Query Suggestions in the Absence of Query Logs

Sumit Bhatia
Computer Science and Engineering
Pennsylvania State University
University Park, PA-16802, USA
sumit@cse.psu.edu

Debapriyo Majumdar
IBM Research India
Bengaluru-560045, India
debapriyo@in.ibm.com

Prasenjit Mitra
Information Science and Technology
Pennsylvania State University
University Park, PA-16802, USA
pmitra@ist.psu.edu

ABSTRACT

After an end-user has partially input a query, intelligent search engines can suggest possible completions of the partial query to help end-users quickly express their information needs. All major web-search engines and most proposed methods that suggest queries rely on search engine query logs to determine possible query suggestions. However, for customized search systems in the enterprise domain, intranet search, or personalized search such as email or desktop search or for infrequent queries, query logs are either not available or the user base and the number of past user queries is too small to learn appropriate models. We propose a probabilistic mechanism for generating query suggestions from the corpus without using query logs. We utilize the document corpus to extract a set of candidate phrases. As soon as a user starts typing a query, phrases that are highly correlated with the partial user query are selected as completions of the partial query and are offered as query suggestions. Our proposed approach is tested on a variety of datasets and is compared with state-of-the-art approaches. The experimental results clearly demonstrate the effectiveness of our approach in suggesting queries with higher quality.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – Query formulation

General Terms

Algorithms, experimentation

Keywords

Query suggestion, query formulation, query completion, query log analysis, enterprise search.

1. INTRODUCTION

As users start typing a query in search engines' query boxes, most search engines assist users by providing a list of queries that have been proven to be effective in the past [26]. The user can quickly choose one of the suggested completions (in some cases, alternatives) and thus, does not have to type the whole query herself. Feuer et al. [15] analyzed more than 1.5 million queries from search logs of a commercial search engine and found that query suggestions accounted for roughly 30% of all the queries indicating the important role played by query suggestions in modern information retrieval systems. Furthermore, Kelly et al. [20] observed that the use of offered query suggestions is more for difficult topics, i.e., topics about which users have little knowledge to formulate good queries.

Most existing works on query suggestion utilize query logs to suggest queries [2, 4, 9, 10, 13, 18, 24, 25, 27]. Such query log based techniques are suitable for systems with a large user base. For example, web search engines have millions of users and if a user has some information need, almost always it turns out that some other users have searched for the same information before. The search engine can then utilize these large amounts of past usage data to offer possible query suggestions.

The scenario, however, is quite different for information retrieval systems that are not on the web or have a smaller user base, and thus, lack large amounts of query log data. For example, search engines built for customized applications in the enterprise domain are used by only a few hundreds or thousands of users. An effective query suggestion mechanism is required because the difference between users not finding some critical information and the users finding the information with ease may have a very significant business impact for the organization. The query logs of such systems are relatively much smaller, especially when a system is newly deployed for a small number of users, and even later on, the log seldom becomes as exhaustive as that of a web search engine. Even if a system can accumulate a relatively large query log over time, it would take a lot of time to be mature enough and most often that time is not affordable due to business needs. Furthermore, the data residing in these systems are also so customized that using a query log from another system is not a valid approach. However, the effectiveness of such enterprise search systems has significant business implications and even a small improvement can have a positive impact on the organization's business. Similarly, for personal data search systems, such as desktop search or personal email search, often there is only a single user resulting in very small query logs. Besides, query logs may not always be accessible in some applications due to privacy and legal constraints. Furthermore, even in the case of general-purpose web search engines, end-users sometimes pose queries that are not there in query logs or are not very frequent. *How to offer meaningful query suggestions to users in such scenarios* is thus, an interesting research problem, which we explore in this paper.

SIGIR'11, July 24–28, 2011, Beijing, China.

Effective query suggestion requires inferring a user's query intent and information needs and then suggesting queries that may help retrieve documents containing relevant information. Formulating such queries that can help information retrieval systems discriminate between relevant and irrelevant documents requires predicting and using words, phrases or their combinations that are present in relevant documents and absent in irrelevant documents, which is an extremely difficult task for an average end-user [8]. Cui et al. [14] have shown that in the term vector-space, a large gap exists between the terms used by users to formulate queries and terms that are present in relevant documents. The aim of a query suggestion mechanism then is to iteratively use the partial information about the user's need in the form of incomplete queries and to suggest to the end user queries that match the end-user's information needs and those for which the search engine knows good results. Inspired by this, we propose a *document-centric probabilistic mechanism* to generate query suggestions that does not depend on query logs and utilizes only the co-occurrence of terms in the corpus. The central idea of our approach is simple and intuitive. As soon as a user starts typing a query, phrases from the corpus that are highly related to the partial user-entered-query are selected as possible completions of the partial query and thus completed queries are offered as query suggestions.

We evaluated our proposed approach with a variety of datasets and compared the performance of our approach with that of the state-of-the-art approach [6] and that of one available in a widely used open-source enterprise search system [1]. The experimental results demonstrate the superiority of our method in its ability to *(i)* offer suggestions for a wide variety of queries, *(ii)* offer more contextually meaningful suggestions, and *(iii)* generate suggested queries with higher retrieval effectiveness when compared to the baselines.

As we discuss in Section 2, though there have been some works in the past that can be adopted for query suggestion without using query logs, but strictly speaking, to the best of our knowledge, this paper is the first to study the problem of *query suggestions in the absence of query logs*. Our approach is simple yet effective and powerful, and as discussed later in Section 6, it also opens up several aspects of improvements and future work aligned with the concept of facilitating user's search without the aid of query logs.

The rest of the paper is organized as follows. Section 2 provides a review of the previous work on query suggestion – both with and without query logs. In Section 3, we define the problem formally and describe the proposed probabilistic approach in detail. Experiments and results are described in Sections 4 and 5. Section 6 concludes the paper and offers directions for future work.

