the means for a deeper understanding of the query intents and of the relationships that may support suggestions. Among them, [21] proposed to enhance the query-flow graph with templates over a hierarchy of entity types (the sort of hierarchy that Wordnet or the *isA* projection of a knowledge base could provide). In [18], the authors considered the mining, ranking, and recommending of so called "entity aspects" (query segments that represent subtopics) in keyword search. Their approach combines several metrics and methods for computing similarity or compatibility, including semantics via word2vec [17] descriptions. Finally, in [7], the authors consider a slightly different query suggestion scenario, in which the focus is on anticipating the user's information need (say, the next query, i.e., a related but not so obvious query suggestion), in the context of a Web page that is currently visited (represents the current query) and of the entities this page may contain (Wikipedia entities); once again, this is done by extending the query-flow graph to an entity-query flow graph.

In our view, these works take only a partial approach in the quest to provide more accurate query suggestions, especially for long-tail queries, based on semantics. They only make use of limited semantic-relatedness measures, such as the entity type hierarchy or word2vec [17]. Our thesis is that better results can be obtained by a broader exploration of the relationships between entities. This can only be done by adopting the meta paths, the direct or composite relationships among entities, as a key ingredient in the process of reasoning for relatedness and of building suggestions. As we show in this paper, this introduces new opportunities, in particular for providing suggestions that are less obvious, but raises also new technical challenges: for instance, the space of meta paths can be very large in general and must be explored and filtered very efficiently in order to support online query suggestions.

The importance of rare queries is emphasized in [11], arguing that search engines are less effective on long-tail queries, and that reformulations are much more common for them.

Finally, entity linking, the problem of identifying entities from a KB in a given piece of text, is also a well-known problem for which most recent efforts focused on how to optimize the latency of generic approaches, which have good precision; see [3] and the references therein. We assume similar overhead in our query processing pipeline as in [3].

3. PRELIMINARIES

In this section, we first revisit query logs in keyword search in Section 3.1, and the well-established notion of query-flow graph in Section 3.2, which conceptually captures the behavior of users when reformulating queries. Then, we introduce necessary terminology and concepts for the knowledge base and the meta paths therein in Section 3.3.

3.1 Query Logs

A typical model for the log of a keyword search engine is a set of records (q_i, u_i, t_i, C_i), where q_i is a query submitted by user u_i at time t_i, and C_i is the set of clicked URLs during this query process.

Following common practice, we partition a query log into task-oriented sessions, where each one is a contiguous sequence of query records from the same user, assuming a fixed maximal separation time t_θ (a typical t_θ value is 30 minutes). Within the same session, we can assume that the user's search intent remains unchanged.

3.2 Query-Flow Graph

One of the most studied directions for the problem of top-k query recommendation we revisit in this paper relies on the extraction of behavioral patterns in query reformulation, from extensive collections of historical search and click records (the query logs). The query-flow graph (QFG in short) [4] is a graph representation of query logs, capturing the "flow" between query units. Intuitively, a QFG is a directed graph of queries, in which an edge (q_i, q_j) with weight w indicates that the query q_j follows query q_i in the same session with probability w in the query log.

More formally, the QFG is defined as a directed graph $G_{qf} = (Q, E, W)$, where Q is the set of nodes, with each node representing a unique query in the log, $E \subseteq Q \times Q$ is the set of edges, and W is a weighting function assigning a weight $w(q_i, q_j)$ to each edge $(q_i, q_j) \in E$. In G_{qf}, two queries q_i and q_j are connected if and only if there exists a session in the query log where q_j follows q_i.

The main application of QFG is to perform query recommendation. Given a graph $G_{qf} = (Q, E, W)$ and a query $q \in Q$, the top-k recommendations for q could be obtained in this graph by some kind of proximity-based top-k node retrieval, be it neighborhood-based (e.g., the queries q' to which q connects with the largest weights $w(q, q')$) or path-based (e.g., by Personalized PageRank w.r.t. node q). No matter what kind of proximity we choose, QFG can only perform well on those popular queries, which appear very frequently in the query log. For those long-tail queries, about which the query log has little information, as shown in our experiments, QFG has very poor recommendation performance. This is why we resort to knowledge bases to address these long-tail queries.

