Entity-Based Query Recommendation for Long-Tail Queries

ZHIPENG HUANG, University of Hong Kong
BOGDAN CAUTIS, University of Paris-Sud, France
REYNOLD CHENG, University of Hong Kong
YUDIAN ZHENG, University of Hong Kong
NIKOS MAMOULIS, University of Ioannina, Greece
JING YAN, University of Hong Kong

Query recommendation, which suggests related queries to search engine users, has attracted a lot of attention in recent years. Most of the existing solutions, which perform analysis of users' search history (or *query logs*), are often insufficient for *long-tail queries* that rarely appear in query logs. To handle such queries, we study the use of *entities* found in queries to provide recommendations. Specifically, we extract entities from a query, and use these entities to explore new ones by consulting an information source. The discovered entities are then used to suggest new queries to the user. In this paper, we examine two information sources: (1) a knowledge base (or *KB*), such as YAGO and Freebase; and (2) a click log, which contains the URLs accessed by a query user. We study how to use these sources to find new entities useful for query recommendation. We further study a hybrid framework that integrates different query recommendation methods effectively. As shown in the experiments, our proposed approaches provide better recommendations than existing solutions for long-tail queries. In addition, our query recommendation process takes less than 100*ms* to complete. Thus, our solution is suitable for providing online query recommendation services for search engines.

ACM Reference Format:
Zhipeng Huang, Bogdan Cautis, Reynold Cheng, Yudian Zheng, Nikos Mamoulis, and Jing Yan. 2018. Entity-Based Query Recommendation for Long-Tail Queries. *ACM Trans. Knowl. Discov. Data.* 1, 1, Article 1 (January 2018), 23 pages. https://doi.org/10.1145/3233186

1 INTRODUCTION

Keyword search, which allows a user to express his/her query with keywords, has become a fundamental tool in Web search engines. In the last decade, significant efforts have been made to improve its accuracy [15]. Recently, the topic of *query recommendation*, which is closely related to keyword search, has attracted a lot of interest from both the research and industry communities. Besides displaying the "*classic ten blue result links*" from his/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 his/her query, almost instantaneously, can be extremely beneficial, in terms of enhancing the user experience and providing guidance to the retrieval process [16].

Authors' addresses: Zhipeng Huang, University of Hong Kong, Hong Kong, zphuang@cs.hku.hk; Bogdan Cautis, University of Paris-Sud, Orsay, France, bogdan.cautis@u-psud.fr; Reynold Cheng, University of Hong Kong, Hong Kong, ckcheng@cs.hku.hk; Yudian Zheng, University of Hong Kong, Hong Kong, ydzheng2@cs.hku.hk; Nikos Mamoulis, University of Ioannina, Ioannina, Greece, nikos@cs.uoi.gr; Jing Yan, University of Hong Kong, Hong Kong, jyan@cs.hku.hk.

1556-4681/2018/1-ART1 $15.00
https://doi.org/10.1145/3233186

Existing works on query recommendation often involve 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, 45]. These query logs are often used to construct a graph representation, which models relationships between queries, terms, webpages, users, etc.; often, an importance weight for each edge is also computed. For example, in [6], 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. However, 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. Due to the low occurrence frequency of long-tail queries, existing query recommendation solutions are ineffective on them. We have tested two well-known query recommendation algorithms [6, 8] on long-tail queries (see Section 9) and found that they are far from accurate, reflecting that there is room for improvement in the recommendation process.

In this paper, we propose a framework for using the entities within a query to provide query recommendations. Our solution consists of three steps: 1) extract entities from the query, 2) use them to find new and related entities with the help of other information sources, and 3) based on the discovered entities, suggest alternative and relevant queries to the user. To illustrate, let us consider the following query.

$$q_1\text{: } \underline{\textit{akira kurosawa}}\ \textit{influence}\ \underline{\textit{george lucas}}$$

In the first step, through an entity-linking process [44], *Akira_Kurosawa* and *George_Lucas* are the entities identified in the query. Then, we need to discover other entities that are conceptually related to them, using other information sources.

We study the use of two common information sources for finding new entities related to a query.

- KB A knowledge base, such as YAGO [41] and Freebase [7], is a rich information source that describes the intricate relationships among real-world entities. We develop a solution, called KB-QRec, to find out useful entity relationships from a KB. The main idea of KB-QRec is to learn the *meta paths* [29], which define semantic relationships between entities. Then, these meta paths are used to discover related entities in a KB. 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*:

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

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*. KB-QRec uses the two discovered entities to suggest for q_1 the following queries:

$$q_2\text{: } \underline{\textit{hidden fortress}}\ \underline{\textit{star wars}}\ \textit{comparison}$$

- Click logs This information source contains URLs that a user clicked after issuing a query. Our intuition is that given two related entities e_1 and e_2, a user tends to browse the same URLs after issuing queries containing e_1 and e_2. For example, if entity *Star_Wars* shares many clicked URLs with *George_Lucas*, then the following query can be recommended:

$$q_3\text{: } \underline{\textit{george lucas}}\ \underline{\textit{star wars}}$$

We have developed a solution called D-QRec to retrieve entity relationships from click logs.

Challenges and solutions. Given the entities found in an input query (e.g., *Akira_Kurosawa* and *George_Lucas* in q_1), how do we find other entities in a KB? As discussed before, *meta 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 [29], hence we need to identify the effective ones and to learn the corresponding weights based on the query log.

As for click logs, how can we assess the degree of relevance between two entities, based on their click information in the query log? Intuitively, the more overlap of clicked URLs we have between the two entities, the more related they are. Using this observation, we define a relevance based score as the conditional transition probability between the entities.

After some related entities have been discovered for the input query, we need to perform fast query recommendation based on them (e.g., q_2 or q_3 in our example). Basically, we achieve this using a cache structure for personalized random walk and also a paralled graph processing platform called PowerWalk. We test our solutions on real-world query datasets, and find that KB-QRec and D-QRec perform better than existing approaches on long-tail queries bearing entities that can be identified.

A problem common to KB-QRec and D-QRec is that they do not work very well for non-long-tail queries, or those whose entities are not identifiable. In view of this issue, we have developed a hybrid framework that integrates different approaches (e.g., QFG [6] and KB-QRec). This solution prioritizes on different methods, such that a method is used only if the ones having higher priority cannot provide sufficient recommendations. Our experiments show that this approach yields better recommendation quality than another scheme that applies linear aggregation of suggestions by the different approaches.

The rest of our paper is organized as follows. We discuss related work in Section 2. Some preliminary information is given in Section 3. We introduce our entity-based query recommendation framework in Section 4. We present KB-QRec in Section 5 and D-QRec in Section 6. We then introduce our integrated solution in Section 7. We describe our efficient implementation in Section 8. Our experimental results are discussed in Section 9. We conclude in Section 10.

2 RELATED WORK

Query auto-completion and query recommendation are two of the most important features in search engines today, and could be seen as facets of the same paradigm: providing accurate query reformulation suggestions on-the-fly, at each keystroke. In query auto-completion, a list of the most relevant continuations to the input (in the typing) query is to be shown for selection. In query recommendation, the suggestion goes one step further by proposing alternative queries, which are not necessarily completions of the input ones. In this section, we review related work on query recommendation. Auto-completion generally relies on prefix-based computations and trie-like data structures; for details, see [3, 10, 38, 39].

2.1 Graph-based Approaches

There is a rich body of research on mining query terms, click-through data, and 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 various graphs involving queries, pages, users, and terms.

