Improving Recommendation for Long-tail Queries via Templates

Idan Szpektor
Yahoo! Research
Haifa, Israel
idan@yahoo-inc.com

Aristides Gionis
Yahoo! Research
Barcelona, Spain
gionis@yahoo-inc.com

Yoelle Maarek
Yahoo! Research
Haifa, Israel
yoelle@ymail.com

ABSTRACT

The ability to aggregate huge volumes of queries over a large population of users allows search engines to build precise models for a variety of query-assistance features such as query recommendation, correction, etc. Yet, no matter how much data is aggregated, the *long-tail* distribution implies that a large fraction of queries are rare. As a result, most query assistance services perform poorly or are not even triggered on long-tail queries. We propose a method to extend the reach of query assistance techniques (and in particular query recommendation) to long-tail queries by reasoning about rules between query templates rather than individual query transitions, as currently done in query-flow graph models. As a simple example, if we recognize that 'Montezuma' is a city in the rare query "`Montezuma surf`" and if the rule '`<city> surf` → `<city> beach`' has been observed, we are able to offer "`Montezuma beach`" as a recommendation, even if the two queries were never observed in a same session. We conducted experiments to validate our hypothesis, first via traditional small-scale editorial assessments but more interestingly via a novel automated large scale evaluation methodology. Our experiments show that general coverage can be relatively increased by 24% using templates without penalizing quality. Furthermore, for 36% of the 95M queries in our query flow graph, which have no out edges and thus could not be served recommendations, we can now offer at least one recommendation in 98% of the cases.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: *Query formulation*; H.2.8 [**Database Applications**]: *Data Mining*

General Terms

Algorithms, Experimentation

Keywords

query templates, query recommendation, query mining

1. INTRODUCTION

Mining query logs on a large scale has been shown to be tremendously useful for web search and web applications. Query logs have been successfully explored for a variety of purposes such as core ranking, automatic query expansion, web caching, user modeling, matching ads, and more. One of the most direct and visible applications of query log mining is query recommendations in its multiple forms, from dynamic query suggestions "as you type" to related searches displayed on the search-engine results page.

Yet, in spite of their relative success, all query-log based methods exhibit the same weakness: they cease being as useful when reaching the long tail. As a result, methods that operate at the level of individual queries cannot, as of today, handle rare or one-time queries and this leads to a significant coverage issue on the long tail.

One approach could be to simply ignore the long tail and concentrate on serving best head and torso needs. This approach, we believe, would probably be a mistake. Indeed, Goel et al. [12] conducted a thorough analysis of the "anatomy of the long tail," and showed that most "ordinary people ..[have].. extraordinary tastes," i.e. all of us exhibit a certain level of eccentricity. Even more importantly, it turns out that supporting the tail boosts the head by providing users a convenient one-stop shop for both their mainstream and niche interests. Following this view, we believe that it is critical to appropriately handle long-tail queries, especially in the query-recommendation family of applications.

In this paper, we focus on related query recommendation, one of the tasks for which the long-tail issue is the most visible. We propose to address the long-tail problem by leveraging *query templates* [1], which are query constructs that abstract and generalize queries. Our key idea is to identify *rules* between templates as means for suggesting related queries. The rationale for our approach is based on the fact that many individual queries share the same query intent while focusing on different entities. Hence, their related queries also share similar structures. As an example, assume that the queries "`Los Angeles hotels`", "`New York hotels`" and "`Paris hotels`" have been abstracted into the common query template "`<city> hotels`". In addition, if "`Los Angeles restaurants`" is a query recommendation for "`Los Angeles hotels`" and similarly "`New York restaurants`" is a recommendation for "`New York hotels`", we can extract the general rule:

'`<city> hotels` → `<city> restaurants`'

Using such a rule, we could then offer for the query "`Yancheng restaurants`", the suggested query "`Yancheng hotels`" even if both are rare queries and had never been observed before.

The key challenge in this approach is to ensure that while significantly improving coverage for query recommendations, we maintain a certain level of quality in order to satisfy users. We even argue that a small drop in quality would be acceptable as long as the user has enough valid recommendations to select from. In contrast, not offering any suggestion damages the user experience, as lack of coverage remains a significant issue: we have observed on a Yahoo! query log sample that 34M queries, out of 95M, had no consecutive query. Thus, in our sample, any query recommendation system that leverages query reformulation would not be able to derive any information for more than 36% of the queries.

In this paper we make the following contributions:

- We introduce the concept of rules between query templates, which can be used to infer recommendations for rare or previously unseen queries. We apply the concept of template rules on the query-flow graph [5]. At a high level, the concept is general and can be used to enhance other query-log constructs.
- We explain how to build templates and more specifically how to extract template rules. Note that unlike prior work on templates, which used semi-supervision on restricted domains [1, 16], we can afford to work at a general, large-scale level because we extract rules rather than stand-alone templates and ambiguity is therefore automatically reduced (see Section 4.5).
- We introduce the *query-template flow graph* as an enrichment of the query-flow graph with templates. We then describe how to use the query-template flow graph for the task of query recommendation.
- We provide an experimental evaluation, which shows that using a query-template flow graph instead of a query-flow graph construct consistently improves the quality of recommendations, especially for long-tail queries. We verify our claims using manual evaluation, as well as a novel automated large-scale evaluation process over millions of tested recommendations.

2. RELATED WORK

Query recommendations. Most query-recommendation methods use similarity measures obtained by mining (*i*) the query terms, (*ii*) the clicked documents, and/or (*iii*) the user sessions containing the queries. Typical methods use a combination of these factors.

Query recommendation based on clicked documents. Baeza-Yates et al. [3] propose a measure of query similarity and use it to build methods for query expansion. Their technique is based on a term-weight vector representation of queries, obtained from the aggregation of the term-weight vectors of the URLs clicked after the query. Wen et al. [27] also present a clustering method for query recommendation that is centered around the query-click graph [4].

The query-click graph is also utilized for finding related documents using random walks [8], finding related keywords for advertising [11], query rewriting through co-citation generalization [2] and ranking related queries using the notion of hitting time [19].