3.3 Knowledge Base and Meta Paths

A knowledge base, or Heterogeneous Information Network (HIN), such as YAGO [15] or DBPedia [24], can be viewed as a set of facts (a.k.a. triples), where each fact describes a relationship between two entities. Different models for knowledge bases have been studied in the literature. One of the most common formalizations of knowledge relies on the *RDF model*, which models triples (s, p, o) to denote a subject, property, and respectively, object. Alternatively, *property graphs* model this type of information by labeled edges and nodes, with labels indicating the edge types and node classes, respectively; additionally, in this model, nodes can carry various property-value pairs.

Independently of syntactic formalization flavors, for the purposes of this work, given a set of entity types (or classes) $\mathcal{L}$ and a set of link types (or relationships) $\mathcal{R}$, we see a knowledge base simply as a directed graph $K = (V_e, E_{ee})$ with an entity type mapping function $\phi : V_e \rightarrow 2^{\mathcal{L}}$ and a relationship mapping function $\psi : E_{ee} \rightarrow \mathcal{R}$. Each node in K represents an *entity* $e \in V_e$, and belongs to some entity types $\phi(v) \subseteq 2^{\mathcal{L}}$; this can be seen as a form of resource typing. Each link $e \in E$ has a relationship label $\psi(e) \in \mathcal{R}$. In a knowledge base K, two entities e_1, e_2 may be connected via multiple edges and paths. Conceptually, each of these paths represents a specific direct or composite relationship between them. We model all these direct and composite relationships by *meta paths* [20]. Specifically, a meta path P in the knowledge base is a sequence of entity types $t_1, \ldots, t_n$ connected by link types $l_1, \ldots, l_{n-1}$, and will be represented as $P = t_1 \xrightarrow{l_1} t_2 \ldots t_{n-1} \xrightarrow{l_{n-1}} t_n$. For example, a meta path $person \xrightarrow{marryTo} person$ represents the direct relationship between entities of the type $person$.

4. METHOD

We now introduce our knowledge-enabled query recommendation approach, denoted in short KB-QREC, which can make use of the rich information in a KB to improve the performance of the query recommendation task for long-tail queries. In Section 4.1, we give an overview to our method, then in Section 4.2 we describe the details for building offline the necessary data structures, from the input query log and the knowledge base. Finally, in Section 4.3 we describe the online phase for recommendation using KB-QREC.

4.1 Overview

Intuitively, KB-QREC is a combination of the query-flow graph G_{qf} with a knowledge base K, the two being bridged by entity-to-query links between entity nodes in K and query nodes in G_{qf}. After we perform entity linking on the queries in G_{qf}, we can further analyze the entities within K. With the rich information in KB, we can better understand the rationale behind the flow of queries.

Formally, the KB-QREC underlying data structure is a quadruple $(G_{qf}, K, t_{\mathcal{EQ}}, \mathcal{P})$, where we have (1) $G_{qf} = (Q, E_{\mathcal{QQ}}, W)$ is a query-flow graph (Section 3.2); (2) $K = (V_{\mathcal{E}}, E_{\mathcal{EE}})$ is a KB (Section 3.3); (3) $t_{\mathcal{EQ}}$ is a transition probability matrix associating to each linked entity-query pair (e, q) a probability $t_{\mathcal{EQ}}(e, q)$ in $(0, 1)$; (4) $\mathcal{P}$ is a set of meta paths over the entity types in K. Specifically, $\mathcal{P}[t]$ is a set of meta paths, each having an associated importance score, for entity type $t \in \mathcal{L}$. We denote by $\mathcal{P}[t][p]$ the weight of meta path p for entity type t.

Intuitively, with the knowledge base K enhancing the original query-flow graph, we can better grasp the behavior of the search engine users, by analyzing the flow among entities. For example, suppose the two queries q_1 *"george lucas"* and q_2 *"star wars"* appear in sequence in the same session, and thus we have $(q_1, q_2) \in E_{\mathcal{QQ}}$. If we can detect the two entities e_1 *George_Lucas* and e_2 *Star_Wars*, we also have $(e_1, q_1), (e_2, q_2) \in t_{\mathcal{EQ}}$. Then we can analyze the flow from e_1 to e_2 in our knowledge base K, in addition to the flow from q_1 to q_2 in the query-flow graph. For example, we can find that in the knowledge base e_1 and e_2 are connected by the meta path $director \xrightarrow{directed} film$, and this is one piece of evidence for the fact that users may tend to search for these two queries jointly. We can exploit such evidence when answering queries w.r.t. directors.