Initial approaches rely on clustering for similar queries [1, 45], where proximity depends on the query-click bipartite graph or on query representations aggregating the term-vectors of clicked pages. The short paper of [47] 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 [9], another method using search *short cuts* was proposed.

The studies of [6, 8] present similar 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; this makes recommendation possible even for queries that may not explicitly appear in the graph (i.e., they are not "covered"). In both works, the selection of the top-k query recommendations is done by performing random walks in the graph. For example, given a query which contains t terms, [8] would compute the random walk with restart for each of the term, and sum up the PPR vectors to generate top-k recommended queries. In [22], the authors consider location-aware query recommendation setting. In [11], the authors rely on the so called *click-through bipartite graph* but also consider the context of the query and its immediately preceding queries, in order to better identify suggestions at query time via suffix-trees. Similar in spirit to our D-QRec's way of exploiting query logs, other works, such as [25, 35], exploit implicit feedback with click-through bipartite graphs, yet without taking into account semantic entities.

2.2 Machine Learning-based Approaches

These approaches do not rely on graphs to find clusters of queries or to search for similar ones. For instance, [18] uses association rules to model the relationships between queries. In [40], the authors consider a deep neural model to generate query suggestions, in the style of machine translation approaches, showing that synthesizing suggestions for rare queries is possible. Similar in style, [24] learns to perform automatic query modifications towards suggestions. Semi-supervised learning-to-rank approaches are considered in [32, 37], by training rankers on pairwise query features. In [20], the authors rely on a Markov model (QVMM) and build a suffix-tree to model query sequences.

2.3 Semantics-based Approaches

Very few works have considered the integration of semantics, in its most common and rich form, the one of a knowledge base (KB), as the means for a deeper understanding of the query intents and of the relationships that may support suggestions. Among them, [43] 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 [36], 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 [30] descriptions.

The above works only make use of limited semantic-relatedness measures, such as the entity type hierarchy or word2vec. Our thesis is that better results can be obtained by a broader exploration of the relationships between entities, through the use of other information sources such as KBs and click logs. Using the direct or composite relationships among entities, as a key ingredient in the process of reasoning for relatedness and of building suggestions, introduces new opportunities for providing less obvious suggestions. We address new technical challenges, such as handling the problem of exploring and filtering an extremely large number of meta paths. A preliminary version of our work can be found in [21], where we studied the use of KBs to perform query recommendation. In this paper, we generalize this idea, by proposing a general solution framework that provides query recommendations based on entities found in queries. Under this framework, we also study D-QRec, which uses click information to retrieve entity relationship information. We further propose a hybrid solution, which can integrate different recommendation methods. Our new experiments demonstrate improved results, compared to those presented in [21].

The problem we consider is related, but does not reduce to, entity recommendation in Web search. [33] introduces the scenario of entity recommendations for Web queries, aiming at judging what entities is the input query about (a kind of linking to implicit entities). They propose a technique having at its core a tripartite-graph, linking queries to URLs or entities. In [4], the entity recommendation task is solved via a learning-to-rank approach, using diverse features such as those from textual corpora (e.g., entity co-occurrence), graph-theoretical ones from semantic triples, etc. While not focusing on how to recommended queries to users, the techniques of these two studies could be seen as possible starting points for other methods covering the first two steps of our proposed framework, complementary to the two methods we consider.

Finally, entity linking, i.e., 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 [5] and the references therein. We assume similar overhead in our query processing pipeline as in [5].

2.4 Long-tail Query Recommendation

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. This is why, in many of the aforementioned works, 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, existing techniques still underperform on the so called *long-tail* queries, 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 [6, 8], simply because little to no evidence is available in the historical records for them.

The importance of rare queries is emphasized in [16], arguing that search engines are less effective on long-tail queries, and that reformulations are much more common for them.

3 PRELIMINARIES

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 for q_i, before issuing another query.

Following common practice, we can partition a query log into task-oriented sessions, where each session 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, even though he may do several query reformulations.

3.2 Query-flow graph

One of the most studied directions for the problem of top-k query recommendation we consider 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) [6] is a graph representation of query logs, capturing the "flow" between query units.

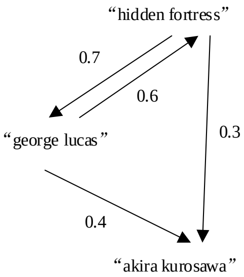

Fig. 1. Illustrating a toy QFG.

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.

For example, Figure 1 illustrates a toy QFG with three queries. Assuming that the input query $q =$"*hidden fortress*", the top recommendation according to this QFG should be "*george lucas*" as it is the out-neighbor with the maximum weight.

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 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 long-tail queries.

3.3 Knowledge base and meta paths

A knowledge base, or Heterogeneous Information Network (HIN) [13], such as YAGO [28] or DBPedia [46], 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*[1], 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 $K = (V_{\mathcal{E}}, E_{\mathcal{E}\mathcal{E}})$ with an entity type mapping function $\phi : V_{\mathcal{E}} \rightarrow 2^{\mathcal{L}}$ and a relationship mapping function $\psi : E_{\mathcal{E}\mathcal{E}} \rightarrow \mathcal{R}$. Each node in K represents an *entity* $e \in V_{\mathcal{E}}$, and belongs to a set of entity types $\phi(v) \subseteq 2^{\mathcal{L}}$; this can be seen as a form of resource typing. Each link

[1] https://www.w3.org/RDF

Table 1. List of symbols.

Symbol	Description
q	a query
Q	the set of queries in our query log
e	an entity
$\theta(q)$	the set of entities within query q
G_{qf}	a QFG
$G_{\mathcal{E}Q}$	the bipartite graph of entities and queries
$G_{\mathcal{D}\mathcal{E}}$	the bipartite graph of documents and entities
$Rec(q)$	the recommended queries for q

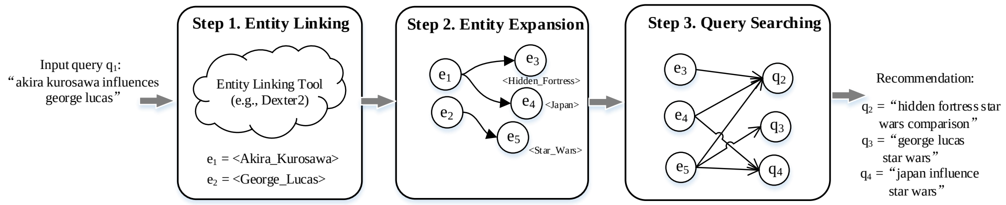

Fig. 2. Example for the three steps in our entity-based query recommendation framework.

$e \in E_{\mathcal{E}\mathcal{E}}$ has a relationship label $\psi(e) \in \mathcal{R}$. For example, entity e ="*George_Lucas*" may have two entity types t_1 =*film maker* and t_2 =*producer*, so $\phi(e) = \{t_1, t_2\}$.

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* [42]. 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 can be represented as follows:

$$P = t_1 \xrightarrow{l_1} t_2 \ldots t_{n-1} \xrightarrow{l_{n-1}} t_n.$$

For example, a meta path $person \xrightarrow{marryTo} person$ represents the direct relationship between entities of the type *person*.

4 ENTITY-BASED QUERY RECOMMENDATION FRAMEWORK

In this section, we give an overview of our framework for entity-based query recommendation. The most commonly used notations are summarized in Table 1.

Given a query q, the framework follows three main steps to generate recommendations for q, as illustrated by our running example in Figure 2:

Step 1 – entity linking. To extract entities from the input query q, we first need to perform entity linking on q, and get the set of entities $\theta(q)$ contained in it. For example in Figure 2, given an input query q_1 = "*akira kurosawa influences george lucas*", we perform entity linking on it and get the entities e_1 and e_2.

