Research Article Open Access

Gaetano Rossiello*, Annalina Caputo, Pierpaolo Basile, and Giovanni Semeraro

Modeling concepts and their relationships for corpus-based query auto-completion

https://doi.org/10.1515/comp-2019-0015
Received February 20, 2019; accepted July 29, 2019

Abstract: Query auto-completion helps users to formulate their information needs by providing suggestion lists at every typed key. This task is commonly addressed by exploiting query logs and the approaches proposed in the literature fit well in web scale scenarios, where usually huge amounts of past user queries can be analyzed to provide reliable suggestions. However, when query logs are not available, e.g. in enterprise or desktop search engines, these methods are not applicable at all. To face these challenging scenarios, we present a novel corpus-based approach which exploits the textual content of an indexed document collection in order to dynamically generate query completions. Our method extracts informative text fragments from the corpus and it combines them using a probabilistic graphical model in order to capture the relationships between the extracted concepts. Using this approach, it is possible to automatically complete partial queries with significant suggestions related to the keywords already entered by the user without requiring the analysis of the past queries. We evaluate our system through a user study on two different real-world document collections. The experiments show that our method is able to provide meaningful completions outperforming the state-of-the art approach.

Keywords: query auto-completion, information retrieval, information extraction, probabilistic graphical model

***Corresponding Author: Gaetano Rossiello:** IBM Research AI, Thomas J. Watson Research Center, Yorktown Heights, NY, USA. E-mail: gaetano.rossiello@ibm.com
Annalina Caputo: ADAPT centre, School of Computer Science and Statistics, Trinity College Dublin, Ireland. E-mail: annalina.caputo@adaptcentre.ie
Pierpaolo Basile, Giovanni Semeraro: Department of Computer Science, University of Bari Aldo Moro, Bari, Italy. E-mail: pierpaolo.basile@uniba.it, giovanni.semeraro@uniba.it

1 Introduction

Query Auto-Completion (QAC), sometimes also referred to as query suggestion, is one of the early facilities that search engines offer to users in order to make the retrieval task easier, quicker, and more effective. QAC helps users to express their information needs by supplying a list of suggestions at each key typed in the search box. A valuable suggestion list should meet some requirements: (1) it should be available in real-time, for really alleviating the user from the burden of explicitly writing down the information need; (2) it should be relevant, i.e. pertaining the portion of query the user has already entered in (and potentially able to predict the whole information need); (3) it should be diverse, in order to offer different interpretations of the user information need and (4) it should be as specific as possible in order to better discriminate relevant documents.

The problem of completing a partially entered user query is usually tackled by exploiting query logs [1]: Past user queries represent a collection of information needs that have been already fully expressed. In web scale search engines, these approaches show excellent performance since they count on a consistent amount of queries collected by millions of users. Indeed, such a “wisdom of crowds” guarantees a wide topic coverage. Moreover, past queries are usually stored in appropriate data structures, like prefix tries, that ensure efficient response time.

However, there are several contexts where it is difficult to collect an adequate amount of query logs. For instance, enterprise content management systems, personal blogs, or desktop search tools, usually have a very limited query database, therefore in these kinds of environments methods based on query logs are not applicable at all. In order to face this issue, a QAC tool should be able to exploit the collection of documents with the aim of providing suggestions for a given partially typed query. This process could be considered as an inverted question answering system, where queries are automatically generated starting from the available textual content. Although the QAC field counts on an active research community [2], most of the research focuses on query log-based approaches, while only few corpus-based methods have been proposed in the literature [3, 4]. Thus, despite the process of automatically generating queries from unstructured texts may clearly have practical implications in real systems, it remains an open challenge from a research point of view.

Generally speaking, in a corpus-based approach the sequences of terms that form a possible suggestion are extracted from the text of the document collection rather than being fed by previous user formulated queries. This poses two main challenging steps: (1) corpus-based QAC methods have to deal with the automatic generation of *meaningful* pieces of candidate suggestions, which have also to be dynamically generated and supplied in-real time; (2) the proposed completions have to be ranked according to their semantic relevance with respect to the context, which consists of the keywords already keyed in.