4.2 Offline Steps in KB-QREC

Before detailing how we perform recommendations with KB-QREC, we first describe how to build the necessary data structures from a query log offline. Section 4.2.1 recalls the query-flow graph G_{qf} approach, Section 4.2.2 shows how to construct an entity-query transition matrix $t_{\mathcal{EQ}}$, and Section 4.2.3 shows how to build the meta path collection $\mathcal{P}$.

4.2.1 Building the Query-Flow Graph G_{qf}

We adopt the approach of [4] to build the query flow graph G_{qf}. Specifically, we perform a linear scan on the query log. For each query pair (q, q') that appears within the same session, in this order timewise, we have $(q, q') \in E_{\mathcal{QQ}}$. We set $w(q, q')$ to be $f(q, q')/f(q)$, where $f(q, q')$ is the frequency of co-occurrences of q and q' in sessions, and $f(q)$ is the frequency of q itself.

4.2.2 Computing $t_{\mathcal{EQ}}$

For each query $q \in Q$, we perform entity linking on it, and denote by $\theta(q)$ the set of entities found in q. Note that, by definition, each entity e detected in a query instance has also a corresponding entity node in the knowledge base K. $t_{\mathcal{EQ}}(e, q)$ measures how relevant an entity e is to the query q. It is defined proportional to the frequency of q in the query log divided by the total frequency of all the queries containing e. Formally, $t_{\mathcal{EQ}}(e, q) = \frac{f(q)}{\sum_{q' \mid e \in \theta(q')} f(q')}$. Note that $\sum_q t_{\mathcal{EQ}}(e, q) = 1$ holds for all e.

4.2.3 Constructing the meta path collection $\mathcal{P}$

If we view $\theta(q)$ as a representation of the query q, we can extract the entity-to-entity transitions within sessions. For example, a query *"star wars"* after query *"geoge lucas"* accounts for a transition from entity *Geoge_Lucas* to entity *Star_Wars*. In our knowledge base, these entities have their corresponding nodes connected via multiple paths, and each such path stands for a specific relationship between the two entities. To capture these relationships, we construct a set of outgoing meta paths in $\mathcal{P}$ for each entity type $t \in \mathcal{L}$ appearing in the query log, as follows.

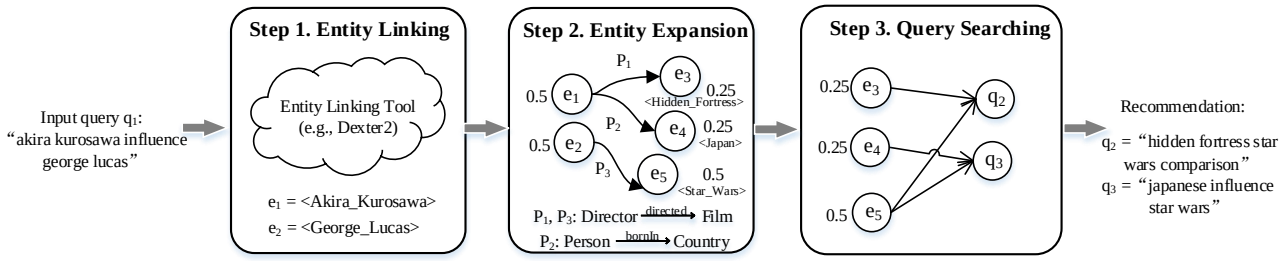

Figure 1: Three steps to generate query recommendations with KB-QREC.

Algorithm 1: KB-QREC recommendation engine

Input: $(G_{qf}, K, t_{\mathcal{EQ}}, \mathcal{P})$, a query q, k
Output: top k recommendation list Rec for q

1 $Rec \leftarrow \emptyset$
2 $\theta(q) \leftarrow entityLinking(K, q)$
3 $RelatedEntities \leftarrow \emptyset$
4 **for** $e \in \theta(q)$ **do**
5 **for** $t \in \phi(e)$ **do**
6 **for** $p \in \mathcal{P}[t]$ **do**
7 $entities \leftarrow getNeighbor(K, e, p)$
8 **for** $e' \in entities$ **do**
9 $RelatedEntities[e'] \leftarrow RelatedEntities[e'] + \frac{1}{|\theta(q)| \cdot |entities|}$
10 $queryWeight \leftarrow \emptyset$
11 **for** $e \in RelatedEntities$ **do**
12 $distribution \leftarrow PPR(G_{qk}, e, RelatedEntities[e])$
13 $queryWeight \leftarrow queryWeight + distribution$
14 $Rec \leftarrow topK(queryWeight)$
15 **return** Rec