2. RELATED WORK

2.1 Relation With Interactive Query Expansion

Query suggestion differs from interactive query expansion (IQE) [16] in a fundamental way. In IQE, the user is offered a list of suggested terms from which she can select a few to *augment the original query*. However, users tend to prefer complete query suggestions to individual term suggestions because of the additional context provided by the complete query suggestion as opposed to an isolated term suggestion [20]. The Real Time Query Expansion (RTQE) technique as proposed by White and Marchionini [28] can also be seen as a means for providing real-time query expansions where a list of expansion terms is offered in real time that the user can select and expand the query. However, in addition to offering only single terms as suggestions, this method is limited by the fact that it requires the user to type complete query terms and can not offer suggestions for incomplete query terms as the goal here is to *refine the original query* so as to improve the retrieval performance. On the other hand, query suggestion mechanisms try to *suggest various possible completions* of the (incomplete) query the user is still typing so as to save time and cover various possible interpretations of the user's information needs.

2.2 Query Suggestion using Query Logs

Large scale web search engine logs contain real, user issued queries. Initial works on query suggestion focus on identifying past queries similar to the current user query. Baeza-Yates et al. [2] cluster queries present in search logs and given an initial query, similar queries from its cluster are identified based on vector similarity metrics and are then suggested to the user. Barouni-Ebrahimi and Ghorbani utilize phrases frequently occurring in queries submitted by past users as suggestions [4]. Gao et al. describe a query suggestion mechanism for cross lingual information retrieval where for queries issued in one language, queries in other languages can also be suggested [17].

In addition to user queries, query logs also contain valuable clickthrough data and session information. Many of the previous works on query suggestion, especially in the web domain, exploit this additional information. By utilizing clickthrough data and session information, Cao et al. propose a context aware query suggestion approach [10]. In order to deal with the data sparseness problem, they use *concept* based query suggestions where a *concept* is defined as a set of similar queries mined from the query-URL bi-partite graph. The user's *context* is defined as the sequence of concepts about which the user has issued queries before submitting the current query. Given this context information, queries that are asked frequently in a given context are mined from search logs and are suggested to the user. For a given user query, Ma et al. [24] utilize clickthrough data to identify semantically related queries from query logs. Song and He [27] combine information from clicked URLs as well as skipped URLs to identify suggestions for rare queries.

In a single search session, a user often modifies her initial query a few times in order to obtain the relevant information. Jones et al. [18] utilize such modifications made by previous users for suggesting queries to other users. Li et al. [22] describe a method to compute pairwise similarity scores between queries based on the hypothesis that queries that co-occur in a search session are related. Given a user query, most similar queries thus identified are used as suggestions. Cucerzan and White [13] utilize user *landing pages* (pages where users finally end up after post-query navigation) to generate query suggestions. For each landing page of a user submitted query, they identify queries from query logs that have these landing pages as one of their top ten results. These queries are then used as suggestions.

Boldi et al. [9] utilize *query flow graphs* for query suggestions. Nodes in a query flow graph are past user queries and an edge from q_i to q_j indicates that both queries are related to a similar information need. Short random walks on a query flow graph are then conducted to generate query suggestions. Baraglia et al. [3] however, note that the models based on query flow graphs suffer from an ageing effect and suggesting queries using the information extracted from them may not be possible over time.

Mei et al. [25] propose an algorithm based on hitting time on the Query-URL bipartite graph derived from search logs. Starting from a given initial query, a subgraph is extracted from the Query-URL bipartite using depth first search. A random walk is then conducted on this subgraph and hitting time is computed for all the

query nodes. Queries with the smallest hitting time are then used as suggestions. Ma et al. [23] show how the hitting time analysis on Query-URL bipartite graph can be utilized to diversify suggestions.

2.3 Query Suggestion without Query Logs

Among the works where query logs are not used as the primary resource, Feuer et al. [15] describe a proximity search based system that suggests alternate queries if the initial query is highly specific (results in very few documents) or too broad (large number of search results). In case of underspecified queries, most frequent terms appearing in close proximity of the query terms in the corpus are added to the query to narrow down the search results. For specific queries, most frequent subphrases of the query are presented as possible modifications. Their work however is similar in principle to *query refinement* and is quite different from query suggestion where the objective is to present different possible queries to the user as soon as she starts typing in the query box. Furthermore, their approach is not real time and requires the user to type a complete query first.

Bast and Weber developed the CompleteSearch engine [6, 5], which is an efficient instant search system that computes and presents search results online and refreshes the results with each letter typed or modified by the user. It also offers real-time autocompletion of the last query term being typed by the user. These completions are extracted from the search results obtained by using the incomplete query (and the primary goal of their work is to use an efficient index to find those search results), in the order of decreasing frequency of the term's occurrence in the search results. When viewed as a query suggestion system, CompleteSearch is limited in the sense that it only offers completions of the last query word instead of suggesting complete queries related to the incomplete query. Moreover, it requires the user to type at least two characters of the last query term. Also, using frequency as the only criterion for ranking the completions is not an optimal choice as the most frequent completions of the last query word often are not related to the query terms already typed by the user. However, the auto-completion feature in their system can be viewed as a query suggestion system, and hence we consider this method as a baseline in our experiments.

3. PROPOSED APPROACH

3.1 Problem Setup and Solution Overview

Consider a user u who has an information need $\mathcal{I}$. She transforms the information need into a query Q and starts typing the query in the query box of a search engine. The user has some information need but is not sure what terms to use to formulate a query. Since the documents indexed by the search engine are not visible to the user, often the terms selected by the user to formulate the queries do not lead to good retrieval performance due to the gap between query-term space and document-term space [14]. To help the user, information retrieval system can search the query logs to identify queries similar to the user's query that have been successful in the past and can suggest such queries to the user. However, our goal is to offer query suggestions to the user even in scenarios where query logs are not available. In the absence of query logs, we adopt a document-centric approach by utilizing the documents in the corpus itself to generate query suggestions on the fly. We extract and index phrases from the document corpus and when the user starts typing a query, we utilize these phrases to complete the partial user query. The completed queries are then offered as suggestions to the user. In the remainder of this section, we first describe the phrase extraction process and then discuss how these phrases can be used to generate query suggestions.