We argue that state-of-the-art corpus-based methods present several limitations in addressing both steps. First of all, the suggestion component is limited to naively extract n-grams from the textual content, without performing any linguistic analysis. Usually, this step requires advanced natural language processing techniques to deeply analyze the document collection in order to extract meaningful and self-consistent completions. Moreover, the ranking module relies on probabilistic models where the probabilities of the candidate completions are computed under the term independence assumption with respect to the query context already typed. For these reasons, corpus-based methods may provide meaningless query auto-com-pletions, which could lead to an unsatisfactory user experience.

In this work, we face the challenges of corpus-based QAC by proposing a unified probabilistic framework for indexing, retrieval, and suggestion generation that operates at the bag-of-concept level. We define a concept as a linguistic structure, such as a noun phrase, a named entity, and so on. The idea is that a complex linguistic structure can capture semantics more appropriately than a single keyword. The main contributions of the paper can be summarized as follows:

– We introduce a new suggestion extraction algorithm from textual contents based on backward sequencing of noun phrases in order to extract a set of meaningful candidate query completions at every typed key.
– We define a new probabilistic semantic model that captures, in the same framework, different types of relationships existing between concepts extracted from the text. To this extent, we exploit factor graphs as a way for modeling higher-order term dependencies, where each factor corresponds to a different semantic perspective [5].

In our framework, the candidate completions extracted from the corpus are ranked according to the dependencies between concepts and their relationships with respect to the query context. Moreover, the proposed model generalizes the previous probabilistic models as well as it is easily extensible by adding new prior knowledge coming from external sources. Indeed, one of the main advantages of the factor graph is its capability to model complex joint probabilities by integrating new factor nodes.

The research questions that we want to address with our investigation are:

– **RQ1**: Is the noun phrase extraction component able to generate more accurate candidate query completions?
– **RQ2**: Is the factor graph model able to provide a better candidate query completion rank related to a given context?
– **RQ3**: Is our framework offering a better user experience by generating more meaningful candidate query completions?

We answer these questions through an exhaustive evaluation by comparing it with the state-of-the art corpus-based approach on two different tasks. In the first task, we design a novel dataset that simulates the process of autocompleting a specific query given a partially typed information need. In detail, we model such a process as the task of predicting a Wikipedia page title given its category. The aim of this benchmark is to test the performance of the extraction component and the ranking function independently as well as to tune the model hyper-parameters. However, a synthetic testbed is not suitable to evaluate the quality of the query completions since this task strictly involves the user interaction. Therefore, the second task exploits a human evaluation in order to assess the capability of the method to provide meaningful and relevant completions to a given user query. In particular, we design a user study using two corpora in different domains, one generic and the other more specific. The experimental results on both tasks show that the proposed framework consistently outperforms the baseline with significant improvements.

The paper in organized as follows. We describe the related work in Section 2. Section 3 details the process that extracts the candidate completions as well as the probabilistic graphical model used for their ranking. Section 4 reports the experimental protocol and the discussion about the results. We conclude the remarks in Section 5 by providing ideas for future investigations.

2 Related work

The query auto-completion problem has been extensively investigated over the last past years. For instance, the works in [6–8] propose studies regarding its usage during the information search by focusing on the user experience. From a technical perspective, the proposed methodologies can be organized in two main categories based on the source used to generate the completions: query log-based and corpus-based approaches.

2.1 Query log-based approaches

Most of the approaches proposed in the literature rely on the analysis of query logs. Web scale search engines take advantages from the availability of a large amount of user interactions to develop effective QAC algorithms. Indeed, previously formulated queries are already meaningful, often well-defined, and they can be easily indexed and retrieved. Therefore, the main challenge for these methods is to provide an effective list of suggestions by taking into account and properly managing an impressive amount of information stored in the query logs, such as the popularity [9–14] or click-through and session data [15–20]. Recently, the work in [21–24] models the QAC problem in a learning to rank framework where the completions are ranked according to a decision function which exploits features, such as frequency in the logs, similarity measures computed by a deep neural network, demographic information, and short-term history-based data or homologous queries. A deep analysis of the methods based on the query logs is out-of-the-scope of this work and comprehensive surveys on the relevant literature can be found in [1, 2, 25].

2.2 Corpus-based approaches

The generation of queries directly from the document content solves the problem of auto-completing queries even in absence of query logs. This problem has not been well-studied in the literature and it still remains an open challenge, despite its undeniable benefits in concrete real-world limited domains. In this section we discuss the state-of-the-art techniques proposed in an attempt to solve this task by focusing on their limitations, which we seek to overcome with our method.