As in [7], we denote by $t_{q \to q'}(e \to e')$ the transition probability from entity e to entity e' via a pair of queries q and q'. If $(q, q') \in E_{\mathcal{QQ}}$ and $(e, q), (e'q') \in t_{\mathcal{EQ}}$, then we have $t_{q \to q'}(e \to e') = w(q, q')/(|\theta(q)| \cdot |\theta(q')|)$. Then, the transition probability from entity e to entity e' can be defined as:

$$t_{\mathcal{EE}}(e, e') = 1 - \prod_{(q,q') \in E_{\mathcal{QQ}}} (1 - t_{q \to q'}(e \to e')).$$

So far, we have defined the transition probability among entities derived from a query log. Suppose now we receive a query q containing entity e: if e has already been encountered in the query logs, we can directly use the information on e to perform recommendations for q. For example, we can directly return the queries q' with largest entity-to-query transition probability $t_{\mathcal{EQ}}(e, q')$, or we can perform Personalized PageRank to retrieve queries with high probability. However, this works only if e has been seen in the query log, and this can be rarely the case for long-tail queries.

To address this problem, we want to share the information between the short-tail and the long-tail queries. One solution is to find a meta path in K to represent the relationship between entity pairs. Then, for a new entity e' that has not been encountered in the query log, we can use this meta path to derive related entities in K. We now describe how to select from the knowledge base K the meta paths that will be used in this way for query suggestion.

Given two entities e and e', most often, there can be a large number of distinct paths in K connecting them. Each of these paths potentially represents a relationship between them. However, not all these paths are equally meaningful. Unsurprisingly, a very long path between e and e' may have a "diluted" meaning, as pointed out also in [20]; therefore a natural approach, and the one we follow here, is to select the shortest paths between e and e' to represent their relationships. Then, for each entity type $t \in \phi(e)$, we bookkeep the meta paths in $\mathcal{P}[t]$ and accumulate the weight $t_{\mathcal{EE}}(e, e')$.

4.3 Online steps in KB-QREC

We are now ready to detail in this section how to generate query recommendations for a given query q, using the KB-QREC method. The overview is given in Algorithm 1, and a runtime example is shown in Figure 1. Generally, there are three steps:

Step 1. Entity Linking. (line 2) We firstly need to perform entity linking on the input query q, and get the set of entities $\theta(q)$ contained in it. Note that each of these entities have corresponding nodes in the knowledge base K. For example in Figure 1, given an input query q_1 *"akira kurosawa influence george lucas"*, we perform entity linking on it and get the entities e_1 and e_2.

Step 2. Entity Expansion. (lines 3-9) for each $e \in \theta(q_1)$, we look at its entity types $\phi(e)$ in the knowledge base K. Then, for each $t \in \phi(e)$, we perform a path-constrained random walk [14] on K with each meta path $p \in \mathcal{P}[t]$, and get the related entities for q_1. To each of these related entities, we assign a weight as defined in line 9 of Algorithm 1. For example, e_1 has the type *Director* and *Person* in K. Then, we use the pre-recorded meta path $Director \xrightarrow{directed} Film$ and $Person \xrightarrow{bornIn} Country$, to find related entities e_3 and e_4. Finally, these related entities, e_3 and e_4, are assigned each a weight of $1.0/(2 \times 2) = 0.25$.

Step 3. Query Searching. (lines 10-14) Last but not least, we need to find out the most relevant queries w.r.t. the set of related entities. One commonly-used way is to perform Personalized PageRank (PPR). Specifically, for each related entity e, we perform a PPR on $G_{qf} \cup t_{\mathcal{EQ}}$, starting from e, with initial probability $RelatedEntities[e]$. We sum up the stable probability distribution for each PPR to obtain the aggregated per-query result. The k queries with the largest aggregated probability should be our recommendations. In our example, we perform PPR computations starting from e_3, e_4, and e_5, respectively, with the corresponding probability. Finally, we sum up the probability distribution over queries to get the top-2 recommendations, q_2 and q_3.