Query recommendation based on query reformulations. Many effective approaches focus on the analysis of user query sessions [10, 14, 30]. Fonseca et al. [10] propose a query recommendation system based on association rules. Another attempt of extracting information from the query log is due to Zhang and Nasraoui [30], who represent each user session by a complete graph where consecutive queries are connected with an edge of a predefined weight d. Non-consecutive queries are connected by an edge weighted with the product of the weights on the walk connecting them. Recent papers have shown that not only the previous query, but also the long-term interests of users, are important for understanding their information needs [18, 22].

Jones et al. [14] introduce the notion of query substitution: for each query, a set of similar queries is obtained by replacing the whole query or its sub-phrases. White et al. [28, 29] use the query rewrites observed in a query log to generate query recommendations. Sadikov et al. [23] have recently proposed to cluster the refinements of a user query by performing a random walk on a query-document graph that incorporates both session and click information.

One limitation for query recommendation that is common to the above works is that recommendations cannot be made for queries that were not seen before. Additionally, the quality of recommendations declines for infrequent queries.

Name-entity and template extraction using query logs. A few researchers have addressed the problem of extracting name-entities and hierarchy knowledge-bases from query logs [9, 21]. The main idea of these papers is to use the query log in order to extract entities and populate lexicons and/or ontology data. Since in this paper we are not concerned with building an entity hierarchy, such a line of research is orthogonal and complementary to ours.

Few works attempt to tag query terms [13, 16], an important step towards generating templates for queries. These works apply semi-supervised approaches for tagging with predefined categories. Recently, Agarwal et al. [1] introduced query templates as means to detect query intent and query attributes for a specific domain. They propose a semi-supervised approach for learning relevant templates in a domain, where both the domain attributes and seed queries are manually provided. Unlike these works, we aim at broad-coverage recommendation using templates for any query, with neither predefined categories nor specific domains.

Templates in Natural Language Processing Finally we note that templates and rules between them have been successfully used in a number of NLP tasks such as information extraction [26], taxonomy population [24], question answering [17] and machine translation [6].

3. PRELIMINARIES

In this section we review the concepts of query logs and the query-flow graph, which are central concepts in our paper.

3.1 Query logs

Query-log analysis is typically used by search engines to build models of the interaction of the users with the search engine, and it is applicable to a wide range of applications. A typical query log $\mathcal{L}$ is a set of records $\langle q_i, u_i, t_i, V_i, C_i \rangle$, where q_i is a query submitted to the search engine; u_i is an anonymized identifier for the user who submitted the query; t_i is a timestamp; V_i is the set of documents returned as results to the query; and C_i is the set of clicked documents.

A useful concept is a *physical session*, or simply *session*: a sequence of queries of one particular user within a specific timeout t_θ. A typical timeout threshold often used in query-log analysis is $t_\theta = 30$ minutes.

3.2 The Query-Flow Graph

The query-flow graph, which was introduced by Boldi et al. [5], is a graph structure that aggregates information contained in a query log. At an intuitive level, a query-flow graph is a graph in which nodes are queries, and edges (q_i, q_j) with large weights indicate that query q_j is a very likely *reformulation* of query q_i. Formally, a query-flow graph is defined as a directed graph $G_{\mathrm{q}} = (Q_0, E_{\mathrm{qq}}, s_{\mathrm{qq}})$ where:

- $Q_0 = Q \cup \{s, t\}$ is the set of distinct queries Q submitted to the search engine plus two special nodes s and t, representing a *starting state* and a *terminal state* of any user search task;
- $E_{\mathrm{qq}} \subseteq Q_0 \times Q_0$ is the set of *directed edges*;
- $s_{\mathrm{qq}} : E_{\mathrm{qq}} \to (0..1]$ is a weighting function that assigns to every pair of queries $(q_i, q_j) \in E_{\mathrm{qq}}$ a score $s_{\mathrm{qq}}(q_i, q_j)$.

In the basic construction of the query-flow graph, two queries q_i and q_j are connected by an edge if there is at least one session in the query log in which q_j follows q_i. The edge weight w may depend on the application; one simple definition is to use as weight the frequency of the transition (q_i, q_j) out of all instances of q_i. While there are more sophisticated versions of query-flow graph the definition of edge weights is orthogonal to the focus of this paper and so we set query-flow graph edge weights to be transition probabilities.

One application of the query-flow graph, suggested by Boldi et al., is query recommendation. The idea is fairly natural: for a query q_i recommend queries q_j for which the weight of the edge (q_i, q_j) in the query-flow graph is high enough. Various alternatives for picking those queries were studied. Those alternatives were inspired by ideas based on PageRank and personalized PageRank.

An obvious limitation of using the query-flow graph for query recommendation is that no recommendation can be made for queries that were not seen before. This is the limitation that we try to alleviate with the present work.

4. QUERY TEMPLATES AND THE QUERY TEMPLATE FLOW GRAPH

In this section, we describe our query-template approach for handling the long-tail problem of query logs. We formally describe query templates and template construction. We then introduce relations between templates, and present a recommendation algorithm based on a query-flow graph enhanced with templates.

4.1 Query Templates

Our notion of query templates is similar to the one used by Agarwal et al. [1]; the main differences are that (*i*) we define templates over a hierarchy of entity types, and (*ii*) we define a global set of templates for the whole query log and we do not restrict our templates on specific domains (say, travel, weather, or movies).

We start our discussion by considering the set of all queries $Q = \{q_1, q_2, \dots\}$ that appear in a query log of a search engine, where each query, $q \in Q$ is a sequence of search terms, $q = w_1 \dots w_r$. For a query q we define a *tokenization* to be a grouping of the terms of q into consecutive sequences of terms. For example, *all* possible tokenizations of "`chocolate cookie recipe`" are: (`chocolate`)(`cookie`)(`recipe`), (`chocolate`)(`cookie recipe`), (`chocolate cookie`)(`recipe`), and (`chocolate cookie recipe`), where parentheses are used only for indicating the token boundaries. Very often, not all possible tokenizations of a query are meaningful.