After performing entity linking on all the queries in the query log, we can obtain a bipartite graph $G_{\mathcal{E}Q}$ of entities and queries, with each edge (e_i, q_j) meaning that $e_i \in \theta(q_j)$. We further associate to each linked entity-query pair (e, q) a probability $t_{\mathcal{E}Q}(e, q)$ in $(0, 1)$, which is proportional to the frequency of q in the query log divided by the total frequency of all the queries containing e.

Algorithm 1: Computing $t_{\mathcal{E}Q}$

```
  Input: query-flow graph $G_{qf}$, knowledge base $K$
  Output: $t_{\mathcal{E}Q}$
1 for $q \in Q$ do
2     $\theta(q) \leftarrow entityLinking(K, q)$
3 for $q \in Q$ do
4     for $e \in \theta(q)$ do
5         $t_{\mathcal{E}Q}(e, q) = f(q) / \sum_{q' \mid e \in \theta(q')} f(q')$
6 return $t_{\mathcal{E}Q}$
```

Algorithm 1 outlines how we compute $t_{\mathcal{E}Q}$. For each query $q \in Q$, we do entity linking on it (line 2), 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. Then we compute $t_{\mathcal{E}Q}(e, q)$ by our definition (lines 3-5).

As our work does not focus on the entity linking problem, we assume here an off-the-shelf entity linking tool is used, as a black box (e.g., Dexter 2.0 [44]). Although any modern entity linking tool can do the job, we stress that the quality of this step directly impacts the performance of our entity-based recommendations.

Step 2 – entity expansion. Once the entities $\theta(q)$ have been identified, our next step is to expand this semantic footprint of the query and find other related entities. This is where we resort to additional information sources. While various alternatives may be envisioned, in this paper, we consider two such information sources, whose role is to capture relevance between entities. In Section 5, we present an entity-based query recommendation method, KB-QRec, which finds related entities in a *knowledge base*. In Section 6, we then introduce a second method, called D-QRec, which relies on query logs and click information to discover click-induced relevance between entities.

Step 3 – query searching. Last but not least, we need to figure out the most relevant queries w.r.t. the set of related entities obtained at the previous step. One natural approach is to rely on Personalized PageRank (PPR) computations from these entities, in an *entity-query* graph. Specifically, for each related entity e, we can perform a PPR w.r.t. e on $G_{qf} \cup G_{\mathcal{E}Q}$, where G_{qf} is a query-flow graph, as mentioned in Section 3.2, and $G_{\mathcal{E}Q}$ is the bipartite graph mentioned in Step 1. Then, we can sum up the PPR vectors to obtain the aggregated per-query result, and the k queries with the largest aggregated PPR probabilities should be our recommendations.

In our example in Figure 2, we perform PPR computations starting from e_3, e_4, and e_5, respectively. Finally, we sum up the PPR vectors over queries to get the top-3 recommendations, i.e., $q_2 - q_4$.

Note that our entity-based query recommendation framework only considers the entities $\theta(q)$, without taking into account the context of these entities. In other words, the recommendation results are the *same* for all queries containing the *same* entities, i.e., $Rec(q_i) = Rec(q_j)$ if $\theta(q_i) = \theta(q_j)$. While this interpretation already proves to be effective for long-tail queries containing entities, it may also harm the results in certain cases. We thus advocate for a best-of-both-worlds approach in Section 7, where we consider combining our entity-based methods with other techniques, which do take into account the query context, such as QFG or TQGraph.

5 KB-QREC: KNOWLEDGE-AWARE ENTITY-BASED QUERY RECOMMENDATION

In this section, we introduce our knowledge-aware query recommendation approach, denoted in short as KB-QREC, which makes use of the rich information in a knowledge base to model the relevance among entities.

We first give an overview of KB-QREC in Section 5.1, then in Section 5.2 we describe the details for building the necessary data structure offline, from the input query logs and the knowledge base. Finally, in Section 5.3 we describe how we find related entities at query time, using KB-QREC.

5.1 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 the knowledge base, we can better understand the rationale behind the flow of queries.

Formally, the data structure underlying KB-QREC is a quadruple $(G_{qf}, K, G_{\mathcal{E}Q}, \mathcal{P})$, where:

- $G_{qf} = (Q, E_{QQ}, W)$ is a query-flow graph as defined in Section 3.2,
- $K = (V_{\mathcal{E}}, E_{\mathcal{E}\mathcal{E}})$ is a KB, as defined in Section 3.3,
- $G_{\mathcal{E}Q}$ is the bipartite graph of entities and queries as mentioned in Section 4,
- $\mathcal{P}$ is a set of meta paths over the entity types in K. Specifically, $\mathcal{P}[t]$ is the set of meta paths starting with entity type $t \in \mathcal{L}$, each having an associated importance score. 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 $(q_1, q_2) \in E_{QQ}$. If we can detect the two entities e_1 = "*George_Lucas*" and e_2 = "*Star_Wars*", we can also 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 then exploit such evidence when answering other queries referring to directors.

5.2 Offline steps in KB-QREC

Before detailing how we perform recommendation with KB-QREC, we first describe how to build the necessary data structures from a query log offline. Section 5.2.1 reviews the query-flow graph G_{qf} approach, and Section 5.2.2 shows how to build the meta path collection $\mathcal{P}$.

5.2.1 Building the Query-Flow Graph G_{qf}. We adopt the approach of [6] to build the query flow graph G_{qf}. Specifically, we perform a linear scan over the query log. For each query pair (q, q') that appear in the same session, in this temporal order, we have $(q, q') \in E_{QQ}$. We set $w(q, q')$ as the conditional transition probability from q to q', i.e. $f(q, q')/f(q)$ 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.

5.2.2 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 "george lucas" accounts for a transition from entity *George_Lucas* to entity *Star_Wars*. In our KB, these entities have their corresponding nodes connected via multiple paths, and each path stands for a specific relationship between the two entities. To capture these relationships, we build a set of outgoing meta paths in $\mathcal{P}$ for each entity type $t \in \mathcal{L}$ appearing in the query log, as follows.

Suppose $(q, q') \in E_{QQ}$, and $w(q, q')$ is the transition probability from q to q'. If we assume each entity pair among $\theta(q)$ and $\theta(q')$ has equal contribution towards the transition probability $w(q, q')$, then we have the contribution of each entity pair as:

$$t_{q \to q'}(e \to e') = \frac{w(q, q')}{(|\theta(q)| \cdot |\theta(q')|)}, \forall e \in \theta(q), \text{and } e' \in \theta(q').$$

Note that the following equation holds for each query q:

$$\sum_{(q,q') \in E_{QQ}} \sum_{e \in \theta(q)} \sum_{e' \in theta(q')} t_{q \to q'}(e \to e') = 1, \forall q \in Q.$$

Then, the transition probability from entity e to entity e' can be defined as:

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

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 the largest entity-to-query transition probability $t_{\mathcal{E}\mathcal{Q}}(e, q')$, 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, our intuition is 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 [42]; therefore a natural approach, and the one we follow here, is to select the shortest paths between e and e' to represent their relationships.

Algorithm 2 details the steps in the computation of $\mathcal{P}$. First, we compute all the entity-to-entity transition probabilities according to Equation 1 (lines 1-4). Then, for each pair of entities (e, e') with non-zero transition probability, we retrieve the shortest paths between e and e' from K (line 6), and transform them into the corresponding meta paths by linking the types instead of the actual entities (line 7). Finally, for each entity type $t \in \phi(e)$, we bookkeep the meta path in $\mathcal{P}[t]$ and accumulate the weight (line 9).