3.2 Phrase Extraction

In order to create a database of phrases that can be used for completing partial user queries, we extract all N-grams of order 1, 2 and 3 (that is unigrams, bigrams and trigrams) from the document corpus. One can also extract higher order N-grams but the number of possible N-grams increases exponentially with the order N [19, Chapter 4] and hence, is not scalable for any real world corpus. Further, while extracting N-grams, we need to take special care of the stop words. Consider the phrase `president of usa`. Each of the possible bi-grams from this phrase (`president of, of usa`) starts or ends with a stop word and is thus, an *incomplete* phrase. Hence, they are not desired as the resulting query completions. One possible solution can be to remove all the stopwords from corpus before extracting N-grams. However, removing stop-words may also lead to loss of semantics and make the resulting suggestions harder to understand. For example, compare `president usa` with `president of usa` and `president in usa`. If we remove the stop-words, the second and third phrase will both reduce to the first phrase even though they mean different things. In order to avoid such difficulties, we use an idea similar to skip-grams [19, Chapter 4] from natural language processing. Instead of *skipping over* and *discarding* the adjacent words, whenever we encounter a stop-word we *jump over* to the next word and *retain* the stop-word so that the resulting phrases do not start or end with stop words. Thus, in the above example, we will only have *one* bi-gram (`president of usa`). Note that now the order of an N-gram is *not* the number of words in the N-gram but the number of *non stop-words*.

3.3 Probabilistic Model for Query Suggestion

Consider the time instant when the user has typed first k characters of the query, which we denote by Q_1^k. Note that these k characters can contain a space character and if it does that means the user has typed more than one query term. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ denote the set of extracted phrases that can be used for generating query suggestions and let $\mathcal{V}$ be the vocabulary of the corpus available to us. Given the incomplete query Q_1^k, $\mathcal{V}$ and $\mathcal{P}$ we wish to construct a set $\mathcal{S} \subset \mathcal{P}$ such that each $s \in \mathcal{S}$ is a possible completion for Q_1^k.

Ideally, we want to offer suggestions to the user such that the queries represent the information need of the user. However, the only information we have about the user is the incomplete query Q_1^k. Further, different users having different information needs can start with same Q_1^k. For example, the queries **linux in**`terview questions` and **linux in**`stallation` have the same prefix `linux in`. Thus, for a given partial query Q_1^k, our task is to select phrases that can be used for generating possible query suggestions. To solve this problem, we ask this question: *Given a partial query Q_1^k and a phrase $p_i \in \mathcal{P}$, what is the probability $P(p_i|Q_1^k)$, i.e., the probability that the user will eventually type p_i after typing Q_1^k?* Once we answer the above question, we can order the phrases by the probability of their being typed after Q_1^k and use the top ranked phrases for offering suggestions to the user. We now describe how to compute $P(p_i|Q_1^k)$.

We start by making the observation that at any given instant of time, Q_1^k can be decomposed as follows:

$$Q_1^k = Q_c + Q_t \tag{1}$$

where, Q_c denotes the completed portion of the query, i.e., the set of words that the user has typed completely. Note that $|Q_c| \geq 0$.

Q_t is the last word of Q_1^k that the user is still typing. Note that it may be a complete word or a partial word. Further, $|Q_t| \in \{0, 1\}$.

Using Bayes' theorem, the probability $P(p_i|Q_1^k)$ can be written as:

$$P(p_i|Q_1^k) = \frac{P(p_i) \times P(Q_1^k|p_i)}{P(Q_1^k)} \tag{2}$$

Further, assuming that the query terms are conditionally independent, $P(Q_1^k|p_i)$ can be written as:

$$P(Q_1^k|p_i) = P(Q_t|p_i) \times P(Q_c|p_i) \tag{3}$$

Using equations 2 and 3, we have:

$$P(p_i|Q_1^k) = \frac{P(p_i) \times P(Q_t|p_i) \times P(Q_c|p_i)}{P(Q_1^k)} \tag{4}$$

where, Q_t and Q_c are as defined in equation (1).
By definition of joint probability, we have:

$$P(p_i)P(Q_t|p_i) = P(p_i, Q_t) = P(Q_t)P(p_i|Q_t). \tag{5}$$

Application of equation (4) to equation (5) yields:

$$P(p_i|Q_1^k) = \frac{P(Q_t)P(p_i|Q_t)P(Q_c|p_i)}{P(Q_1^k)} \tag{6}$$

Further, we note that Q_1^k and Q_t remain the same for all the phrases given a user submitted partial query. Therefore $P(Q_t)$ and $P(Q_1^k)$ are constants for a given user query and thus, can be safely ignored since we are interested only in the relative ordering of phrases. Taking these observations into account, equation (6) reduces to:

$$P(p_i|Q_1^k) \stackrel{rank}{=} \underbrace{P(p_i|Q_t)}_{\text{phrase selection probability}} \times \underbrace{P(Q_c|p_i)}_{\text{phrase-query correlation}} \tag{7}$$

The above equation summarizes our proposed model for query suggestion. The first component of equation (7) measures the probability that phrase p_i can be typed by the user given that he has already typed Q_t. The second component measures the correlation between p_i and component Q_c of the user query. We now describe how these two probabilities can be estimated.

3.4 Estimating Phrase Selection Probability

Selecting a candidate phrase given a partial word is a two step process. First find a completion of the partial word and then select a phrase that contains that completed word. Figure 1 illustrates this process in terms of a graphical model that can be used to estimate $P(p_i|Q_t)$. The root node corresponds to the event that the user has typed Q_t – a partial word. This partial word can be completed in m different ways where m is the number of words in vocabulary that start with Q_t. Let $\mathcal{C} = c_1, \ldots, c_m$ be the set of m such possible word completions represented by corresponding nodes in Figure 1. Let $\mathcal{P}_i = p_{i1}, \ldots, p_{1n_i}$ be the set of n_i phrases that contain the completed word c_i. Given Q_t, each completion c_i has a probability $P(c_i|Q_t)$ of being selected. Once c_i is selected as a possible word completion, we select a phrase $p_{ij} \in P_i$ with the probability $P(p_{ij}|c_i)$. In this way, the probability of selecting a phrase given a partial word is expressed as follows.

$$P(p_{ij}|Q_t) = \underbrace{P(c_i|Q_t)}_{\text{term completion probability}} \times \underbrace{P(p_{ij}|c_i)}_{\text{term to phrase probability}} \tag{8}$$

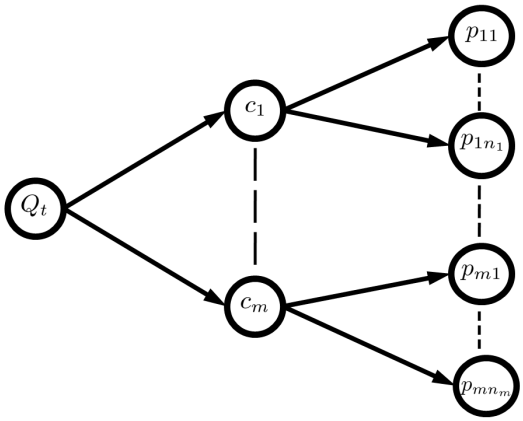


Figure 1: Graphical depiction of phrase selection process.