In addition to the query log, we also assume that we have access to an additional source of information: a *generalization hierarchy* $H = (E, R)$ on a set of entities E. As usual, the set of generalizations $R \subseteq E \times E$ contains a pair $e_i \to e_j$ if the entity e_j is a semantic generalization of the entity e_i (R induces a DAG on E). We also say that e_i *is a type of* e_j. For example, R may contain the following pairs:

```
chocolate → food
chocolate → drink
cookie → dessert
chocolate cookie → dessert
dessert → food
food → substance
recipe → instruction
...
```

We overwrite the notation to denote $e_i \to e_j$ if there is a sequence of generalizations in R that leads from entity e_i to entity e_j in the hierarchy H. So, in the above example, we have `chocolate` $\to$ `substance`, since `chocolate` $\to$ `food` $\to$ `substance`. If $e_i \to e_j$, we write $d(e_i, e_i)$ to denote the *shortest path* from e_i to e_j using the generalizations of R. So, $d($`chocolate`,`substance`$) = 2$.

Now let $q = w_1 \dots w_r$ be a query with a tokenization $q = k_1 \dots k_s$. Assume that the i-th token of the query, k_i is present as an entity in the generalization hierarchy H, namely $k_i = e_i \in E$, and that $e_i \to e_j$, for some other entity e_j. We define a template $t(q \mid k_i \to e_j)$ of q by replacing the entity k_i with a *placeholder* of type $<e_j>$ in the query q. In other words $t(q \mid k_i \to e_j) = k_1 \dots k_{i-1}{<}e_j{>}k_{i+1} \dots k_s$. The angle brackets $<\cdot>$ are used to indicate that a substitution by a placeholder has taken place. We denote the set of templates of q by $T(q) = \{t_1, t_2, \dots\}$.

In our running example, some of the possible templates for the query $q =$"`chocolate cookie recipe`" are:

```
<food> cookie recipe
<drink> cookie recipe
<food> recipe
<substance> recipe
chocolate cookie <instruction>
```

We define $k_q(t)$ to be the token of q that was replaced when generating t and $e(t)$ to be the placeholder in t. In our example, $k_q($"`<food> recipe`"$)$ is '`chocolate cookie`' and $e($"`<food> recipe`"$)$ is '`food`'.

A template is utilized by instantiating it with a token whose type matches the type of the template placeholder. We denote by $I_{\mathrm{H}}(t \mid k)$ the query generated by instantiating template t with token k based on the generalization hierarchy H. For example, $I_{\mathrm{H}}($"`<food> recipe`" $\mid$ "`soup`"$)$ is "`soup recipe`". The result of instantiating an invalid token of an unmatched type is *null*. The power of template instantiation lies in its ability to generate queries that were not observed before, as a cross product of templates and entities.

For each template $t \in T(q)$ of a query q we associate a score $s_{\mathrm{qt}}(q, t)$. This score provides a measure of goodness of the template t as a generalization of the query q. Intuitively, for our query "`chocolate cookie recipe`", the template "`<food> recipe`" should have a higher score than "`<substance> recipe`". This captures the notion that the more we generalize a query, the less confident we are, the risk being to over-generalize. In addition, "`<food> recipe`" should have a higher score than "`<drink> cookie recipe`", so as to give preference to more semantically valid templates.

4.2 Template Rules

Rules among templates enable the inference of meaningful recommendations for queries that have been seen very rarely, even for the first time. Once queries have been generalized to templates, the next step is to mine for frequent transitions among templates, which we call *template rules*. The transition '`<city> hotels` $\rightarrow$ `<city> restaurants`' shown in the introduction is an example of a template rule.

The power of template rules lies in their ability to provide inference rules by aggregating over many different individual query transitions. Thus, template rules eliminate noise very effectively and capture meaningful transitions. More discussion on template rules follows in the rest of this section.

4.3 The Query-Template Flow Graph

4.3.1 Representation

A central concept in this paper is the *query-template flow graph*, which extends the query-flow graph of Boldi et al [5]. The query-template flow graph is a graph whose nodes represent not only the queries Q but also the templates T. In addition to the query-to-query edges E_{qq}, there are also edges E_{qt} that specify the mapping from a query q to its templates $T(q)$, and also edges E_{tt} between templates. More formally, the query-template flow graph is a graph $G_{\text{t}} = (Q_0, T, E_{\text{qq}}, E_{\text{qt}}, E_{\text{tt}}, s_{\text{qq}}, s_{\text{qt}}, s_{\text{tt}})$, where

- Q_0, as in the query-flow graph, is the set of queries plus the starting and terminal queries;
- T is the set of all templates formed by the queries of Q, that is, $T = \bigcup_{q \in Q} T(q)$;
- E_{qq} is the set of query-to-query transition edges, as in the query-flow graph;
- E_{qt} is the set of query-to-template mapping edges, that is, edges between queries $q \in Q$ and templates of $T(q)$;
- E_{tt} is the set of template-to-template transition edges, to be defined shortly;
- s_{qq} are query-to-query scores on the edges of E_{qq}, as in the query-flow graph;
- s_{qt} are query-to-template scores on the edges of E_{qt} mentioned above, to be defined shortly;
- s_{tt} are template-to-template scores on the edges of E_{tt}; $s_{\text{tt}}(t_1, t_2)$ is a measure of goodness that template t_2 is a good transition from template t_1, to be defined shortly;

We next describe how the parts of the query-template flow graph are constructed in this paper, namely, the set of templates T, the query-to-template and template-to-template edges, E_{qt} and E_{tt}, and their respective scores s_{qt} and s_{tt}.

4.3.2 Template construction

As discussed above, templates are constructed with the help of a generalization hierarchy H over entities. In our implementation, H is the WordNet 3.0 hypernymy hierarchy [20] and the Wikipedia category hierarchy, connected together via the YAGO[1] induced mapping [25]. This hierarchy features more than 1,750,000 entities with more than 4,470,000 direct generalizations between them. In general, our approach can employ any method for generating an entity hierarchy, e.g. from query logs [9, 21].

All the distinct queries in the query-flow graph are processed. For each query q we construct the set of templates $T(q)$ as follows:

First we normalize q by converting all characters to lower case, collapsing continuous spaces, removing non-printable characters, etc.

Then, every word n-gram up to length 3 in q is considered as a token for replacement by a placeholder. For every n-gram k, we add to $T(q)$ all the templates formed by replacing k with each of its generalizations in H (if any). We note that n-grams that consist of stop-words are ignored.