Note that the recommendation process only considers the entities $\theta(q)$, without considering the unique information about query q itself. In other words, the recommendation results are the *same* for all queries containing the *same* entities, i.e., $Rec(q) = Rec(q')$ if $\theta(q) = \theta(q')$. While this interpretation already proves to be effective for long-tail queries containing entities, it may also harm the results in certain cases. We intend to study how to combine the KB-QREC method with an existing one, to get a best-of-both-worlds approach, in the future.

5. EXPERIMENTS

We compare the performance of three configurations: the query-flow graph (denoted as QFG), the term-query-flow graph (denoted as TQGRAPH), and the knowledge-enable method (denoted as KB-QREC), focusing on their performance on long-tail queries.

5.1 Implementation

For QFG, we re-implemented the query-flow graph as described in [4]. We adopt the typical recommendation approach by maximum weight, i.e., for an input query q, the query q' with largest value of $w(q, q')$ will be the one recommended. An advantage of

this approach is that it can provide very locally related recommendations with high efficiency.

For TQGRAPH, we directly used a Scala implementation published by the authors of [12]. We used the default parameters, i.e., a restart probability for the random walk $\alpha_{re} = 0.9$ and the convergence distance $\epsilon = 0.005$.

For KB-QREC, we used YAGO [13] as our KB. YAGO is a large-scale knowledge graph derived from Wikipedia, WordNet, and GeoNames. We use its "Core Facts", i.e., YAGO-Core [16], which contains $4M$ facts (edges) of 125 types, over $2.1M$ entities. These entities have $365K$ entity types, organized in a hierarchy with 5 layers. We follow the procedure in Section 4.2.3 to select the meta paths, and use the approach in Section 4.3 to provide recommendations.

5.2 Dataset

We used in all our experiments a well-known public dataset from a major commercial search engine, which consists of web queries collected from $657k$ users over a two months period. This dataset is sorted by anonymous user ID and sequentially arranged, containing $20M$ query instances corresponding to around $9M$ distinct queries. After we sessionized the query log with $\theta_t = 30min$, we obtained a total of $12M$ sessions. As the focus of this paper is not on entity linking, we directly used Dexter2 [22] to tag the entities from queries. After entity linking, we obtained a total of $0.4M$ distinct entities in our dataset.

5.3 Methodology

We adopt the automatic evaluation process described in [21], to assess the performance of the five configurations as predictors of the next query in a session. Basically, we make use of part of the query logs (training data) to predict the user's behavior over a kept-apart query log (the test data). In the test query log, we denote by $q_{i,j}$ the j_{th} query in the session s_i. We assume that $\{q_{i,j} \mid j > 1\}$ is a good recommendations for query $q_{i,1}$ which, as argued in previous literature, is a reasonable assumption for scalable evaluation.

To assess the performance of each method, we simply count their "hits" in the test data, i..e., for each session thereof, how many times one of the recommended top-k queries for $q_{i,1}$ is in $\{q_{i,j} \mid j > 1\}$. While the objective of this evaluation approach may not necessarily be aligned with what a good recommendation may be on particular instances, by being entirely unsupervised and used on a large number of sessions, it becomes a strong indicator of the techniques' performance. For a more complete assessment, we also intend to perform a complementary user study in the future.

In order to test how robust each method is, we used 90%, 50% and 10% of the query logs to train the recommendation systems. We denote them as D_{90}, D_{50}, and D_{10}. Note that the smaller the query log, the less historical information we have on queries. The statistics of these datasets are shown in Table 1. We can see that the number of sessions and the number of distinct queries drops almost linearly with the size of query log, but the number of distinct entities is more stable. This means that we can still rely on the information on entities, even when we have a very small query log.

Table 1: Statistics about the datasets

	$\#sessions$	$\#queries$	$\#entities$
$Dataset$	$12M$	$9.2M$	$0.40M$
D_{90}	$11M$	$8.4M$	$0.39M$
D_{50}	$5.9M$	$4.9M$	$0.30M$
D_{10}	$1.2M$	$1.1M$	$0.13M$

We used the remaining 10% of the query log to generate the test sessions. We first extracted those sessions with at least two queries, and obtained 467473 such sessions. As explained before, we then took the first query of each session as input and the following queries as the ground truth recommendations. Formally, the ground truth for input query $q_{i,1}$ is $\{q_{i,j} \mid j > 1\}$, where $q_{i,j}$ is the j_{th} query appearing in the i_{th} session.