5.3 Finding related entities using KB-QRec

We are now ready to detail how to find related entities based on the entity linking result $\theta(q)$ using KB-QRec.

For each $e \in \theta(q)$, we obtain its entity types $\phi(e)$ in the knowledge base K. Then, for each entity type $t \in \phi(e)$, we perform a path-constrained random walk (PCRW) [26] on K, with each type-related meta path $p \in \mathcal{P}[t]$, and get the related entities for e w.r.t. meta path p. To each of

Algorithm 2: Computing $\mathcal{P}$

```
Input: query-flow graph $G_{qf}$,knowledge base $K$, entity transition probability $t_{\mathcal{E}\mathcal{E}}$
Output: $\mathcal{P}$
for $(q, q') \in E_{QQ}$ do
    for $e \in \theta(q)$ do
        for $e' \in \theta(q')$ do
            Compute $t_{\mathcal{E}\mathcal{E}}(e, e')$ using Equation 1
for $(e, e') \in t_{\mathcal{E}\mathcal{E}}$ do
    $path \leftarrow getShortestPath(K, e, e')$
    $metapath \leftarrow getMetaPath(K, path)$
    for $t \in \phi(e)$ do
        $\mathcal{P}[t][metapath] \leftarrow \mathcal{P}[t][metapath] + t_{\mathcal{E}\mathcal{E}}(e, e')/|\phi(e)|$
return $\mathcal{P}$
```

these related entities e', we assign a weight as follows:

$$w_{e'} = \frac{1}{|\theta(q)|} \sum_{e \in \theta(q)} \sum_{t \in \phi(e)} \sum_{p \in \mathcal{P}[t]} \mathcal{P}[t][p] \cdot PCRW(e, e' \mid p),$$

where $\mathcal{P}[t][p]$ is the weight we get for meta path p during the offline phase, and $PCRW(e, e' \mid p)$ is the PCRW value from e to e' w.r.t. meta path p.

For example, in Figure 2, e_1 has the types <*Director*> and <*Person*> in K. Suppose we learned the following meta path with weight $\frac{1}{2}$:

$$Director \xrightarrow{directed} Film.$$

Then, we can use this pre-recorded meta path to find entities related to e_1. Finally, the related entity, e_3 is assigned a weight of $\frac{1}{2} \cdot 0.5 = 0.25$ (supposing 0.5 is the PCRW value from e_1 to e_3).

After obtaining the related entities, KB-QRec goes to Step 3 (query searching), as described in Section 4. Basically, for each of the related entities e', we do a PPR on $G_{qf} \cup G_{\mathcal{E}Q}$, with importance weight $w_{e'}$. 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-3 recommendations, $q_2 - q_4$.

6 D-QREC: CLICK-AWARE ENTITY-BASED QUERY RECOMMENDATION

In this section, we introduce a second entity-based query recommendation method, named D-QRec which makes use of the click information from query logs to model the relevance between entities. Given a query log, in the offline steps, we can build a bipartite graph consisting of entity nodes and document nodes (or URLs), denoted the Document-Entity Graph (DEG in short). Formally, the DEG is a graph $G_{\mathcal{D}\mathcal{E}} = (V, E)$, where:

- The node set V consists of entity nodes and document nodes, i.e., $V = V_{\mathcal{E}} \cup V_{\mathcal{D}}$.
- The edge set E consists of entity-to-document edges and document-to-entity edges, i.e., $E = E_{\mathcal{D}\mathcal{E}} \cup E_{\mathcal{E}\mathcal{D}}$. If we have $e \in \theta(q)$, and document d is clicked when the query q was issued, we have $(e, d) \in E_{\mathcal{E}\mathcal{D}}$ and $(d, e) \in E_{\mathcal{D}\mathcal{E}}$.

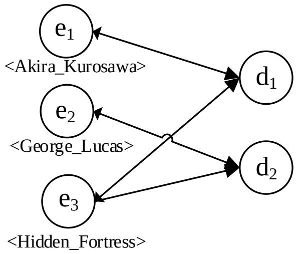

Fig. 3. Illustrating the Document-Entity Graph (DEG).

Figure 3 illustrates a simple DEG, with three entity nodes and two document ones. For example, there is an edge between entity e_1 and document d_1 because we recorded in the logs a query containing e_1 and leading to a click on document d_1.

We assign weights to the DEG edges, proportional to the frequency of the co-occurrences of the entity-document pair, as follows:

$$w(e, d) = \frac{\sum_{e \in \theta(q), d \in C(q)} f(q)}{\sum_{e \in \theta(q)} f(q)}$$

$$w(d, e) = \frac{\sum_{e \in \theta(q), d \in C(q)} f(q)}{\sum_{d \in C(q)} f(q)}$$

where $f(q)$ is the frequency of query q.

6.1 Finding related entities using D-QRec

Suppose q is the query issued by the user, and $\theta(q)$ is the set of entities that we obtain in Step 1 (entity linking). As the online step to retrieve related entities, we can perform Personalized PageRank (PPR) from each $e \in \theta(q)$ over the DEG, and get the PPR vectors over entities. Then, the entities with the largest aggregated PPR values are passed to the final Step 3, to find related queries.

For example, in Figure 3, assuming that $\theta(q) = \{e_1, e_2\}$, we can perform PPR from e_1 and e_2, respectively, and get the summation of the PPR vectors, leading to entity e_3 as the retrieved one.

7 A HYBRID RECOMMENDATION SYSTEM

By design, both our KB-QRec and D-QRec methods exploit at query time only the entities within the input query, ignoring the remaining (textual) content that may accompany them. Therefore, we can potentially increase their effectiveness if we use them as parts of a more general *hybrid framework*, which combines KB-QRec and D-QRec with other methods that focus on the so-called flow and treat each query or term therein as a graph node.

In general, a query recommendation method M can be viewed as a mapping

$$\mathcal{M} : (QL, q, q') \rightarrow s,$$

where QL is a query log, q is a query issued by a user, q' is a candidate query to be recommended for q, and s is a score assigned by the system to q' w.r.t. q. The top-k recommendation for q consists of the k candidate queries q' with the largest scores.

Given multiple recommendation methods, we can simply combine them into a hybrid recommender system $\mathcal{S}$ by linearly aggregating their results:

$$\mathcal{S}(QL, q, q') = \sum_i w_i \times \mathcal{M}_i(QL, q, q') \tag{2}$$

where w_i is the weight of the i-th method, such that $\sum_i w_i = 1$.

However, there are several issues with this linear combination approach. On one hand, it is not clear how to set the weights. One possible solution may be to use some query history as a validation set to tune the optimal weights but, as observed in our experiments, the performance of the linear hybrid method does not vary much w.r.t. the weights, if the same values are used for all queries. On the other hand, due to the different nature of the various recommendation methods, the scores they generate may have different scale. For example, we observe that D-QRec gives very small recommendation scores to the top-k recommendation candidates while QFG gives significantly larger ones. Hence, it is not an easy task to combine them linearly.