In addition to the above general scheme, we found that the coverage of the produced templates can be improved by taking care of a small number of special cases, which appear often in query logs, when an n-gram is not found in the YAGO hierarchy. If there are context words around such an n-gram (the n-gram is not the whole query) we further look into the following cases:

- If the n-gram is an email address, we generate an email-typed template, e.g., "`ggg@yahoo.com instant message`" becomes "`<email> instant message`".
- If the n-gram is a url, we generate a url-typed template, e.g. "`nbc.com login`" becomes "`<URL> login`".
- If the n-gram contains numbers, we generate a numerically-typed template, e.g., "`555-7777 address`" becomes "`<000-0000> address`".
- Otherwise, if the n-gram is a noun-phrase, we create a postfix-typed template, e.g., "`luxury cars sale`" becomes "`<?-cars> sale`".

4.3.3 Query-to-template edges

Query-to-template edges E_{qt} are used to specify the mapping from a query q to the templates $T(q)$. That is, we have $(q, t) \in E_{\text{qt}}$ if and only if $t \in T(q)$. As already mentioned, some templates in $T(q)$ are more confident generalizations than others, a notion captured by the score $s_{\text{qt}}(q, t)$ as the goodness of the edge $(q, t) \in E_{\text{qt}}$.

Let q be a query and $t \in T(q)$ be a template for q. To compute the score $s_{\text{qt}}(q, t)$ we first compute a score $S_{\text{qt}}(q, t)$, and we then normalize all scores $S_{\text{qt}}(q, t)$ to ensure that the total score from the query q sums to 1. For the unnormalized scores $S_{\text{qt}}(q, t)$ we use the following rules, which we found to work well in practice.

- If t was obtained by substituting the token $k_q(t)$ with an entity e in the hierarchy H then, intuitively, the higher e is located in the hierarchy, the less confident we are in the template quality. Consequently, we select our score to be exponentially decaying with the distance $d(k_q(t), e)$ between the token and the generalization e in the hierarchy:

$$S_{\text{qt}}(q, t) = \alpha^{d(z,e)}$$

where α was set to 0.9 in our experiments.

[1]`www.mpi-inf.mpg.de/yago-naga/yago/`

- For all email-typed, url-typed, and numerically-typed templates t of a query q we set $S_{\text{qt}}(q,t) = 0.5$.
- Finally, for all postfix-typed templates t of a query q we set $S_{\text{qt}}(q,t) = 0.1$.

In addition, if q' is a query and $(q,q') \in E_{\text{qq}}$ then, for convenience of notation, we set $S_{\text{qt}}(q,q') = 1$, otherwise, $S_{\text{qt}}(q,q') = 0$. Normalization is then performed by setting

$$s_{\text{qt}}(q,t) = \frac{S_{\text{qt}}(q,t)}{\sum_{t' \in T(q)\cup Q} S_{\text{qt}}(q,t')}, \text{ where } t \in T(q) \cup Q$$

It is important to note that both the template construction procedure and the query-to-template score function are not dependent on query occurrences in the query log, only on the hierarchy H. Hence, whenever we encounter a new query q that is currently not in Q_0 (an unseen query), we can add it to Q_0 as well as its mapping edges to already existing templates in T in an on-line fashion.

4.3.4 *Template-to-template edges*

The set of edges E_{tt} forms the set of template rules that were introduced in Section 4.2. Template rules describe the possible transitions between templates for suggesting related templates. Recalling our running example, a possible transition could be '`<food> recipe` $\rightarrow$ `healthy <food> recipe`'. Each template rule in E_{tt} is accompanied by a score s_{tt}, which captures the confidence of generating valid related queries by instantiating the corresponding templates.

Consider two templates t_1 and t_2 in the query-template flow graph. A directed edge (t_1,t_2) is in E_{tt} if and only if:

- $e(t_1) = e(t_2)$, i.e., they have the same placeholder type.
- there is at least one edge (q_1,q_2) in the query-flow graph between queries q_1 and q_2 such that $t_1 \in T(q_1)$, $t_2 \in T(q_2)$ and $k_{q_1}(t_1) = k_{q_2}(t_2)$, that is the substituted token in q_1 and in q_2 is the same. We call such query-to-query edges *support edges* and denote the set of support edges for (t_1,t_2) by $\text{Sup}(t_1,t_2)$.

As an example, an edge '`sandwich recipe` $\rightarrow$ `healthy sandwich recipe`' in the query-flow graph would give rise to the edge '`<food> recipe` $\rightarrow$ `healthy <food> recipe`' in the query-template flow graph.

To compute the score $s_{\text{tt}}(t_1,t_2)$ between the templates t_1 and t_2, we sum over the scores of all supporting pairs of queries, and we normalize so that the total score out of each template sums to 1:

$$S_{\text{t}}(t_1,t_2) = \sum_{(q_1,q_2)\in \text{Sup}(t_1,t_2)} s_{\text{qq}}(q_1,q_2),$$

$$s_{\text{tt}}(t_1,t_2) = \frac{S_{\text{t}}(t_1,t_2)}{\sum_t S_{\text{t}}(t_1,t)}.$$

4.4 Generating Query Recommendations

We next discuss how to utilize the query-template flow graph in order to provide query recommendations. Given an input query q that was submitted by a user, our task is to recommend other queries q' that are likely to be helpful for the user. Our output is a ranked list of candidate related-queries, ordered by their confidence score (highest first).

Using the query-flow graph, the candidate related queries are those for which there is a directed edge (q,q') in E_{qq}. The related query confidence score $r(q,q')$ is $s_{\text{qq}}(q,q')$.

Similarly, utilizing the query-template flow graph, the candidate related queries are those for which there is either a directed edge (q,q') in E_{qq} or there is a mapping edge (q,t) and a template-to-template transition edge (t,t'), such that $I_{\text{H}}(t' \mid k_q(t)) = q'$, that is the instantiation of template t' with the mapping token of q to t results in q'.

For example, one recommendation for the query "`Adele Astaire`" is "`Adele Astaire biography`", based on the mapping edge '`Adele Astaire` $\rightarrow$ `<artist>`' and the transition edge '`<artist>` $\rightarrow$ `<artist> biography`'.