The work in [26] is the first to introduce the problem of auto-completing queries in search engines by exploiting document collections. It defines an efficient data structure for a priori indexing of term pairs in order to generate completions in linear time. This method is limited to suggest only one word at time, while our system is able to complete partial queries with phrasal concepts.

The authors of [3, 4, 27] propose probabilistic models for corpus-based query auto-completion. The candidate completions are n-grams extracted from the corpus that match the last partial term of the query. Then, their models rank the suggestion list according to the joint probability computed given the candidate n-grams and the partial user query. However, the use of n-grams to complete a query may provide not coherent, meaningless, or incomplete suggestion, which may cause an unsatisfactory user experience. For example, an n-gram can be a truncated concept since it can terminate with a verb. To face this issue, we propose a novel approach for extracting consistent and meaningful completions. Moreover, the joint probabilities are computed under the term independence assumption, thus meaningless completions, not related to the query context, may be provided. Finally, these systems fail to provide a suggestion list when all the keywords in the partial typed query and the extracted n-grams do not occur together in at least one document. The method proposed in this paper aims at addressing all these limitations.

In the framework proposed in [28] the sequences of bi-grams extracted from the corpus are selected and ranked through several steps by involving different heterogeneous models, such as multinomial, bi-grams and vector space. The different measures are then combined to produce the final score. Since pairs of bi-grams are concatenated during the completion generation, there is no guarantee of obtaining a meaningful suggestion list. Moreover, this framework heavily relies on an encyclopedic external source, like Wikipedia, thus it does not fit well in specific domains. This issue does not affect our method since it is general and can be adopted in every domain.

More similar to a query recommender system, the framework proposed in [29] suggests phrasal concepts for literature search. The relevance of an extracted phrase is computed via language modeling, while its similarity with the input query is computed by the label propagation algorithm. The phrase suggestion component is activated once the user has completely entered either the whole query or a long text, therefore it is not properly considered as an auto-completion method. We adopt the same idea of extracting phrasal concepts in our framework.

Table 1: Examples of query auto-completions. Suggestions are generated from the corpus of Wikipedia abstracts.

micro
micro*biological activity*
micro*array analysis*
micro*politan statistical area*
micro*chip based technology*
micro*soft games studios publishing*
microsoft **wind**
microsoft **wind***ows*
microsoft **wind***ows server*
microsoft **wind***ows operating systems*
microsoft **wind***ows sharepoint services*
microsoft **wind***ows client and server platforms*
microsoft windows **secur**
microsoft windows **secur***ity*
microsoft windows **secur***ity improvements*
microsoft windows **secur***ity vulnerabilities*

3 Methodology

Log-based QAC usually picks candidate suggestions from the queries previously formulated by other users. Thus, the candidate suggestions are already meaningful and usually well-formed. Conversely, in a corpus-based approach the candidate suggestions are dynamically generated by exploiting a document collection. This raises the challenge of generating suggestions that have to be meaningful and correlated with the first part of the query. As a consequence, these approaches require techniques to extract pieces of information from textual content as well as to model and rank them according to the other keywords the user has already typed in.

Table 1 reports some examples of auto-completions provided by our QAC system for different partially typed queries. In the example, the candidate suggestions are automatically generated by indexing the Wikipedia abstracts as corpus. In the first example, the query consists only of the prefix **micro**, therefore no context is provided that can narrow the completions toward the real user intent. In absence of other contextual information, the suggestion list generated for the given prefix should provide several topics in order to cover different user information needs. However, as the user continues typing the query, our system is able to provide suggestions that are suitable completions for the given prefix and, at the same time, that correlate meaningfully with the keywords already keyed in. The last two examples report such scenario. Considering the latter example, the candidate suggestion *security improvements* completes the prefix **secur** and is semantically related to the context "microsoft windows". The proposed QAC method discards those matching suggestions, like *security district*, that do not pertain to the given context. Moreover, it has to be pointed out that our system is able to generate auto-completions for a given context even when that exact sequence of terms does not co-occur in the corpus. For instance, the keywords in the completion sequence *"microsoft windows security improvements"* showed in the last example never occur together in any Wikipedia abstract. This means that simply learning a language model [30] on the textual corpus and applying it as a QAC system is not enough to obtain satisfactory auto-completions, especially for small sized document collections.