Table 2. An example illustrating our hybrid method

input	M_1	M_2
q_1	R_{11}	$\emptyset$
q_2	R_{21}	R_{22}
q_3	$\emptyset$	R_{32}

To avoid this pitfall, as a better hybrid approach, we propose a priority-based method. More precisely, suppose that we need to combine n recommendation methods, we can assign a different priority to each of them. If the method with the highest priority M_1 can provide enough recommendations for the input query q, we just take them as the system's output. Otherwise, we check whether the method with the second highest priority can provide recommendations for q, and so on. In this way, a recommendation method is only considered if the higher-priority ones cannot provide sufficient recommendations. In Section 9, we compare the performances of hybrid systems following different prioritization schemes.

Consider the toy example in Table 2. We can see that model M_1 can provide a recommendation for two input queries q_1 and q_2. Under our prioritized hybrid method, the final recommendation for q_1 (or q_2) would be just R_{11} (or R_{21}). For q_3, as the model M_1 with top priority cannot provide recommendations, we would consider the second model, and output R_{32} as our final recommendation.

8 EFFICIENT IMPLEMENTATION OF PPR

Query recommendation requires responses within typing latency, typically less than $100ms$. Recall that our recommendation methods require PCRW (in the KB-aware method) and PPR computations (in the click-aware method) for Step 2, followed by other PPR computations in Step 3. The running time of these steps would grow as the size of query logs becomes larger. Therefore, a key component in our framework is the efficient implementation of PageRank-like computations.

8.1 PPR cache

One straightforward technique would be to maintain a PPR cache, which basically saves the PPR result for an entity node after a query containing it was issued, and uses the result directly when this entity is queried again. In our Step 3, we can first check whether the related entities are already in the cache. If so, we can directly aggregate their PPR vectors and return the recommended queries.

Table 3 shows an example of such a PPR cache. Suppose in Step 3, we need to perform two PPRs, one for <Akira_Kurosawa> with weight 0.7 the other for <George_Lucas> with weight 0.3. Suppose

Table 3. Illustrating PPR Cache

entity	PPR vector over queries
<Akira_Kurosawa>	"akira kurosawa movie": 0.5 "akira kurosawa george lucas": 0.5
<George_Lucas>	"akira kurosawa george lucas": 0.4 "george lucas star wars": 0.6

the PPR vectors for both entities are already in the cache. Then, we only need to aggregate the PPR results with a linear combination, and get the aggregated PPR vectors over queries: "akira kurosawa george lucas": 0.47, "akira kurosawa movie": 0.35, "george lucas star wars": 0.18.

Although such a PPR cache can be efficient, it requires a lot of space for storing the PPR vectors, which in a real-world scenario dealing with many queries and large logs seems unaffordable. This is why we consider approximate PPR computation solutions, as discussed next.

8.2 PowerWalk

PowerWalk [27] is the state-of-art distributed framework for efficient PPR computation, which strikes a balance between offline indexing and online querying.

8.2.1 Offline indexing. During the offline indexing phase, we use this Monte-Carlo method to compute an approximate PPR vector for each entity node and each query node, in a *PPR index*. Note that the accuracy of the PPR index can be adjusted depending on the available main memory. The offline index is built on the VENUS system [12], which implements a disk-based graph processing approach.

8.2.2 Online querying. For the online querying phase, we can use the vertex-centric decomposition algorithm (VERD) [27] to compute PPR vectors based on the index. The basic idea of VERD is that the PPR vector of a node can be approximated from the PPR index values of its neighbors. The VERD algorithm is built on PowerGraph [19], a distributed in-memory graph engine.

9 EXPERIMENTS

We compare the performances of four query recommendation methods: the query-flow graph (denoted QFG) [6], the term-query-flow graph (denoted TQGraph) [8], our knowledge-aware method (KB-QRec), and our click-aware method (D-QRec). In addition, we present the performance of our priority-based hybrid system, combining TQGraph, KB-QRec and D-QRec by different priority settings.[2]

9.1 Implementation

For QFG, we created the query-flow graph as described in [6]. 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 locally related recommendations efficiently.

For TQGraph, we directly used a Scala implementation published by the authors of [17]. We used their default parameters.

For KB-QRec, we used YAGO [41] 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 [23], which contains $4M$ facts (edges) of 125 types, over $2.1M$ entities. These entities have $365K$ entity types,

[2] We cannot compare with the system of [43], as source code is not available.

organized in a hierarchy with 5 layers. We follow the procedure in Section 5.2.2 to select the meta paths, and use the approach in Section 5.3 to provide recommendations.

Regarding the design of the hybrid system, we also compare our priority-based approach with a linear aggregation one. We test by prioritizing over three single query recommendation methods: TQGraph (short as T), KB-QRec (short as K), and D-QRec (short as D).

9.2 Dataset

In all our experiments, we used 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 in 2006. This dataset is sorted by anonymous user Id, 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 [44] to tag the entities from queries. After entity linking, we obtained a total of $0.4M$ distinct entities in our dataset.

9.3 Automatic evaluation

9.3.1 Methodology. We adopt the automatic evaluation process described in [43], 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 segment of 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 recommendation for query $q_{i,1}$ which, as argued in previous literature, is a reasonable assumption for practical evaluation in a broader scope.

To assess the performance of each method, we simply count its "hits" in the test data, i.e., for each session, 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 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. However, for a more complete assessment, we will also show the results of a complementary user study later on.

9.3.2 Experimental setup. 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 4. 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 4. 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, after training, to generate the test sessions. We first extracted the 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

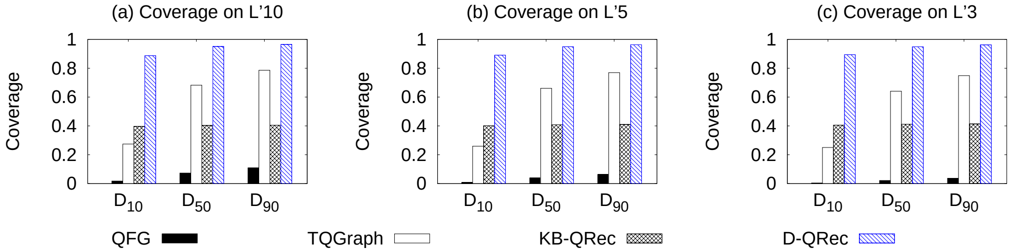

Fig. 4. Coverage results for single methods on long-tail queries with entities.

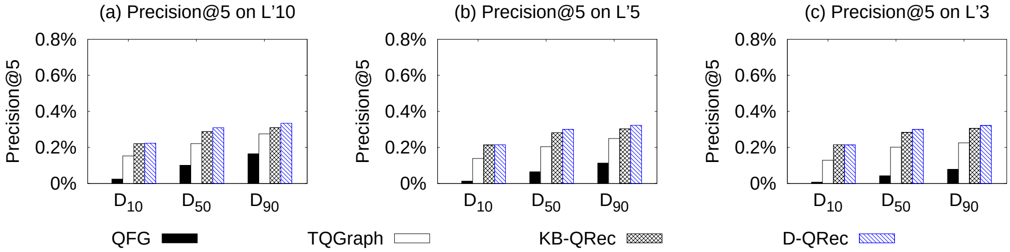

Fig. 5. Precision@5 results for single methods on long-tail queries with entities.

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.

In addition, we extracted the long-tail queries from the test sessions: if the frequency of a query in the whole query log does not exceed a threshold θ_f, we refer to it as belonging to the long-tail class. 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 test datasets 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 for which the evaluated method provides at least one recommendation.
- *Precision@5*. Percentage of ground truth queries that appear in the top-5 recommendation list of the method. Formally, $precision@5 = \frac{\#HIT}{5 \cdot \#query}$, where $\#HIT$ is the total number of recommended queries that are part of the ground truth, and $\#query$ is the number of input queries.