Notice that the recommended query q' does not have to be in Q, i.e. it does not have to appear in the query log. This is because q' is generated as an instantiation of a template with a token extracted from the input query (which itself could be an unseen query). The confidence score $r(q,q')$ based on the query-template flow graph is:

$$r(q,q') = s_{\text{qt}}(q,q')\cdot s_{\text{qq}}(q,q') + \sum_{\substack{t\in T(q)\,\wedge \\ (t,t')\in E_{\text{tt}}\,\wedge \\ I_{\text{H}}(t'|k_q(t))=q'}} s_{\text{qt}}(q,t)\cdot s_{\text{tt}}(t,t').$$

Since both scores $s_{\text{qt}}(q,t)$ and $s_{\text{tt}}(t,t')$ are normalized, $r(q,q')$ is always between 0 and 1. In particular, it can be interpreted as the probability of going from q to q' by one of the feasible paths in the query-template flow graph. We note that in our experiments we rank first queries that were seen before as related ($s_{\text{qq}}(q,q') > 0$), assuming that they are more reliable recommendations.

4.5 Discussion

Choosing the right template or set of templates for a query is a difficult task, since many queries consist of more than one term, and many terms are ambiguous. For example, in "`jaguar transmission fluid`" it is unclear on which term the focus is. Additionally, "jaguar" has several meanings based on which the query could be generalized. Hence, prior work on query templates focused on semi-supervised approaches in specific domains, where the task is simpler [1, 16]. However, we aim at a general large-scale application of template rules for all kinds of queries. For our advantage, extracting rules instead of single templates diminishes the problems mentioned above. First, shared terms between queries typically coincide with the query focus. Second, aggregating occurrence statistics for rules emphasizes the correct sense of ambiguous terms under the query context.

For example, the query transition '`jaguar transmission fluid` $\rightarrow$ `jaguar used parts`' indicates that "jaguar" is the focus. In addition, by observing other transitions, e.g. '`toyota transmission fluid` $\rightarrow$ `toyota used parts`', the template rule '`<car> transmission fluid` $\rightarrow$ `<car> used parts`' is brought forward as the frequent rule, diminishing the effect of other possibly extracted rules, such as '`<feline> transmission fluid` $\rightarrow$ `<feline> used parts`'.

4.6 Complexity and System Design

One nice feature of the proposed method is that it can be implemented efficiently in a *map-reduce* environment, such as the hadoop platform[2]. For building the query-template flow graph, templates may be generated independently for each query, together with their mapping scores, at hadoop nodes that keep the hierarchy H in memory. Template nodes are then aggregated at reduce time. If H is too big to fit

[2]`http://hadoop.apache.org/`

Table 1: Statistics for the query-flow graph and the query-template flow graph used in our experiments (query statistics are the same for the query-flow graph and the query-template flow graph).

	queries	templates
# nodes	95,279,132	5,382,051,983
# edges	83,513,590	4,345,497,267
avg in/out degree	0.88	0.81
max out-degree	14,145	34,249
	(craigslist)	(<album>)
max in-degree	14,317	133,874
	(youtube)	(<institution>)

Table 2: Accuracy of each configuration and each sample-set based on manual evaluation

	QFG	QTFG
set-A	98.48%	97.84%
set-B	97.65%	98.86%
set-C	—	94.38%

in main memory, it may be partitioned among the hadoop nodes. In this case, different n-gram pieces of one query may be processed by different hadoop nodes and be aggregated at the end. Finally, the template-to-template edges and scores may also be computed in a map-reduce fashion as a sequence of joins—we omit the details.

On the other hand, generating query recommendations needs to be an online process. For a computationally viable solution, the idea is again to split the query-template flow graph among different back-end nodes, mapped according to queries and query templates. A new query is first parsed by a broker to generate the set of possible templates. Then the templates are sent to the appropriate back-end nodes to compute candidate recommendations and scores, which are returned to the broker in order to be aggregated and produce the final ranking.

5. EXPERIMENTS

In this paper, we tested the proposed query-template flow graph on the task of query recommendation. We compared between two configurations: (*i*) recommendations using the query-flow graph, denoted QFG; and (*ii*) recommendations using the query-template flow graph, denoted QTFG.

We conducted two experiments. The first experiment follows the typical manual evaluation of query recommendation methods [3, 10, 18], where the quality of recommendations is assessed by human judges. However, manual evaluation suffers from limitations, such as human labor and scale. Therefore, as a second experiment, we propose a novel automatic evaluation approach for the query recommendation task.

5.1 Implementation

For constructing the query-flow graph, and consequently the query-template flow graph, we used a development query log from which we sampled user sessions. For testing the graphs, sessions were sampled from a later query log, denoted the *test-query-log* (see each experiment for the exact test-set construction details).

Table 1 presents statistics for the two graphs. From the figures we see that 56 candidate templates are generated on average for each query, and this reflects the number of template-to-template edges generated. While the query-template flow graph performance is significantly better than the query-flow graph (see the following sections), this generation is still noisy and many of the generated templates and edges between templates are incorrect, though with lower scores. In future work, we plan to improve our identification of correct templates and edges that should be generated for a given query or a given edge in the query-flow graph.

The average degree of a node is very small, but the variance is huge. While about a third of the nodes (both for queries and templates) have no outgoing edges and about a similar percentage have no incoming edges, the maximum in and out degrees are very large. For templates, some of the large degrees are due to very general templates, while some queries (and consequently templates) are simply very frequent in sessions, as shown in the table.

5.2 Manual Evaluation

Following the typical evaluation procedure in prior work, the authors evaluated a random sample of query recommendations for the two tested configurations, QFG and QTFG.

For each configuration we randomly sampled 300 queries that occurred in the test-query-log, and which were followed by a consecutive query within a single session. Each system suggested related queries, from which one recommendation was chosen randomly out of the top 10 recommendations. In total, 300 pairs of an input query and one of its recommendations were sampled for each configuration, denoted *set-A*.

In addition, 100 pairs were randomly chosen from the 300 pairs sampled for QFG. We then extracted, for the input queries in these pairs, the recommendations by QTFG that were ranked in the same position as those in the query-flow graph pairs. This sample, denoted *set-B*, compares more directly between the two methods on the same input queries and the same recommendation ranked positions.

Finally, 100 queries, for which no recommendation could be provided by QFG (queries that were not seen before), were randomly sampled from the test-query-log, and one of the top 10 recommendations by QTFG was randomly selected for each. This sample, denoted *set-C*, assesses the extension capabilities of the query-template flow graph.

In total, 800 query-recommendation pairs were evaluated. Two of the authors blindly evaluated these pairs, with an overlap of 100 queries for agreement assessment. The measured agreement between the two judges is 93% and the corresponding Kappa statistic [7] is 0.37, which is typically viewed as "fair" agreement [15]. No decision was made by the authors for about 10% of the examples.