We further extracted the long-tail queries from the test sessions: if the frequency of a query in the whole query log is no more than a threshold θ_f, we refer to it as belonging to the long-tail. Then, we form our test sessions by those whose first query $q_{i,1}$ belongs to the long-tail. By setting θ_f to be 10, 5, and 3, respectively, we have three different testing dataset L_{10}, L_5, and L_3. Note that these three test datasets contain only long-tail queries w.r.t. the frequency threshold θ_f, but they include both entity-bearing queries and queries without entities. As our knowledge-enabled method can only apply to entity-bearing queries, we further filter our all sessions where the first query is not of this kind, i.e., $|\theta(q_{i,1})| = 0$. This leads to the three test datasets L'_{10}, L'_5, and L'_3, which contain only sessions starting by long-tail queries with entities.

We stress here that the feature of containing entities or not is not really affected by the frequency; head, torso, or tail queries alike have a ratio of approximately 60% entity-bearing queries. We measured the following two metrics to evaluate the performance of each configuration:

• $Coverage$. Percentage of input queries that can be answered with at least one recommendation.

• $Precision@5$. Percentage of recommended queries in the top-5 lists that are in the ground truth, as described previously. Formally, $Precision@5 = \frac{\#HIT}{5 \cdot \#query}$, where $\#HIT$ is the number of recommended queries that are part of the ground truth, and $\#query$ is the number of input queries.

5.4 Results and Analysis

Figure 2 presents the coverage results on L'_{10}, L'_5, and L'_3. We can see that (i) all the configurations show a decrease in coverage when we use a smaller query log, but the knowledge-enabled methods remain relatively stable. In particular: KB-QREC still has the same coverage even when we use 10% of query log. (ii) KB-QREC holds a stable coverage on queries with different frequencies, while QFG and TQGRAPH have a lower coverage as the queries have lower frequency. (iii) When we use 10% of query log, KB-QREC has higher coverage than QFG and TQGRAPH.

Figure 3 presents the precision@5 results on L'_{10}, L'_5 and L'_3. We can see that: (i) KB-QREC has the highest precision@5 on all the three datasets, (ii) QFG has extremely low precision@5, because of its low coverage, and (iii) similar to the results of coverage, all the three configurations show a decrease in precison@5 when we use a smaller query log, but KB-QREC remains relatively stable.

If we compare the precision@5 results across the three data configurations, we can see that (i) performance for QFG drops when we test on a longer tail (queries with lower frequency), and (ii) KB-QREC has a rather stable performance even when we test on queries with lower frequency. This is because the KB-QREC method only considers the entities $\theta(q)$ in q. Even when the query q does not appear frequently in the query log, KB-QREC can still make use of the flow recorded for other queries q', containing the same entities. This property makes KB-QREC suitable for long-tail queries, and even for those that were not seen before in the query log.

The extra information in the knowledge base can account for all these improvements on both coverage and precision@5. It is natural that more information leads to better performance. An advantage of our KB-QREC method is that the amount of information in the knowledge base remains the same, no matter how large is the

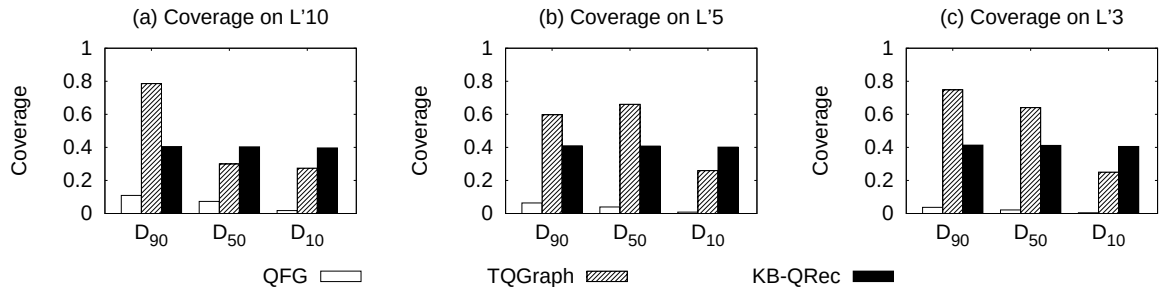

Figure 2: Coverage results on long-tail queries with entities.