9.3.3 Results of the four methods for long-tail entity-bearing queries. Figure 4 presents the coverage results for four independent methods. We can see that (i) D-QRec has the highest coverage (more than 90%), even for the smallest query log D_{10}, (ii) KB-QRec and TQGraph have relatively high coverage, as TQGraph is also proposed for long-tail queries. (iii) QFG has extremely low

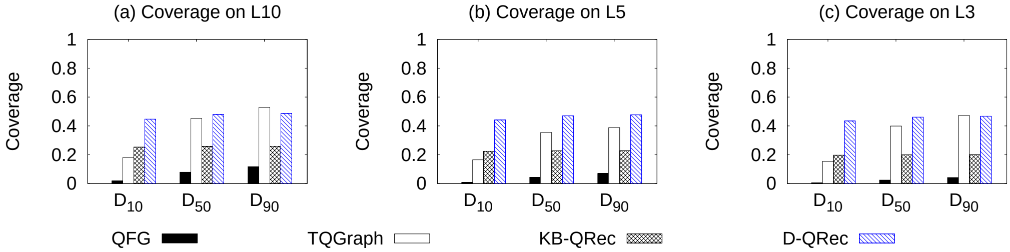

Fig. 6. Coverage results for single methods on general long-tail queries.

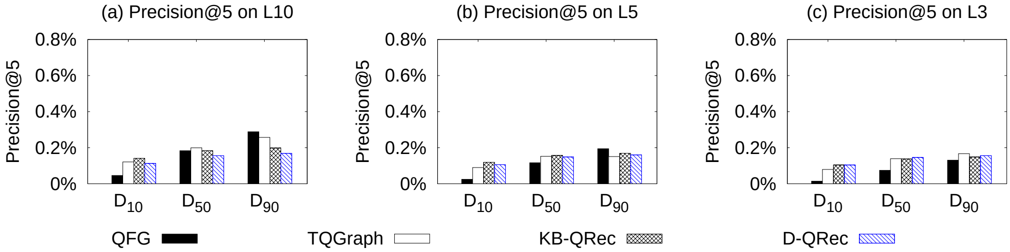

Fig. 7. Precision@5 results for single methods on general long-tail queries.

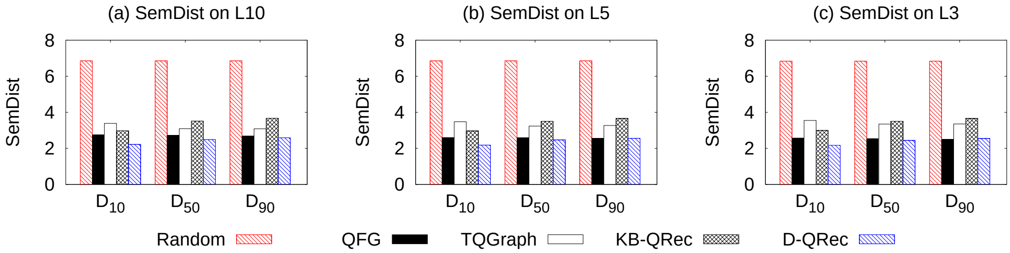

Fig. 8. SemDist results for single methods on general long-tail queries.

coverage compared to the other three methods, (iv) TQGraph and QFG have lower coverage when we use a smaller query log (from D_{90} to D_{10}), while D-QRec and KB-QRec have stable coverage. This is because when we are using a smaller query log, there is a higher chance that we have not seen the query or the terms therein in the logs, making QFG and TQGraph incapable of giving any recommendation.

Figure 5 presents the precision@5 results for the four methods. We can see that (i) D-QRec has the highest precision@5, (ii) QFG has the lowest precision@5, (iii) KB-QRec does better than TQGraph, even though it has lower coverage, and (iv) all the four methods have higher precision@5 when we use a larger query log (from D_{10} to D_{90}).

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

9.3.4 Results of the four methods for general long-tail queries. Recall that the previous results show that the two proposed methods can yield better recommendations for long-tail queries with at least one entity. Now, for a complete assessment, we show the results for general long-tail queries, including those in which we cannot find entities.

Figure 6 presents the coverage results. We can see that the results are quite similar to those in Figure 4, with some notable differences. Compared with the results in Figure 4, QFG has almost the same coverage, while the other three methods have slightly worse performance. For TQGraph, this is because a non-entity-bearing query often has a strange format (a URL or just messy code) thus it is more difficult for recommendation. For our two entity-based methods, this is because almost half of the queries are without entities, and our proposed methods cannot handled these queries properly.

Figure 7 presents the precision@5 results. With respect to the results of Figure 5, the notable differences are as follows. First, QFG has slightly better precision@5, while TQGraph has almost the same precision@5. Our entity-based methods (KB-QRecand D-QRec) have slightly worse performance, as expected, for the same reasons as before.

9.3.5 Semantic quality of the non-hit recommendations. The values of precision@5 for all four recommendation methods we test are quite low, because precision@5 only consider the recommended queries that strictly hit the ground truth. However, the non-hit recommended queries may still be useful, as long as they bear a semantic similarity to the ground truth. For example, assuming that the ground truth query is "booking restaurant chicago", a recommendation "dinner booking chicago" can be considered useful, as there is a clearly close semantic meaning.

To evaluate the semantic distance between the ground truth query and the recommended queries, we adopt the popular NLP technique word2vec [31], which can represent the words in a vector space, while preserving their semantic proximity. More precisely, we directly use the pre-trained word vectors provided by GloVe [34], which use 300 dimensions trained over $6B$ tokens. For a query q with ground truth set $g(q)$, supposing a recommended query is q', then we define the semantic relevance of q' w.r.t. the ground truth as the minimal distance to a query $q^* \in g(q)$. Formally,

$$SemDist(q') = min_{q^* \in g(q)} dist(q', q^*),$$

where $dist()$ is the Euclidean distance between two vectors.

Figure 8 shows the average SemDist of the top-5 recommendation for the four independent methods and, for comparison, for a Random baseline, which recommends queries uniformly at random. We can see that the four query recommendation methods have similar SemDist, while Random has a relatively large SemDist. This means that the recommended queries given by the four methods are generally useful. The SemDist measure remains stable w.r.t. the size of the query log and the frequency of the test queries.

9.3.6 Diversity. We also conducted an analysis of the diversity in the recommended queries. We compare the difference between the *successful recommendations* (log hits) and the original input queries.

We observe that a large portion of the successful recommendations are rather simple reformulations of the input query q. For an extreme example, a successful recommendation for query "akira kurosawa" is "Akira Kurosawa". Obviously, such a recommendation provides little extra information to the user. A better recommendation would be "akira kurosawa hidden fortress", which carries extra information. Inspired by this observation, we evaluate the quality of each method's hits, by looking at the difference between the recommended query and the input query. We use the average edit distance (short as *AED*), larger AEDs meaning more diverse recommendations.

We only report here the results for the datasets L_5' with D_{50}, as the ones on the other datasets are similar. As shown in Table 5, the methods QFG and TQGraph tend to recommend queries which are very similar to the original one. Compared with them, KB-QRec and D-QRec have a more diverse recommendation result, because they do not consider the text of the queries and reach diverse recommendations by means of other entities.

Table 5. Results of Average Edit Distance (*AED*)

	QFG	TQGraph	KB-QRec	D-QRec
AED	4.267	3.299	11.631	11.128

9.4 A user study

Table 6. Results of the user study

Method	Coverage	Not Useful	Somewhat Useful	Very Useful
QFG	14%	48.3%	23.2%	28.5%
TQGraph	28%	49.5%	31.8%	18.7%
KB-QRec	64%	44.2%	36.9%	18.9%
D-QRec	76%	40.3%	44.9%	14.7%

Following the evaluation methodology from prior work involving editorial assessment, we also conducted a user study to further compare the performance of all the configurations with D_{50} as the query log.