Table 2 presents the results of the manual evaluation. The accuracy of both methods is very high, with a slight gain for the query-flow graph in general (*set-A*). Yet, when looking at proposed related queries for the same input queries at the same ranked position (*set-B*), we see that actually it is QTFG that has a slight gain over QFG. Nevertheless, the power of QTFG over QFG is demonstrated by the results on the set of unseen queries (*set-C*): for this set, even though the overall performance of QTFG drops slightly it still remains very high (94.38%), while the QFG could not provide any recommendation at all.

5.3 Automatic Evaluation

5.3.1 Motivation

While manual evaluation provides a good quality assessment for query recommendation systems, it has several limitations. First, the task is hard to define—which recommendations should be considered as valid related queries. This difficulty is reflected in the mediocre Kappa value achieved for the inner annotator agreement. Furthermore, the two annotators reported that though almost all recommendations seem "related", some seem more useful than others, a quality that is hard to measure.

A second limitation of the manual evaluation is that the amount of examples that are analyzed manually typically sums up to no more than several hundred examples [3, 8, 10, 30], which may not be statistically indicative (compared to query-log sizes). Finally, manual evaluation is a slow process, which poses a real problem when several rounds of evaluation are needed during the development of a new system or algorithm.

A different approach is to directly evaluate a tested recommendation method in a search engine via bucket testing. Yet, this is not a feasible option for the majority of researchers in the field.

5.3.2 Methodology

To overcome the limitations of the current evaluation approaches, we propose a novel automatic evaluation based on a previously unseen query log. This query log is partitioned into sessions and from each session the list of consecutive query pairs are extracted. Our assumption is that if a user entered two queries, $\{q_i, q_{i+1}\}$, one after the other in the same session, then the second query may be viewed as a valid related query for the first query. Furthermore, we assume that since a user explicitly entered q_{i+1} as related to q_i, q_{i+1} should be considered more related to q_i compared to other possible related queries. Thus, a recommendation system should also provide q_{i+1} as a recommendation to q_i, and rank it high. These seem reasonable assumptions, as they are also behind recommendation methods that learn from query logs, such as the query-flow graph.

Our proposed automatic evaluation is to test how many of the pairs $\{q_i, q_{i+1}\}$, extracted from the new query log, are also proposed by the tested recommendation system. Furthermore, these recommendations should be ranked high. This evaluation setup has the advantage of being fast, as it is automatic. In addition, it may be applied to a large number of pairs (several millions in our experiment bellow). Finally, there are no agreement issues between annotators, since any related query generated by real users is taken as a valid example – as the gold standard.

We note that this evaluation ignores other recommendations for a given query by the tested system, which may also be valid. Thus, the absolute quality of recommendations is not directly assessed. Yet, the proposed evaluation is very useful as a relative comparison between different recommendation systems, since when testing on a large volume of recommendations that were useful for users, each system should propose (and rank high) a reasonable amount of them. This evaluation measures how well a recommendation system answers the need for related queries as explicitly formulated by a large number of real users.

5.3.3 Experimental Setup

We extracted pair occurrences from the sampled sessions in our test-query-log (see Section 5.1). We experimented with two versions of this test-set. In the first version, denoted *all-pairs*, all pairs of consecutive queries were extracted from each session, summing up to 3,134,388 pairs. In the second version, denoted *first-last*, we generated just one pair from each session, which consists of the first and the last query in the session. In this setup, which contains 4,591,044 pair occurrences, we assume that the last query in a session is the real target query of the user, and we test if the systems can propose it given the first query in the session.

To confirm that these gold-standard pairs indeed describe related queries, 100 pairs were sampled from the *first-last* dataset and were blindly evaluated by two of the authors together with the 800 examples judged in Section 5.2 (900 in total). The accuracy achieved for this sample is 100%, that is all pairs are related queries. This result further supports our choice of user generated pairs as a valid test-set for related query recommendation.

We measured the following figures over the performance of each tested configuration:

- How many of the tested pairs could be proposed by the system (upper-bound coverage).
- How many pairs were ranked in the top 100 recommendations of the system.
- How many pairs were ranked in the top 10 recommendations of the system (the typical list visible for users).
- How many pairs were ranked highest (first place).
- Mean Average Precision (MAP) of the test-pairs, viewing no more than 100 recommendations per query (if the pair is not in the top 100, its precision is 0).
- Average position (rank) of test-pairs, only for those that appear in the top 100 recommendations.

Some of the pairs occur in our dataset more than once. We thus report the above figures both for *pair occurrences* and for *unique pairs* (considering each pair only once).

While our initial target for using the query-template flow graph was to address the long tail of infrequent queries and unseen queries, we also noticed that the query-template flow graph helps to better rank recommendations in general. Thus, in addition to the QFG and QTFG configurations we experimented, on the *first-last* data-set, with a configuration of query-template flow graph restricted only for re-ranking recommendations by query-flow graph, denoted QTFG-RERANK.

5.3.4 Results

Tables 3 and 4 present the results for the *all-pairs* and *first-last* test-sets. The first result seen from the tables is that by utilizing our current implementation of the query-template flow graph the coverage of the test-pairs increases relatively by 22-24% (depending on the test-set). If only unique pairs are considered, the increase is even more substantial, by about 45% for both test-sets. This shows that the query-template flow graph significantly increases the coverage for the long tail of infrequent or unseen queries. For example, for the input query "`1956 dodge lancer`", QTFG ranked first the test-pair {"`1956 dodge lancer`", "`1956 dodge lancer for sale`"} using rules such as '`1956 <car>` $\rightarrow$ `1956 <car> for sale`', while QFG could not suggest any related query for "`1956 dodge lancer`". Other examples are presented in Table 5.