Since we have no other information about user's information need except for the partial word Q_t, we make a simplifying assumption. We assume that phrases in the corpus that are more important have a higher chance of being used by the user for formulating queries than the less important ones. One way to assess the importance of phrases is by using their occurrence frequencies in the corpus. However, this naïve approach has two serious shortcomings:

- In our formulation, the first step in phrase selection is to first find a completion c of the last query word Q_t with a probability $P(c|Q_t)$ and then select all phrases that contain that completion. If we use only raw frequencies to compute $P(c|Q_t)$, some of the important, but rare, completions will get suppressed. Hence, while computing $P(c_i|Q_t)$, we normalize the frequencies of different completions by their IDF values as follows.

$$P(c_i|Q_t) = \frac{freq(c_i) \times IDF(c_i)}{\sum_{i=1}^{m} freq(c_m) \times IDF(c_m)} \tag{9}$$

- In general, the frequency of unigrams in the corpus is much higher than the frequency of bigrams and trigrams. For the datasets used in this paper, we observe the average frequency of unigrams to be 15–25 times higher than the average bigram and trigram frequencies. Such large differences in frequency values suppresses the selection of bigrams and trigrams as compared to unigrams. Cui et al., [14] also faced similar problems and suggested using a constant multiplication factor to normalize the frequencies of bigrams and trigrams. Instead of using a constant factor for all phrases, we employed a normalization factor that normalizes raw frequencies of different order n-grams using the log ratio of their average frequencies as follows.

$$freq_{norm}(\text{order } m \text{ n-gram } p) = \frac{freq(p)}{log(avgFreq(m))} \tag{10}$$

where,
$avgFreq(m)$ is the average frequency of all n-grams of order m.

Using this formulation, the term to phrase probability can be computed as follows.

$$P(p_{ij}|c_i) = \frac{freq_{norm}(p_{ij})}{\sum_{k=1}^{n_i} freq_{norm}(p_{ki})} \quad (11)$$

3.5 Estimating Phrase-Query Correlation

The phrase selection component of equation (7) selects phrases on the basis of the last query word (Q_t) only. It does not take into account the context in which the user has typed Q_t. For example, consider following two partial queries: `bill gate` and `india gate`. The first query is related to Bill Gates and the second query is about a historical monument in India. For both these queries, the last word is same and therefore, we will end up with the same set of phrases for both these queries even though they represent very different information needs. Clearly, we require a mechanism to identify whether a given phrase is *contextually* important or not.

The second component of equation (7) takes into account such a relationship between a phrase and the user-submitted query. It represents the probability that the user has typed Q_c given that we know that the selected phrase p_i represents the completion of Q_c. In other words, given that p_i represents the latter half of the *complete query* we want to compute the probability that Q_c is the first half of the complete query.

By using the laws of probability, $P(Q_c|p_i)$ can be written as follows:

$$P(Q_c|p_i) = \frac{P(Q_c, p_i)}{P(p_i)} \quad (12)$$

Here, $P(Q_c, p_i)$ represents the probability of the joint occurrence of Q_c and p_i and $P(p_i)$ represents the probability of observing p_i alone. Both these probabilities can be estimated using the corpus as follows:

$$P(Q_c|p_i) = \frac{|D_{Q_c} \cap D_{p_i}|}{|D_{p_i}|} \quad (13)$$

Here, D_{p_i} and D_{Q_c} represent the sets of documents that contain phrase p_i and Q_c respectively. In order to find the set of documents containing a particular phrase p, we make a simplifying assumption and approximate D_p as the set of documents that contain all the constituent words in phrase p. Mathematically,

$$D_p \approx \bigcap_{w \in p} D_w \quad (14)$$

where D_w is the set of documents containing word w.

This approximation has two important advantages. First, it greatly simplifies finding the set D_p as all the required sets of documents containing the constituent words (i.e., D_w's in equation 14) are already available in the search engine's index in the form of postings lists of respective words. Second, it also helps overcome the data-sparseness problem. As an example, consider the following three search queries: `linux install firefox`, `install firefox linux` and `firefox install linux`. All these queries represent the same information need and are represented using the same set of terms. However, the ordering of the constituent terms is different in all three queries. Thus, it is possible that the phrases present in the corpus may have a different ordering than what the user has typed and thus, we will miss such phrases. Further, in the relevant documents these terms might not always appear together as a phrase. In Section 4, we show that the baseline method (SimSearch) that relies on such exact phrase matches performs poorly and is not able to offer any suggestions at all for many queries. Decomposing a phrase into its constituent terms avoids the above problems. By such a formulation, phrases that contain terms that co-occur frequently with the user query are given a higher weight. It also helps in making sure that the resulting query suggestions have good retrieval capability as the resulting query suggestions will consist of terms that frequently co-occur.

To summarize our approach, given a user query, the system first finds all the possible completions of the last query word. These completions are then used to identify a set of phrases that can be used for generating possible query suggestions. All the phrases in this candidate set are then assigned a probability score using equation 7. The top 10 highest scoring phrases are then presented to the user after appending to the Q_c portion of the user query.

		Ubuntu	TREC
No. of Documents		109,137	210,158
Avg Document Size (words)		378.02	374.26
N-Grams Extracted	**Unigrams**	114,702	191,123
	Bigrams	1,062,204	3,020,118
	Trigrams	888,492	2,677,026

Table 1: Different datasets used and their properties. These datasets differ in number of documents and average document length (measured in number of words). Number of words in a document was obtained by using the Unix `wc` command.

4. EXPERIMENTAL PROTOCOL

4.1 Data Description

We evaluated our proposed query suggestion mechanism using the following two datasets:

1. **TREC:** This dataset, available on TREC Disk4, consists of more than 200,000 news articles published in Financial Times between years 1991–1994.