This section describes a corpus-based QAC method that addresses the aforementioned challenge. We can now formally define the corpus-based QAC as a probabilistic problem.

Definition 1. *Given:*
- $D = \{d_1, d_2, \ldots, d_n\}$*, a document collection;*
- *Q, the partial query typed by the user. The query Q can be split into the context Q_c, the sequence of completely defined query keywords, and the prefix Q_p, the last partially typed keyword, with $|Q_c| \geq 0$ and $||Q_p|| \geq 1$. $|Q_c|$ denotes the number of keywords in Q_c, while $||Q_p||$ denotes the number of characters in Q_p;*
- *$S = \{s_1, s_2, \ldots, s_k\}$, the set of candidate suggestions extracted from D whose prefix matches Q_p;*

the probability of a candidate suggestion s, given the partial query Q, is defined as follows:

$$P(s|Q_c) \tag{1}$$

In the example "microsoft windows secur" in Table 1, Q_c corresponds to "microsoft windows" ($|Q_c| = 2$) and Q_p to "secur" ($||Q_p|| = 5$), while $S = \{$*secur*ity, *secur*ity improvements, *secur*ity vulnerabilities$\}$. In our problem definition, $||Q_p|| \geq 1$ because the QAC mechanism is triggered when at least one character has been typed by the user. However, the query may not contain complete keywords, thus $|Q_c| \geq 0$. This occurs when the user has just started typing some characters of the prefix, as in the first example of Table 1. In the remainder of this section, we first introduce a technique to extract meaningful pieces of information from a textual collection in order to gather the set S of candidate suggestions. Then, we propose a probabilistic factor graph model which works at a concept level to compute the probability $P(s|Q_c)$ in formula (1). Finally, we describe the framework to dynamically generate a suggestion list given a partial user query.

3.1 Candidate suggestion extraction

As candidate suggestions we adopt noun phrases extracted by a chunker through the analysis of the textual content of a document collection. Noun phrases, which usually embed a noun with pre- and post-modifiers, are a syntactic part of the sentence that is usually involved in the activity described by the verb. The key idea is that a noun phrase represents a self-consistent concept and it tends to be more informative than other types of chunks. For example, in Table 1, all suggestions that complete the last partial terms are noun phrases extracted from Wikipedia abstracts. However, completions based only on noun phrases would not consider those suggestions built upon the single terms that constitute the chunk. For example, the chunk "windows operating system" would complete the prefix "wind" but not "operat", which could be completed by "operating system". To solve this problem, we introduce the concept of *backward n-gram*. A backward n-gram is defined as the set of term sequences $\{w_i \dots w_n |\ 1 \leq i \leq n\}$ that can be built backwards from a given sequence of terms $w_1 w_2 \dots w_n$. We index each backward n-gram that can be built from the noun phrases extracted from a corpus. Thus, for the chunk in the above example, our framework extracts "windows operating system", "operating system" and "system".

3.2 Probabilistic factor graph model

Following Definition (1), we now analyze the probability $P(s|Q_c)$ in the cases of a partially typed query with ($|Q_c| > 0$) and without a context ($|Q_c| = 0$).

3.2.1 Suggestion without context

In absence of query context ($|Q_c| = 0$), the model computes the marginalization of s over the whole corpus. Thus, $P(s|Q_c)$ can be written as:

$$P(s|Q_c) = P(s) = \sum_{d \in D} P(s, d) = \sum_{d \in D} P(s|d)P(d) \tag{2}$$

where s, Q_c and D are introduced in Definition (1). Since $P(d)$ is a constant value for each candidate suggestion s, equation (2) can be approximated as follows:

$$P(s) = \sum_{d \in D} P(s|d)P(d) \approx \sum_{d \in D} P(s|d) \tag{3}$$

We can estimate the probability of a phrase s given a document d by means of its frequency:

$$P(s|d) = \frac{tf(s, d)}{|d|} \tag{4}$$

where $tf(s, d)$ denotes the frequency of the suggestion s in the document d. Finally, by replacing equation (4) in (3), we have:

$$P(s) \approx \sum_{d \in D} \frac{tf(s, d)}{|d|} \tag{5}$$

In absence of dynamic information, the marginalization of each suggestion s can be precomputed at indexing time.