From the L_5 dataset mentioned before, we randomly extracted 50 queries to be our testbed. For each of the five configurations, we get the top-5 recommendations for these queries. For each query q and each of its recommendations q', we form a pair (q, q') to be assessed by 30 editors. Then, we shuffle these pairs so that the editors cannot distinguish from which configuration each pair originates. The editors were asked to give a rating to each of these pairs, using one of the following scores: $not\ useful$, $somewhat\ useful$, and $very\ useful$. Each pair was assessed by at least 6 users. We also record the coverage of each configuration.

The results are shown in Table 6. We can see that (i) our entity-based methods (KB-QRec and D-QRec) have much better coverage than QFG and TQGraph, (ii) among the four methods, if we do not consider the coverage, QFG has the best recommendation quality in terms of users' satisfaction; however, having such a low coverage means it cannot provide any recommendation for most of the queries, and (iii) TQGraph and our KB-QRec have similar user feedback overall, while our D-QRec has more recommendations in the "Somehow Useful" category.

In summary, considering their high coverage, our entity-based methods give a better performance trade-off for long-tail queries.

9.5 Evaluation of the hybrid recommendation system

We also evaluated the performance of the priority-based hybrid system, for various ways of setting the priorities, and also with respect to the linear hybrid system. We combined the three best performing methods, i.e., TQGraph (short as T), KB-QRec (short as K) and D-QRec (short as D),

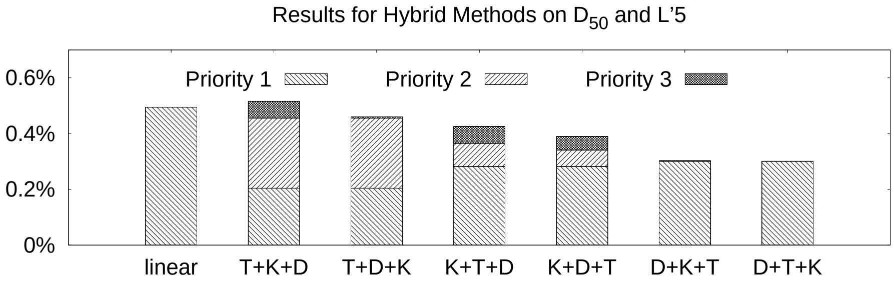

Fig. 9. Precision@5 results for hybrid systems on long-tail queries with entities.

and we test all the six possible priority orders. For example, we denote by "T+K+D" the system giving TQGraph the top priority, KB-QRec the second priority, and D-QRec the lowest priority. For the linear aggregation system as discussed in Section 7, we show its performance when using the optimal weight combination.

As all the configurations have the same coverage, we only compare their performances by precision@5. The results are in Figure 9. We can see that (i) the priority-based approach "T+K+D" achieves the best precision@5, and (ii) if we give D-QRec the top priority, the hybrid system behaves like D-QRec alone. This is because D-QRec has very high coverage (almost 100%), so the remaining methods do not often have the chance to contribute to the result of the hybrid system.

9.6 Efficiency

In this section, we evaluate the running time of the proposed methods. All the experiments were performed on a 3.40 GHz quad-core machine running Ubuntu 12.04, with 16 GB of main memory.

Table 7 shows the performance of the baseline methods using PPR cache. Observe that they are both fast and can provide instantaneous query recommendations.

Table 7. Average Running Time of QFG and TQGraph

	D_{10}	D_{50}	D_{90}
QFG	2ms	6ms	14ms
TQGraph	2ms	9ms	17ms

For the offline part of KB-QRec, the times for building the necessary data structures, including reading and writing on disk, are shown in Table 8. As we can see, the time for building the index increases linearly with the size of the query log. Specifically, on D_{90}, it takes 132 minutes to build the index, including the meta path set $\mathcal{P}$, which remains a reasonable time range for this offline step. By design, D-QRec does not require offline processing.

For the offline part of PowerWalk, as VENUS is not open-source, we asked the authors of [19] to run the necessary experiments on our behalf. The testbed is a single machine with 16GB RAM and a quad-core 3.70GHz CPU (Intel E5-1620). The running time for building the offline index for PowerWalk is shown in Table 8.

For the online part, we need to follow the three steps detailed in Section 4 to perform recommendations.

For Step 1, as pointed out in [5], entity linking can be done in a fast and space-efficient manner. In our experiments, we directly use the open-source framework of Dexter2 [44] to perform entity linking. Its average running time for is $59ms$ per query, similar to the result of the $Wikifier$

Table 8. Efficiency for building index offline.

	D_{10}	D_{50}	D_{90}
KB-QRec	14 min	56 min	132 min
PowerWalk	5.74 s	44.87 s	62.93 s

Table 9. Efficiency (in ms)

	Step 2 (KB-QRec)	Step 2 (D-QRec)	Step 3 (basic)	Step 3 (PPR cache)	Step 3 (PowerWalk)	KB-QRec	D-QRec
D_{90}	34 ms	2ms	91 ms	9 ms	9ms	43 ms	11 ms
D_{50}	34 ms	1ms	55 ms	5 ms	2ms	36 ms	6 ms
D_{10}	33 ms	1ms	13 ms	1 ms	2ms	35 ms	3 ms

method [14], as reported in [5]. If one would use instead the *FEL* method proposed in [5], the entity linking time would be further reduced to $0.4ms$.

Table 9 shows the average running time for Steps 2 and 3, and for the end-to-end system using the two proposed methods. We can see that (i) it takes $34ms$ for entity expansion in KB-QRec (Step 2), and this does not change with the size of the query log, because we only perform path-constrained random walks on the KB; (ii) Step 2 for D-QRec is very fast, as the number of entities is relatively small, compared with the number of queries; (iii) if we do not use a cache, the running time for PPR computations (Step 3) varies almost linearly with the size of query log; on the largest query log, D_{90}, this step takes $91ms$; (iv) if we do use a cache, as discussed in Section 8.1, the time for PPR computations is tenfold reduced; now, it takes only $9ms$ on D_{90}; (v) adopting PowerWalk can lead to similar running times as with the PPR cache, while PowerWalk does not require nearly as much storage; and (vi) taking into consideration all the overhead, assuming PowerWalk is adopted as the PPR speed-up component, the online running time for KB-QRec is around $40ms$, while that of D-QRec is around $10ms$, i.e., both methods generate their query recommendations instantaneously.

10 CONCLUSION

In this paper, we propose an entity-based framework for query recommendation, one of the most visible features in Web search today. Our framework first extracts entities from the input query, and uses these entities to explore new ones, which are then used to provide query suggestions. We develop two algorithms, namely KB-QRec and D-QRec, which use KBs and click logs respectively to provide query recommendations that are both relevant and diverse. The performance of our suggested methods for long-tail queries is significantly better compared to the state-of-the-art techniques. This is important, as long-tail queries constitute a large part of the query traffic in Web search engines and it is difficult to recommend queries for them.