2. **Ubuntu:** This dataset, used in previous research [7], consists of more than 100,000 discussion threads crawled from `ubuntuforums.org`, a set of 25 queries and relevance judgments. The documents contain discussions on various topics related to Linux and Ubuntu. One motivation for using this dataset was to test the robustness of the proposed approach as the text here is much more informal and noisy as compared to the TREC dataset.

For both the datasets, all the documents were pre-processed to remove punctuations and all the text was converted to lower case. N-Grams (order 1, 2 and 3) were extracted as described in Section 3. For extracting n-grams, stop words used came from a general stop word list of 429 words used in the Onix Test Retrieval Toolkit[1]. The number of N-Grams extracted for each dataset along with some other statistics are summarized in Table 1.

4.2 Baseline Methods

We compare our proposed approach with the following two baseline methods:

1. **Similarity based phrase search (SimSearch):** In this method, the phrase index is searched to find all the phrases that contain the user submitted partial query as a subphrase. The selected phrases are presented to the user in order of their

[1]http://www.lextek.com/manuals/onix/stopwords1.html

Dataset	Original Query	Type-A Query	Type-B Query
Ubuntu	automount hard drive partitions	automount	automount hard drive part
	virtualbox keyboard problem	virtualbox	virtualbox keyb
	wine microsoft office	wine	wine microso
TREC	falkland petroleum exploration	falkland	falkland petro
	encryption equipment export	encryption	encryption equip
	radioactive waste	radioactive	radioactive was

Table 2: Examples of queries used in experiments.

occurrence frequency. This method is used by the popular open source enterprise search server Solr[2].

2. **CompleteSearch (CompSearch):** This method refers to the query suggestion mechanism implemented in CompleteSearch search engine [6, 5] (as discussed in the section on related work).

We do not compare our system with any method that uses query logs because we strongly believe that such a comparison would essentially be comparing algorithms for totally different technical problems and would be unfair.

4.3 Test Queries

In order to compare our proposed approach with the baselines, we need a set of queries for which the suggestions can be generated. We evaluated our approach with a set of 40 test queries for each dataset. For the Ubuntu dataset, queries came from the 70 most searched for topics on ubuntu forums (provided by ubuntu forums) and for the TREC dataset, queries from TREC topics 351–400 (title field) were used. For each dataset, we removed stop words from each query, discarded all the single word queries and then randomly selected 20 queries from the remaining queries. Each such query was then used to generate two different partial queries as follows, leading to 40 partial test queries for each dataset.

- **Type-A Queries:** These queries were generated by retaining only the first keyword of each of the 20 queries.
- **Type-B Queries:** These queries were generated by retaining the first keyword of the query followed by the first k characters of the remaining query string ($2 \leq k \leq$ length of the remaining query string). The number k was selected at random for each query. Note that we use $k \geq 2$ because the CompSearch method requires users to type at least two characters as discussed in Section 2. The proposed method however, has no such restriction.

For example, the query correspinding to TREC topic 387 is `radioactive waste`. The type-A query for this query is `radioactive` and type-B query is `radioactive was`. This example also illustrates the main motivation behind generating two types of queries. Type-A queries are, in general, broader in scope than their type-B counterparts. For the query `radioactive`, all of `radioactive waste management`, `radioactive waste disposal` etc. could be valid query suggestions. However for `radioactive was`, the query suggestion mechanism should take into account the context in which the user is typing the last query word and ensure that queries such as `radioactive washington post` are not suggested. Some examples of queries used in our experiments are given in Table 2.

[2] `http://lucene.apache.org/solr/features.html#Query`

4.4 User Study

Due to the absence of standard test collections for the query suggestion task, we conducted a user study to measure the quality of the suggestions generated by different query suggestion methods. For each test query, we collected the top 10 suggestions generated by the three methods (SimSearch, CompSearch and our proposed method (Probabilistic)). Evaluation was performed with the help from 12 volunteers who were our colleagues and were not associated with the project. All the subjects were experienced users of search engines and used both web search engines as well as enterprise search engines on a daily basis. Each assessor was shown a randomly selected partial user query from the set of test queries and lists of query suggestions produced by the three methods. The order in which the suggestions produced by different methods were presented to the assessors was randomized for each query and each evaluator and the assessors did not know which set of suggestions was generated by which method. The suggestions produced by a particular method were presented in decreasing order of their respective scores. Each query was evaluated by three unique assessors and each provided evaluations for an equal number of test queries. For each query suggestion the subjects were asked to assign one rating from among the four possible options that are summarized in Table 3. The final rating for a suggestion was decided by a majority vote. By a *meaningful* (and hence relevant) suggestion for a partial query, we mean a query that represents *one possible prediction of the user's information need* as a valid search query, given the partial query typed by the user. Also, we do not desire almost duplicate suggestions, or malformed combination of words (which do not make sense) to appear in the list of suggestions. By designing the four options for rating each suggestion and by providing a clear guideline to the assessors, we made a best possible effort to convey the notion of a suggestion being *meaningful* to our assessors. The final judgments had to be subjective and we had to depend on our assessors as they are experienced users of several kinds of search engines and also the query suggestion features of web-search engines. In total, there were 1906 suggestions evaluated by the assessors and a majority decision (two out of three assessors assigned the same rating) was obtained for 91.45% (1743) of the suggestions. Further, 1147 suggestions were assigned a **Y**(meaningful) rating by the majority vote and out of these 915 suggestions were judged as **Y** by all the three assessors. These inter-assessor agreement results strongly indicate that the notion of suggestions being relevant or meaningful was considered very consistently among them.

A suggestion for which all the three assessors provided different ratings was assigned the rating of "Not Sure". For the purpose of comparing different query suggestion methods, we considered only suggestions marked as **Y** to be relevant and suggestions marked with any of the three remaining ratings were treated as non-

Rating	Meaning
Y	Yes, a meaningful suggestion
N	No, not a meaningful suggestion, or badly formed as a query
D	An (almost) duplicate suggestion, conveys no new information
??	Not sure

Table 3: Different rating options available to users and their meanings.

relevant. We use this information to compare success rates, precision etc. for the different methods in the following sub-sections.