3.2.2 Suggestion with context

When the user provides some context keywords ($|Q_c| > 0$), the context has to be taken into account during the process of suggestion generation. Given a context, the probability of a suggestion can be computed as:

$$P(s|Q_c) = \frac{P(s, Q_c)}{P(Q_c)} = \frac{\sum_{d \in D} P(s, Q_c, d)}{\sum_{s \in S} \sum_{d \in D} P(s, Q_c, d)} \tag{6}$$

Since the computation of $P(s|Q_c)$ on the whole document collection D every time the user types a new character is an infeasible task, we approximate the probability by computing equation (6) on the set of pseudo-relevant documents R retrieved for the query Q_c. The set R is retrieved by a language model with Dirichlet smoothing [31]. We can drop the denominator $P(Q_c)$ since it is invariant for all suggestions s:

$$P(s|Q_c) \approx \frac{\sum_{d \in R} P(s, Q_c, d)}{\sum_{s \in S} \sum_{d \in R} P(s, Q_c, d)} \propto \sum_{d \in R} P(s, Q_c, d) \tag{7}$$

According to equation (7), the conditional probability of the suggestion s given Q_c is approximated by the sum of the joint probability of s, Q_c and d over the set of pseudo-relevant documents R. In order to compute $P(s, Q_c, d)$, our solution exploits the concept representation and the factor graph model introduced in [5] for representing dependencies between query concepts. Given a text X, we define the Bag-of-Concept (BoC) of X, denoted by Σ^X, as the subset of the powerset of X, with the constraint that each element of Σ^X should be a linguistic structure. Each linguistic structure represents a type of concept. In our model, terms and noun phrases are used as linguistic structures. Table

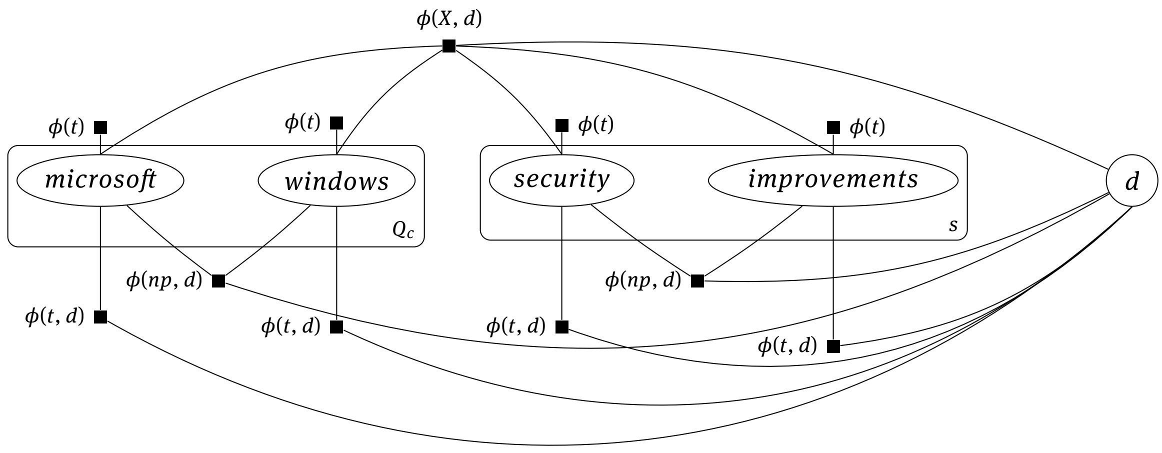


Figure 1: Factor graph structure built on the context Q_c="microsoft windows", the candidate suggestion s="*security improvements*" and a document d for partial query Q_c+Q_p="microsoft windows secur".