Since semantics complements the behavioral patterns that can be extracted from query logs, our technique is orthogonal to existing methods. As such, it should be combined with semantic-agnostic approaches, such as the query-flow graph or the term-query flow graph, which exploit the sequentiality of similar queries and the “wisdom-of-crowds”. We develop this idea into a hybrid recommendation system, which acts as a middleware over various methods for selecting recommendation candidates, and uses a priority-based integration scheme. Using a real-world dataset from a major commercial Web search engine, extensive automatic evaluation and experiments with editorial assessments show that both our entity-based methods, and the hybrid system integrating them, can bring significant improvements, in terms of both coverage and precision. Furthermore, our experiments on efficiency show that our techniques can be used within “typing latency”, and can provide almost instantaneous results.

ACKNOWLEDGMENTS

Reynold Cheng, Zhipeng Huang, Yudian Zheng and Jing Yan were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 104004572, 102009508, 104004129). Bogdan Cautis was partially supported by the French research project ALICIA (grant ANR-13-CORD-0020). Nikos Mamoulis was supported by the European Union’s Horizon 2020 research and innovation programme under grant agreement No 657347. We would like to thank the reviewers for their insightful comments.

REFERENCES

[1] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. 2004. Query Recommendation Using Query Logs in Search Engines. In *Current Trends in Database Technology - EDBT Workshops*. Springer, 588–596.

[2] Ricardo A. Baeza-Yates and Alessandro Tiberi. 2007. Extracting semantic relations from query logs. In *KDD*. ACM, 76–85.

[3] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*. ACM, 107–116.

[4] Roi Blanco, Berkant Barla Cambazoglu, Peter Mika, and Nicolas Torzec. 2013. Entity recommendations in web search. In *International Semantic Web Conference*. Springer, 33–48.

[5] Roi Blanco, Giuseppe Ottaviano, and Edgar Meij. 2015. Fast and space-efficient entity linking for queries. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. ACM, 179–188.

[6] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. 2008. The query-flow graph: model and applications. In *CIKM*. ACM, 609–618.

[7] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. ACM, 1247–1250.

[8] Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini. 2012. Efficient query recommendations in the long tail via center-piece subgraphs. In *SIGIR*. ACM, 345–354.

[9] Daniele Broccolo, Lorenzo Marcon, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. 2012. Generating suggestions for queries in the long tail with an inverted index. *Inf. Process. Manage.* 48, 2 (2012), 326–339.

[10] Fei Cai, Shangsong Liang, and Maarten de Rijke. 2014. Time-sensitive Personalized Query Auto-Completion. In *CIKM*. ACM, 1599–1608.

[11] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. 2008. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 875–883.

[12] Jiefeng Cheng, Qin Liu, Zhenguo Li, Wei Fan, John CS Lui, and Cheng He. 2015. VENUS: Vertex-centric streamlined graph computation on a single PC. In *ICDE*. IEEE, 1131–1142.

[13] Reynold Cheng, Zhipeng Huang, Yudian Zheng, Jing Yan, Ka Yu Wong, and Eddie Ng. 2017. Meta Paths and Meta Structures: Analysing Large Heterogeneous Information Networks. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*. Springer, 3–7.

[14] Xiao Cheng and Dan Roth. 2013. Relational inference for wikification. *Urbana* (2013), 1787–1796.

[15] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice*. Vol. 283. Addison-Wesley Reading.

[16] Doug Downey, Susan T. Dumais, and Eric Horvitz. 2007. Heads and tails: studies of web search with common and rare queries. In *SIGIR*. ACM, 847–848.

[17] Henry Feild and James Allan. 2013. Task-aware query recommendation. In *SIGIR*. ACM, 83–92.

[18] Bruno M. Fonseca, Paulo Braz Golgher, Edleno Silva de Moura, Bruno Pôssas, and Nivio Ziviani. 2003. Discovering Search Engine Related Queries Using Association Rules. *J. Web Eng.* 2, 4 (2003), 215–227.

[19] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*. 17–30.

[20] Qi He, Daxin Jiang, Zhen Liao, Steven CH Hoi, Kuiyu Chang, Ee-Peng Lim, and Hang Li. 2009. Web query recommendation via sequential query prediction. In *Data Engineering, 2009. ICDE’09. IEEE 25th International Conference on*. IEEE, 1443–1454.

[21] Zhipeng Huang, Bogdan Cautis, Reynold Cheng, and Yudian Zheng. 2016. KB-Enabled Query Recommendation for Long-Tail Queries. In *CIKM*. 2107–2112.

[22] Zhipeng Huang and Nikos Mamoulis. 2017. Location-Aware Query Recommendation for Search Engines at Scale. In *International Symposium on Spatial and Temporal Databases*. Springer, 203–220.

[23] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta structure: Computing relevance in large heterogeneous information networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1595–1604.

[24] Alpa Jain, Umut Ozertem, and Emre Velipasaoglu. 2011. Synthesizing high utility suggestions for rare web search queries. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 805–814.

[25] Shan Jiang, Yuening Hu, Changsung Kang, Tim Daly Jr, Dawei Yin, Yi Chang, and Chengxiang Zhai. 2016. Learning query and document relevance from a web-scale click graph. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 185–194.

[26] Ni Lao and William W Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine learning* 81, 1 (2010), 53–67.

[27] Qin Liu, Zhenguo Li, John Lui, and Jiefeng Cheng. 2016. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *CIKM*. ACM, 195–204.

[28] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. 2013. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*.

[29] Changping Meng, Reynold Cheng, Silviu Maniu, Pierre Senellart, and Wangda Zhang. 2015. Discovering meta-paths in large heterogeneous information networks. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 754–764.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). http://arxiv.org/abs/1310.4546

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems* 26 (2013), 3111–3119.

[32] Umut Ozertem, Olivier Chapelle, Pinar Donmez, and Emre Velipasaoglu. 2012. Learning to suggest: a machine learning framework for ranking query suggestions. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 25–34.

[33] Patrick Pantel and Ariel Fuxman. 2011. Jigs and lures: Associating web queries with structured entities. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 83–92.

[34] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[35] Shuyao Qi, Dingming Wu, and Nikos Mamoulis. 2016. Location Aware Keyword Query Suggestion Based on Document Proximity. *TKDE* 28, 1 (2016), 82–97.

[36] Ridho Reinanda, Edgar Meij, and Maarten de Rijke. 2015. Mining, Ranking and Recommending Entity Aspects. In *SIGIR*. ACM, 263–272.

[37] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. 2013. Learning to rank query suggestions for adhoc and diversity search. *Information Retrieval* 16, 4 (2013), 429–451.

[38] Milad Shokouhi. 2013. Learning to personalize query auto-completion. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 103–112.

[39] Milad Shokouhi and Kira Radinsky. 2012. Time-sensitive query auto-completion. In *SIGIR*. ACM, 601–610.

[40] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A Hierarchical Recurrent Encoder-Decoder for Generative Context-Aware Query Suggestion. In *CIKM*. ACM, 553–562.

[41] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*. ACM, 697–706.

[42] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.

[43] Idan Szpektor, Aristides Gionis, and Yoelle Maarek. 2011. Improving recommendation for long-tail queries via templates. In *WWW*. ACM, 47–56.

[44] Salvatore Trani, Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. 2014. Dexter 2.0: an open source tool for semantically enriching data. In *ISWC*. 417–420.

[45] Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. 2001. Clustering user queries of a search engine. In *Proceedings of the 10th international conference on World Wide Web*. acm, 162–168.

[46] www-dbpedia 2011. DBpedia 3.7. http://wiki.dbpedia.org/Downloads37. (2011).

[47] Zhiyong Zhang and Olfa Nasraoui. 2006. Mining search engine query logs for query recommendation. In *Proceedings of the 15th international conference on World Wide Web*. ACM, 1039–1040.