4.5 Statistical Significance Tests

In order to assess the statistical significance of the results obtained, we used the paired two-sample one-tailed t-test. Statistically significant improvements over the SimSearch and CompSearch methods are indicated by **S** and **C**, respectively for $p < 0.01$ (99% confidence interval) whereas **s** and **c** indicate statistically significant improvements with $p < 0.05$ (95% confidence interval) over SimSearch and CompSearch, respectively.

5. RESULTS AND DISCUSSIONS

5.1 Success Rate of Different Methods

A popular metric to compare the performance of different query suggestion methods is *coverage*; it has been used previously to compare various query suggestion mechanisms [10, 18]. Coverage of a query suggestion method is defined as the fraction of queries for which the method is able to offer at least one query suggestion. However, we feel that coverage is, in general, a relaxed evaluation metric as it does not take into account the quality of the queries suggested. What if the queries suggested are not meaningful? As opposed to the query-log based query suggestion methods that generally offer frequent past *user queries* as suggestions, the query log oblivious methods evaluated in this paper *generate* suggestions by combining partial user query with terms/phrases derived from the corpus and hence, may not always generate meaningful queries. Also note that for a given partial query, in general, there will be *multiple* possible valid completions. We consider a query suggestion method successful for a given partial query if it is able to generate at least one meaningful suggestion for the partial query. The *Success Rate* for a query suggestion method is then defined as the fraction of queries for which the method was successful.

Table 4 summarizes success rates for different query suggestion methods on both the datasets. We show results separately for type-A queries, type-B queries and all queries considered together. We observe that the proposed method (Probabilistic) is able to generate at least one meaningful suggestion for all the queries in all cases. The CompSearch method fails to generate a meaningful suggestion for one type-B query for the TREC dataset. The SimSearch method on the other hand, performs worst and is not able to offer any meaningful suggestion for a large number of type-B queries. The improvements achieved by the Probabilistic and CompSearch methods over SimSearch were significant statistically. This is because the SimSearch method relies on finding phrases that contain the user-submitted partial query as a sub-phrase. However, as discussed in Section 3, this approach suffers from the data sparseness problem. Many times it happens that the query terms typed by the user are not present in the corpus as a phrase even though we may have many documents in the corpus containing all the query terms. CompSearch and Probabilistic methods overcome the problem of data sparseness by using the last query word only to select a set of candidate phrases.

Ubuntu			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	**1.00**	**1.00**	**1.00**
Type-B	0.75	**1.00**$^{\mathbf{s}}$	**1.00**$^{\mathbf{s}}$
Overall	0.875	**1.00**$^{\mathbf{s}}$	**1.00**$^{\mathbf{s}}$
TREC			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	**1.00**	**1.00**	**1.00**
Type-B	0.15	0.95$^{\mathbf{S}}$	**1.00**$^{\mathbf{S}}$
Overall	0.575	0.975$^{\mathbf{S}}$	**1.00**$^{\mathbf{S}}$

Table 4: Success Rate of different query suggestion methods for the two datasets. Superscripts s and S indicate statistically significant improvements over SimSearch with $p < 0.05$ and $p < 0.01$, respectively (one-tailed t-test).

5.2 Quality of Suggestions

The user study described in a previous sub-section provided us with the information that whether a particular query suggestion was meaningful or not. By utilizing these relevance judgments, we now compare different query suggestion mechanisms for their ability to generate meaningful query suggestions. Some examples of suggestions generated by the different methods for some of the test queries are given in Table 5. The table illustrates some cases where the CompSearch and SimSearch methods failed to produce as many meaningful suggestions as our proposed approach. While for SimSearch failure to produce suggestions can be attributed to data sparseness as discussed before, poor performance of CompSearch is due to its limited scope as it only searches for most frequent completions of the last query word.

For a given query, the precision of a query suggestion method is defined as the fraction of suggestions generated that are meaningful. Note that since an exhaustive set of all possible suggestions for a given query is not available, recall cannot be computed. Also, for the query suggestion task, precision is a much more important metric than recall as the number of suggestions that can be offered is limited by the screen space. Hence high precision values at top ranks is favored for this task. Table 6 reports the MAP (mean average precision) values for different methods. The proposed method outperforms the two baseline methods, achieving statistically significant improvements in almost all the cases. We also show the precision values achieved by different methods at different ranks (rank 1 to 10) in Figure 2. We observe that the proposed method outperforms both the baselines methods in all the cases. The high MAP and precision values indicate that the proposed method is able to offer a larger number of meaningful suggestions in comparison to the baselines. Further, consistently high precision values at top ranks ($\approx$ 80% till rank 3) indicate that in general, meaningful suggestions are presented at earlier ranks to the user, which is a very desirable characteristic. Also, the drop in precision values from rank one to last position is much less for the proposed approach in comparison to the baselines, illustrating the *consistency* of our proposed approach. The strengths of the proposed approach are displayed specifically by the results for type-B queries where it comprehensively outperforms both the baselines methods. These

Query = mount			Query = falkland		
SimSearch	**CompSearch**	**Prob**	**SimSearch**	**CompSearch**	**Prob**
mount	mount	mount	falklands	falklands	falklands
mounted	mounted	unable to mount	falkland	falkland	falklands war
mounting	mounting	mount point type	falkland islands	falklanders	falkland islands
mounts	mounts	sudo mount	falklands war		falklands conflict
sudo mount	mountpoint	able to mount	falklands conflict		1982 falklands
unable to mount	mountcifs	mountpoint	1982 falklands		1982 falklands conflict
system mount	mountable	try to mount	falkland islands government		falkland islands government
file system mount	mounter	mount the drive	1982 falklands conflict		falklands war in 1982
mount point type	mountunmount	mount the partition	falkland arms		1982 falklands war
system mount point type	mountpoints	file system mount	falklanders		invasion of the falklands
Query = screen resoluti			**Query = encryption equip**		
screen resolution	screen resolution	screen resolution	<No Suggestions Produced>	encryption equipment	encryption equipment
screen resolutions	screen resolutions	screen change the resolution			encryption digital equipment
preferences screen resolution	screen resolutionrefresh	screen native resolution			encryption office equipment
change screen resolution	screen resolutioni	screen set the resolution			encryption electronic equipment
screen resolution settings	screen resolutionbut	preferences screen resolution			encryption telephone equipment
login screen resolution	screen resolutio	screen correct resolution			encryption equipment and services
screen resolution hi	screen resolution1280	screen resolution and refresh			encryption video equipment
screen resolution issue	screen resolutionsrefresh	screen low resolution			encryption medical equipment
changing screen resolution	screen resolutionsize	screen monitor resolution			encryption transmission equipment
screen resolution preferences	screen resolutiondisplay	screen comparing resolution			encryption original equipment

Table 5: Examples of suggestions generated by different query suggestion methods.