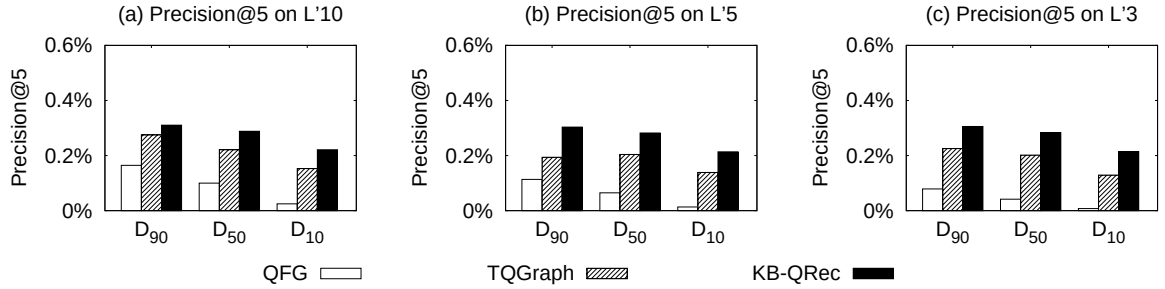

Figure 3: Precision@5 results on long-tail queries with entities.

query log trained upon. This can explain why the performance of KB-QREC remains stable even with a relatively small query log, as when we test on queries with very low frequency.

6. CONCLUSION

In this paper, we propose a knowledge-enabled approach for query recommendation, one of the most visible features in Web search. Its performance for long-tail queries significantly overpasses that of state-of-the-art methods. This is important, as it is well-known that most queries are rare and it is precisely for them that query recommendations are most useful. Using a real-world dataset from a major commercial Web search engine, our automatic evaluation shows that the knowledge-enabled method can bring significant improvements, in terms of both coverage and precision.

Acknowledgements

Zhipeng Huang, Yudian Zheng, and Reynold Cheng were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129). We would like to thank the reviewers for their insightful comments.

7. REFERENCES

[1] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology - EDBT Workshops*, 2004.

[2] R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, 2007.

[3] R. Blanco, G. Ottaviano, and E. Meij. Fast and space-efficient entity linking for queries. In *WSDM*, 2015.

[4] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM*, 2008.

[5] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. ACM, 2008.

[6] F. Bonchi, R. Perego, F. Silvestri, H. Vahabi, and R. Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *SIGIR*, 2012.

[7] I. Bordino, G. D. F. Morales, I. Weber, and F. Bonchi. From machu_picchu to "rafting the urubamba river": anticipating information needs via the entity-query graph. In *WSDM*, 2013.

[8] D. Broccolo, L. Marcon, F. M. Nardini, R. Perego, and F. Silvestri. Generating suggestions for queries in the long tail with an inverted index. *Inf. Process. Manage.*, 2012.

[9] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, 2008.

[10] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines - Information Retrieval in Practice*. Pearson Ed., 2009.

[11] D. Downey, S. T. Dumais, and E. Horvitz. Heads and tails: studies of web search with common and rare queries. In *SIGIR*, 2007.

[12] H. Feild and J. Allan. Task-aware query recommendation. In *SIGIR*, 2013.

[13] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li. Meta structure: Computing relevance in large heterogeneous information networks. In *SIGKDD*, 2016.

[14] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 2010.

[15] F. Mahdisoltani, J. Biega, and F. M. Suchanek. YAGO3: A knowledge base from multilingual Wikipedias. In *CIDR*, 2015.

[16] C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang. Discovering meta-paths in large heterogeneous information networks. In *WWW*, 2015.

[17] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[18] R. Reinanda, E. Meij, and M. de Rijke. Mining, ranking and recommending entity aspects. In *SIGIR*, 2015.

[19] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[20] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB*, 2011.

[21] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *WWW*, 2011.

[22] S. Trani, D. Ceccarelli, C. Lucchese, S. Orlando, and R. Perego. Dexter 2.0: an open source tool for semantically enriching data. In *ISWC*, 2014.

[23] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *WWW*, 2001.

[24] DBpedia 3.7. http://wiki.dbpedia.org/Downloads37.

[25] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW*, 2006.