When focusing on the top-10 recommendations per query, the difference in test-pair coverage is widening. This result

Table 3: Results on the *all-pairs* data-set for each configuration

pair occurrences					
	QFG		QTFG		relative increase
total in test-set		3134388		3134388	
upper-bound coverage	(22.65%)	709832	(28.17%)	882851	24.37%
# in top-100	(16.97%)	531854	(25.49%)	799001	50.23%
# in top-10	(9.49%)	297462	(20.74%)	649939	118.49%
# ranked highest	(2.86%)	89740	(10.01%)	313638	249.5%
MAP		0.050		0.137	
avg. position		18.35		8.3	
unique pairs					
	QFG		QTFG		relative increase
total in test-set		2755922		2755922	
upper-bound coverage	(13.28%)	366047	(19.38%)	533960	45.87%
# in top-100	(12.06%)	332487	(17.25%)	475323	42.96%
# in top-10	(8.41%)	231811	(13.52%)	372481	60.68%
# ranked highest	(2.86%)	78783	(6.5%)	179093	127.32%
MAP		0.047		0.089	
avg. position		12.33		9.43	

Table 4: Results on the *first-last* data-set for each configuration

pair occurrences							
	QFG		QTFG-RERANK		QTFG		relative increase QTFG vs. QFG
total in test-set		4591044		4591044		4591044	
upper-bound coverage	(27.63%)	1268579	(27.63%)	1268579	(33.85%)	1554282	22.52%
# in top-100	(18.78%)	862232	(24.79%)	1137920	(28.59%)	1312408	52.21%
# in top-10	(10.22%)	469165	(19.37%)	889383	(21.53%)	988568	110.71%
# ranked highest	(3.21%)	147501	(9.32%)	427828	(10.11%)	464260	214.75%
MAP		0.055		0.128		0.140	
avg. position		19.36		9.52		10.75	
unique pairs							
	QFG		QTFG-RERANK		QTFG		relative increase QTFG vs. QFG
total in test-set		3856355		3856355		3856355	
upper-bound coverage	(15.51%)	598114	(15.51%)	598114	(22.62%)	872377	45.85%
# in top-100	(13.7%)	528333	(14.48%)	558255	(18.86%)	727220	37.64%
# in top-10	(9.23%)	355843	(11.63%)	448383	(14.13%)	545062	53.17%
# ranked highest	(3.33%)	128267	(6.01%)	231797	(6.92%)	266862	108.05%
MAP		0.052		0.079		0.094	
avg. position		13.48		8.63		10.97	

is also reflected by the higher MAP values when utilizing query-template flow graph. The difference is at its most when looking at the highest ranked recommendation, where the number of user-generated related queries that are ranked first is twice as much for unique queries and more than thrice as much for pair occurrences. This surprising result indicates that query-template flow graph not only provides recommendations for unseen or rare queries, but it also helps to better rank recommendations that are proposed by the query-flow graph. For example, QTFG ranked first the test-pair {"`gangrene`", "`gangrene symptoms`"} using rules such as '`<pathology>` → `<pathology> symptoms`', while QFG ranked it at 23rd place.

The surprising result that the query-template flow graph helps to better rank the query-flow graph recommendations is also expressed by the performance of the QTFG-RERANK configuration, where a significant improvement in performance is achieved just by re-ranking the query-flow graph output by the query-template flow graph. This is further emphasized by looking at the average position of recommendations in the top-100. While QTFG recommendations are better positioned on average than QFG, QTFG-RERANK achieves the best average positions for user-generated related queries. The reason for lower positions in QTFG than in QTFG-RERANK is due to additional unseen queries for which only QTFG could provide recommendations. For these queries the average position is lower than for the seen ones, as expected, and hence the lower overall average position.

5.3.5 Analysis

To further understand the behavior of the query-template flow graph vs. the query-flow graph, we conducted several analyses on the results of the *first-last* test-set. We first looked at the potential of the query-template flow graph to extend the query-flow graph for nodes in the query-flow graph that have no outgoing edges. The queries in these nodes were seen before (in the development query log, from which the query-flow graph was constructed), but no related queries were observed for them (they were only observed as related queries for other queries). There are 33,883,564 such queries (36% of the nodes), and the query-template flow graph managed to propose at least one related query for 98% of them. This further emphasizes the potential benefit of the query-template flow graph for previously unseen query relations.

Next, we analyzed the change in test-pairs ranking when

Table 5: Examples of test-pairs that were ranked high by QTFG while QFG could not propose any related query for the input query, showing the large verity of rules that were learned by our approach.

Pair	Rank	Example rule used
{"8 week old weimaraner", "8 week old weimaraner puppy"}	1	8 week old <breed> → 8 week old <breed> puppy
{"1910 swimsuit", "1910 swimsuit mens"}	1	1910 <clothing> → 1910 <clothing> mens
{"a thousand miles notes", "a thousand miles piano notes"}	2	<single> notes → <single> piano notes
{"aaa office twin falls idaho", "aaa twin falls idaho"}	1	aaa office <city> → aaa <city>
{"air force titles", "air force ranks"}	2	<military service> titles → <military service> ranks
{"beach cameras", "live beach cameras"}	1	<geo formation> cameras → live <geo formation> cameras
{"guangzhou flights", "guangzhou map"}	6	<capital> flights → <capital> map
{"i am legend action figure", "i am legend"}	2	<fiction> action figure → <fiction>
{"name for salt", "chemical name for salt"}	2	name for <compound> → chemical name for <compound>

Table 6: Ranking differences of suggestions for the *first-last* test-set

	QFG		QTFG	
total in test-set		2755922		2755922
# in top-100		528333		727220
# in top-100 but not in the top-100 of the other	(1.42%)	7486	(28.38%)	206373
# in top-10 but not in the top-10 of the other	(3.07%)	16211	(28.25%)	205430
# better positioned when both in top-100	(13.47%)	71170	(35.05%)	254919