Ubuntu			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	0.4597	0.1638	$\mathbf{0.5022^{C}}$
Type-B	0.2193	0.2309	$\mathbf{0.4746^{SC}}$
Overall	0.3395	0.1974	$\mathbf{0.4884^{SC}}$
TREC			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	0.5429	0.2709	$\mathbf{0.5697^{C}}$
Type-B	0.0614	0.2010	$\mathbf{0.3975^{SC}}$
Overall	0.3022	0.2359	$\mathbf{0.4836^{SC}}$

Table 6: Mean Average Precision (MAP) values achieved by different query suggestion methods for the two datasets. Superscripts S and C indicate statistically significant improvements over SimSearch and CompSearch methods, respectively (one tailed t-test, $p < 0.01$).

Ubuntu			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	3.4308	$\mathbf{3.7779}$	3.5282^{s}
Type-B	2.8877	3.7118	$\mathbf{4.0856^{Sc}}$
Overall	3.1592	3.7448	$\mathbf{3.8069^{S}}$
TREC			
	SimSearch	**CompSearch**	**Probabilistic**
Type-A	4.7781	4.4922	$\mathbf{4.8323^{c}}$
Type-B	1.0497	4.9914	$\mathbf{5.5113^{SC}}$
Overall	2.9139	4.7418	$\mathbf{5.1718^{SC}}$

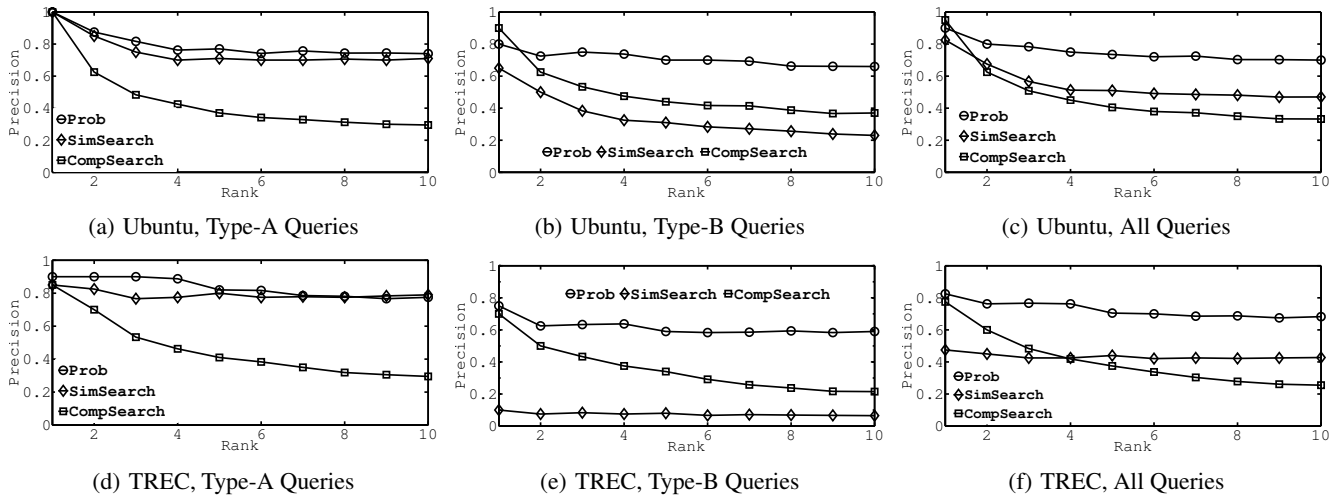
Table 7: Average clarity scores for queries generated by different query suggestion methods for the two datasets. Statistically significant improvements over SimSearch and CompSearch methods are indicated by superscripts S(s) and C(c), respectively using a one-tailed t-test with $p < 0.01$ ($p < 0.05$).

results assert the superiority of the proposed method in offering *contextually relevant* suggestions to the user.

5.3 Retrieval Effectiveness of Suggested Queries

One of the main motivation for query suggestion is to present users with queries that can lead to improved retrieval performance. What is the retrieval effectiveness of queries suggested by different methods and how do the different methods compare with each other? We try to find an answer to this question in this section.

We utilize *query clarity score* as proposed by Cronen-Townsend et al. [12] to measure the retrieval performance of suggested queries. Clarity score has been shown empirically to correlate positively with average precision [12] and is also used as a measure of ambiguity in a query with respect to a collection of documents [11,



(a) Ubuntu, Type-A Queries (b) Ubuntu, Type-B Queries (c) Ubuntu, All Queries

(d) TREC, Type-A Queries (e) TREC, Type-B Queries (f) TREC, All Queries

Figure 2: Precision achieved by different query suggestion methods at different rank positions (rank 1 to rank 10) for Ubuntu and TREC datasets. First and second figures in each row show precision values for type-A and type-B queries respectively. The last figure in each row shows the precision values for all queries taken together. The proposed probabilistic method outperforms the two baseline methods in all cases. Also note that the drop in precision values for the proposed method is much less in comparison with the two baselines.

12]. Specifically, clarity score of a query increases if we add terms that reduce query ambiguity and it decreases on adding terms that make the query more ambiguous. Since we are generating suggestions by adding phrases to the partial user query, we expect the method that is able to select phrases related to the user query to have a higher clarity score. Note that for both the datasets used in this paper, relevance judgments are available only for *original queries* and not for the *suggested queries* that are generated by different query suggestion methods. Once we generate a partial query from an original topic, the suggested queries for that partial query can be different from the original topic (in fact diverse suggestions are desired). Therefore, retrieval effectiveness of the suggested queries cannot be measured by the available relevance judgments. Hence, the need for using a query performance predictor (clarity score).

Clarity score for a query is computed as the Kullback-Leibler divergence between the query language model and the collection language model. Mathematically, clarity score for a query q with respect to a collection of documents $\mathcal{C}$ is given by

$$\text{Clarity}(q, \mathcal{C}) = \sum_{v \in \mathcal{V}} P(v|q) \log_2 \frac{P(v|q)}{P(w|\mathcal{C})}, \qquad (15)$$

where $\mathcal{V}$ is the vocabulary of the collection.