2 shows an example of the BoC representation for the text "microsoft windows security improvements" by adopting the linguistic structures used in our model. We build X as the context of the partial query (Q_c) concatenated with the candidate suggestion (s). For example, in the query "microsoft windows secur" in Table 1, "microsoft windows" is the context Q_c, a candidate suggestion s for the prefix "**secur**" may be "security improvements", and X = "microsoft windows security improvements". Moreover, in order to grasp the correlation between concepts in both the candidate suggestion and the query context, it is relevant to model the relationships that exist between concepts in Σ^X. We meet this requirement by exploiting the factor graph model as a probabilistic graphic model. Differently from [5], we apply the factor graph to the QAC task by modelling the relationships between the candidate suggestion, the query context and the document. A factor graph is a bipartite graph with two types of nodes: *variable* nodes, that represent the stochastic variables, and *factor* nodes, that represent the mathematical function "argument of" that relate variables. The strength of factor graphs relies on their capability of expressing a joint probability on a set of variables as the product of factors [32]. In our model, we represent terms as variable nodes, and we build a factor node for each relationship between concepts in Σ^X. A global factor gathers the relationship existing between all the variable nodes. Formally, we define the factor graph $H = (V, E)$, where $V = X \cup \{d\}$ is the set of variables (or *vertices*), and $E = \{(k, d) | k \in \Sigma^X\}$ is the set of hyperedges that link each concept in Σ^X to the document d. Figure 1 shows the factor graph built on Σ^X from the previous example X ="microsoft windows security improvements", where $\phi(t, d)$ represents the factor node that links each term to the document, $\phi(np, d)$ is the factor node associated to noun phrases, and $\phi(X, d)$ is the global factor that captures the relation between all terms with respect to d. Then, we can compute the joint probability in equation (7) as product of factors defined in our model:

$$P(s \mid Q_c) \propto \sum_{d \in R} P(s, Q_c, d) \triangleq \sum_{d \in R} \prod_{e \in E} \phi_e(V_e) \qquad (8)$$

where $V_e \subseteq V$ is the set of vertices linked to the edge e. In addition, we can consider the sum of logarithms in order to avoid underflow errors. However, since factors have to be positive, we introduce the exponential function and we obtain $\log(\exp(\phi)) = \phi$. Thus, the rank of a suggestion s is defined as:

$$P(s \mid Q_c) \overset{rank}{=} \sum_{d \in R} \sum_{e \in E} \lambda(\phi_e) \phi_e(V_e) \qquad (9)$$

where $\lambda(\phi_e)$ is a boosting factor for each ϕ_e.

In our model we define four factors; each factor is associated to a similarity function defined as follows:

Term Document Factor ($\phi(t, d)$) represents the relationship between each term t and a document d, expressed as the term frequency smoothed with Dirichlet priors [33].

Term Collection Factor ($\phi(t)$) represents the relevance of a term within the collection D in a way similar to the inverse document frequency.

Noun Phrases Document Factor ($\phi(np, d)$) is the number of occurrences of a noun phrase np within a document d.

Global Factor ($\phi(X, d)$) represents the relationship between the whole sequence X and the document d. The idea is to promote those suggestions s that appear near the context Q_c in d. We exploit a Gaussian kernel function computed over all pairs of terms in X.

Finally, the introduction of the aforementioned factors in equation (9) produces the following ranking function:

$$P(s \mid Q_c) \overset{rank}{=} \sum_{d \in R} \Bigg[\underbrace{\lambda(t) \sum_{t \in \Sigma^X} \log \left(\frac{tf(t, d) + \mu \frac{tf(t,D)}{|D|}}{\mu + |d|} \right)}_{\phi(t,d)} \underbrace{- \lambda(t) \sum_{t \in \Sigma^X} \log \left(\frac{tf(t, D)}{|D|} \right)}_{\phi(t)} \underbrace{+ \lambda(np) \sum_{np \in \Sigma^X} \log \left(\frac{tf(np, d)}{|d|} \right)}_{\phi(np,d)} \underbrace{+ \lambda(\Sigma) \sum_{t_i \in \Sigma^X} \sum_{i \neq j} \exp \left(- \frac{min\{dist(t_i, t_j, d)\}^2}{2\alpha^2} \right)}_{\phi(X,d)} \Bigg] \quad (10)$$

where tf is the term frequency, μ is the Dirichlet smoothing, α is a parameter of the Gaussian kernel function, and $dist(t_i, t_j, d) = |pos(t_i, d) - pos(t_j, d)|$ is a proximity function based on the position of a term in a document denoted by pos.