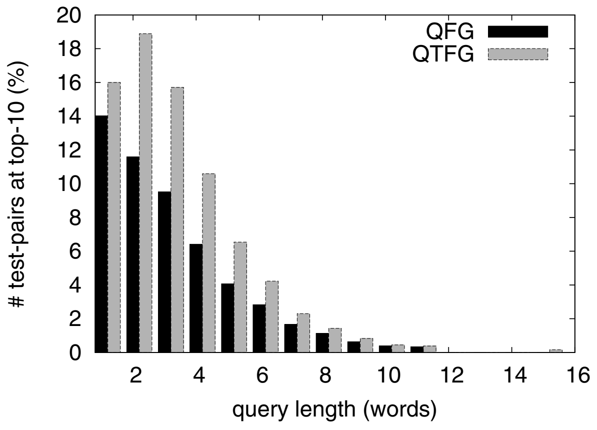

Figure 1: *first-last* pair coverage at top-10 suggestions vs. query length

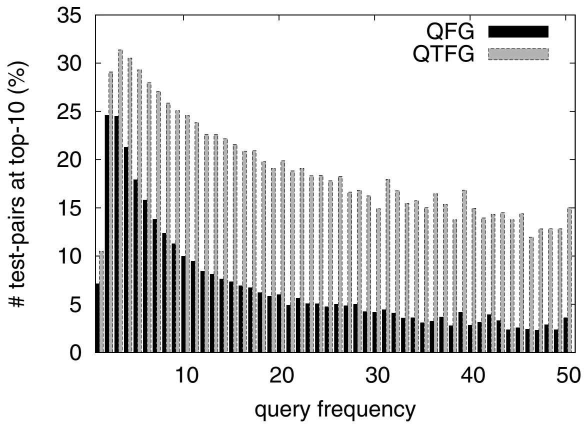

Figure 2: *first-last* pair coverage at top-10 suggestions vs. query frequency

the input query length (in words) varies. From the graph presented in Figure 1, we see that the query-flow graph shows a typical decline in performance for longer queries, as these are less frequent and thus have less history to rely on. However, things are different when using the query-template flow graph. It actually performs better when the input queries have a few additional words, since they act as a disambiguating context for ranking appropriate recommendations (see Section 4.5).

In our third analysis, presented in Figure 2, we looked at the change in test-pairs ranking quality when the input query frequency varies. The plot in Figure 2 shows that both configurations find it hard to suggest test-pairs as recommendations for queries that occur only once in the test set. This behavior agrees with the manual evaluation, which showed lower suggestion quality for unseen queries. There is a substantial increase in performance for queries that occur twice and then a decline as the frequency increases, since frequent queries have a more diverse history and its hard to predict the best suggestions from this history. Yet, we see that the query-template flow graph consistently outperforms the query-flow graph, managing to pick more appropriate suggestions for frequent queries. For example, for the query "`jack johnson lyrics`", which occurred 14 times in the test-set, QTFG ranked the pair {"`jack johnson lyrics`", "`jack johnson music`"} 5th, while QFG ranked it 43rd.

Finally, we analyzed the differences in ranks of the same suggestions by the query-flow graph and the query-template flow graph. This analysis, presented in Table 6, shows that many test-pairs that either cannot be processed by the query-flow graph or are ranked very low, are positioned in the top-10 recommendations in the query-template flow graph. This indicates that the query-template flow graph handles well recommendations that cannot be processed by the query-flow graph. Furthermore, only very few pairs (3%) are ranked significantly higher by the query-flow graph than by the query-template flow graph (in the top-10), which means that the query-template flow graph hardly hurts the ranking of well ranked recommendations by the query-flow graph. This is despite the fact that about 13% of the test-pairs are ranked higher by the query-flow graph than by the query-template flow graph. These improvements are not significant, as they are mainly at the lower part of the top-100 ranks and are usually not displayed to users.

To conclude, the analysis shows that utilizing the query-template flow graph on top of the query-flow graph improves consistently the ranking of user-generated related queries, without any significant loss in performance for a specific type of queries.

6. CONCLUSIONS

We introduced the concepts of rules between query templates and the query-template flow graph as an abstraction and a generalization approach for relations between queries. This novel approach is useful for addressing the long tail of rare or previously unseen queries in various search-related tasks. Yet, it is also helpful in discovering important relations between frequent queries, for example for better ranking possible suggestions in query recommendation.

We conducted two query-recommendation experiments, a manual evaluation and a novel large-scale automated evaluation. The manual evaluation showed that both the query-template flow graph and the baseline query-flow graph are very adequate as methods for query recommendations. Yet, our automatic evaluation over millions of query-recommendation pairs showed that the query-template flow graph consistently outperforms the query-flow graph by ranking higher recommendations that were explicitly chosen by users. More importantly, the query-template flow graph provides good suggestions for many unseen queries, for which the query-flow graph could not provide any suggestion.

In future work, we plan to apply the query-template flow graph on other search-related tasks and to further explore its structure and behavior. In addition, we want to improve the quality of rule extraction. Finally, we would like to investigate the identification of edges from templates to queries, which capture relations such as between different towns in a district and the single airport that serves them.

7. REFERENCES

[1] G. Agarwal, G. Kabra, and K. C.-C. Chang. Towards rich query interpretation: Walking back and forth for mining query templates. In *WWW*, 2010.

[2] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: Query rewriting through link analysis of the click graph. In *VLDB*, 2008.

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

[4] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD*, 2000.

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

[6] D. Chiang. A hierarchical phrase-based model for statistical machine translation. In *ACL*, 2005.

[7] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, April 1960.

[8] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, 2007.

[9] M. Fernández-Fernández and D. Gayo-Avello. Hierarchical taxonomy extraction by mining topical query sessions. In *KDIR*, 2009.

[10] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani. Using association rules to discover search engines related queries. In *LA-WEB*, 2003.

[11] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW*, 2008.

[12] S. Goel, A. Broder, E. Gabrilovich, and B. Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *WSDM*, 2010.

[13] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *SIGIR*, 2009.

[14] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, 2006.

[15] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 1977.

[16] X. Li, Y.-Y. Wang, and A. Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *SIGIR*, 2009.

[17] D. Lin and P. Pantel. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):343–360, 2001.

[18] J. Luxenburger, S. Elbassuoni, and G. Weikum. Matching task profiles and user needs in personalized web search. In *CIKM*, 2008.

[19] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM*, 2008.

[20] G. A. Miller. Wordnet: A lexical database for english. In *Communications of the ACM*, 1995.

[21] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *CIKM*, 2007.

[22] M. Richardson. Learning about the world through long-term query logs. *ACM Transactions on the Web*, 2008.

[23] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *WWW*, 2010.

[24] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005.

[25] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge - unifying wordnet and wikipedia. In *WWW*, 2007.

[26] I. Szpektor and I. Dagan. Learning entailment rules for unary templates. In *COLING*, 2008.

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

[28] R. W. White, M. Bilenko, and S. Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *SIGIR*, 2007.

[29] R. W. White, M. Bilenko, and S. Cucerzan. Leveraging popular destinations to enhance web search interaction. *ACM Transactions on the Web*, 2(3), 2008.

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