For computing clarity score, we used the Lemur search engine toolkit[3]. The query language model was computed using the relevance model 1 as described by Lavrenko and Croft [21]. For each test query, we computed the clarity scores for all the suggestions generated by a particular query suggestion method and computed the average clarity value for resulting suggestions for the query. This computation is repeated for all the test queries and we report the mean average clarity values achieved by a query suggestion method. The results are summarized in Table 7.

From the table we observe that the proposed Probabilistic approach achieves statistically significant improvements over the baselines in all the cases except for CompSearch method with type-A queries for Ubuntu dataset. Here it is important to note that the SimSearch method failed to generate suggestions for many difficult queries and both the CompSearch and SimSearch methods generated less than 10 suggestions for many queries. On the other hand, the proposed method generated 10 suggestions for all the queries. Thus, the proposed method is able to offer suggestions for a variety of queries as well as it is able to generate suggestions without having an adverse effect on retrieval effectiveness of queries.

[3] `http://www.lemurproject.org/`

6. CONCLUSIONS AND FUTURE WORK

We have shown that meaningful query suggestions can be made in the absence of query logs with an unsupervised probabilistic approach using the occurrence of terms and phrases in a corpus of documents. Experimental results on two different datasets are encouraging and our proposed approach achieved statistically significant improvements over two state-of-the-art baselines. However, there are several aspects that have been out of the scope of this work and offer interesting future research directions, which we discuss below.

1. Our approach is essentially unsupervised and is limited to suggesting queries that are formed by combining a few related phrases from the corpus. Naturally, our system generates some suggestions that are badly formed and hence are not meaningful as search queries, although they may yield results as search queries. One possible future goal would be to ensure that the badly formed combination of phrases are eliminated from the suggestions.
2. Leveraging semantic and ontological information associated with the phrases would be another possible direction. Once the user has typed `hotel`, the system can try to consciously suggest queries such as `hotels in new york`, `hotels in bangalore`, etc. Use of synonyms and synonymous phrases to enable the system to suggest alternatives also needs to be explored.
3. Another interesting aspect that has been out of the scope of this paper is a systematic approach towards diversifying the suggested queries such that not only does the search engine suggest meaningful queries but it also suggests queries that cover a diverse range of concepts and thereby meets the information needs of a broad range of users.

4. In this work, we have primarily focused on the effectiveness of the queries suggested. The target systems for our approach typically have smaller scale datasets and so for that purpose, the efficiency of our algorithm is not critical, but it will be interesting to explore how we can produce similarly effective query suggestions more efficiently than we currently do so that our method can be applied to a relatively larger scale as well.

7. REFERENCES

[1] Solr–Enterprise Search Platform, `http://lucene.apache.org/solr/`.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. *Query Recommendation Using Query Logs in Search Engines*, volume 3268/2004 of *Lecture Notes in Computer Science*, pages 588–596. Springer Berlin / Heidelberg, November 2004.

[3] R. Baraglia, C. Castillo, D. Donato, F. M. Nardini, R. Perego, and F. Silvestri. Aging effects on query flow graphs for query suggestion. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 1947–1950, 2009.

[4] M. Barouni-Ebarhimi and A. A. Ghorbani. A novel approach for frequent phrase mining in web search engine query streams. In *CNSR '07: Proceedings of the Fifth Annual Conference on Communication Networks and Services Research*, pages 125–132, Washington, DC, USA, 2007. IEEE Computer Society.

[5] H. Bast and I. Weber. Type less, find more: Fast autocompletion search with a succinct index. In *SIGIR'06*, pages 364–371, 2006.

[6] H. Bast and I. Weber. The CompleteSearch Engine: Interactive, Efficient, and Towards IR& DB integration. In *CIDR'07*, pages 88–95, 2007.

[7] S. Bhatia and P. Mitra. Adopting inference networks for online thread retrieval. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1300–1305, Atlanta, Georgia, USA, July 11-15 2010.

[8] D. C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Commun. ACM*, 28(3):289–299, 1985.

[9] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *WSCD '09: Proceedings of the 2009 workshop on Web Search Click Data*, pages 56–63, 2009.

[10] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD'08*, pages 875–883, 2008.

[11] S. Cronen-Townsend and W. B. Croft. Quantifying query ambiguity. In *Proceedings of the second international conference on Human Language Technology Research*, pages 104–109, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[12] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 299–306, 2002.

[13] S. Cucerzan and R. W. White. Query suggestion based on user landing pages. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 875–876, 2007.

[14] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 325–332, 2002.

[15] A. Feuer, S. Savev, and J. A. Aslam. Evaluation of phrasal query suggestions. In *CIKM'07*, pages 841–848, 2007.

[16] B. M. Fonseca, P. B. Golgher, B. Pôssas, B. A. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *CIKM'05*, pages 696–703, 2005.

[17] W. Gao, C. Niu, J.-Y. Nie, M. Zhou, J. Hu, K.-F. Wong, and H.-W. Hon. Cross-lingual query suggestion using query logs of different languages. In *SIGIR'07*, pages 463–470, 2007.

[18] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 387–396, 2006.

[19] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall, 2nd edition, 2009.

[20] D. Kelly, K. Gyllstrom, and E. W. Bailey. A comparison of query and term suggestion features for interactive searching. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 371–378, 2009.

[21] V. Lavrenko and W. B. Croft. Relevance based language models. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127, 2001.

[22] Y. Li, B. Wang, S. Xu, P. Li, and J. Li. Querytrans: Finding similar queries based on query trace graph. In *WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 260–263, Washington, DC, USA, 2009. IEEE Computer Society.

[23] H. Ma, M. R. Lyu, and I. King. Diversifying query suggestion results. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[24] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 709–718, 2008.

[25] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 469–478, 2008.

[26] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(1–2):1–174, 2010.

[27] Y. Song and L. wei He. Optimal rare query suggestion with implicit user feedback. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 901–910, 2010.

[28] R. W. White and G. Marchionini. Examining the effectiveness of real-time query expansion. *Inf. Process. Manage.*, 43(3):685–704, 2007.