3.3 Query auto-completion framework

Our framework relies on two separate data structures: an inverted index and a forward index. The inverted index is required to retrieve the pseudo-relevant document set (R), as well as to store precomputed weights and positions for each concept. Given a document collection, the inverted index is built on every concept type involved in the proposed model, namely terms and noun phrases. Then, a posting list of each term and noun phrase extracted from the document content is created. On the other hand, for each document a forward index is built for storing noun phrases exploited as candidate suggestions at query completion time. Since this index is involved in the prefix completion process, we use a prefix trie data structure to increase the system performance.

```
Input: $query, n, k$
Output: $suggestList$
  $context \leftarrow$ FirstKeywords($query$)
  $prefix \leftarrow$ LastTerm($query$)
  if CountTokens($context$) = 0 then
    $suggestList \leftarrow$ PrecomputatePhrases($prefix$)
  else
    $relDocs \leftarrow$ RelevantDocs($context, n$)
    $phraseList \leftarrow$ ExtractPhrases($prefix, relDocs$)
    for all $phrase \in phraseList$ do
      $score \leftarrow 0$
      for all $doc \in relDocs$ do
        $p \leftarrow$ FactorGraph($context, phrase, doc$)
        $score \leftarrow score + p$
      end for
      $suggest \leftarrow$ Concat($context, phrase$)
      $score \leftarrow score$ / CountTokens($suggest$)
      $suggestList$.Add($suggest, score$)
    end for
  end if
  $suggestList \leftarrow$ SortDesc($suggestList$)
  $suggestList \leftarrow$ TopK($suggestList, k$)
```

Figure 2: Query Auto-Completion Algorithm

Figure 2 describes our algorithm for corpus-based QAC. As first step, the query is split into the context and the last partial term. If the context is empty, the method accesses the forward index to retrieve those noun phrases that match the last term (the prefix). In this case, the score for each candidate suggestion is precomputed at indexing time. In presence of a context, the top n pseudo-relevant documents are retrieved by means of a language model. Then, the method selects only those noun phrases that occur in this set of documents. The algorithm computes a score for each extracted noun phrase that is the sum of probabilities given by the factor graph applied to the noun phrase, the context, and every pseudo-relevant document. The score is normalized taking into account the suggestion length. The normalization demotes the importance of those suggestions composed of many terms. Finally, the suggestion list is sorted in descending order and the top-k suggestions are provided to the user.

Table 2: The Bag-of-Concept representation model for the text “microsoft windows security improvements” using terms and noun phrases as concepts.

Type	Concepts
Terms	[“microsoft”, “windows”, “security”, “improvements”]
Noun Phrases	[“microsoft windows”,“security improvements”]

Table 3: Examples of artificial queries built using Wikipedia categories and page titles using the three different prefix lengths.

Context (category)	Relevant suggestion (page title)	$\|\|Q_p\|\|$	Partial query (Q_c+Q_p)
secession	separatism	1	secession **s**
impressionist composers	claude debussy	1	impressionist composers **c**
philosophical logic	rationality	2	philosophical logic **ra**
texas counties	hemphill county	2	texas counties **he**
semiconductor physicists	herbert kroemer	3	semiconductor physicists **her**
corporate typefaces	bell centennial	3	corporate typefaces **bel**

4 Evaluation

We test the effectiveness of our method exploiting two different tasks. The first is an *in vitro* evaluation that simulates the query completion. We choose to perform such evaluation in order to assess the performance of our methodology on a large number of queries. To the best of our knowledge, this is the first reproducible experiment for the corpus-based QAC task. However, since in this task the queries are automatically generated and do not reflect a real user information need, we decide to validate the proposed method also from a qualitative point of view by means of a user study on two different datasets. Through such experiments we can assess the effectiveness of our method, which exploits concepts and their relationships to generate meaningful and relevant suggestions. In all the experiments we decided not to evaluate the methods when the context is empty (i.e. $|Q_c| = 0$). The reason behind this choice is that, in absence of a context, both our approach and the baseline method base their ranking on the computation of the probability of a suggestion. Hence, the differences are not significant.

4.1 Baseline system and evaluation metrics

We compare the performance of our probabilistic Factor Graph model (**FG**) against a corpus-based probabilistic auto-completion model that makes use of n-grams (**BL**) [3] to generate the candidate suggestions. Since a good QAC system should present the relevant auto-completions at the top of the suggestion list, we evaluate the methods in terms of Mean Reciprocal Rank (MRR) and Success Rate at n (SR@n). MRR gives the mean of the reciprocal of the rank of the correct suggestion, while SR tells us if the relevant completion occurs in the top n ($n \in \{1, 5, 10\}$) suggestions. Moreover, in the user study we consider also the Mean Average Precision (MAP) metric in addition to those used in the Wikipedia title completion task. This choice is due to the fact that each query may have multiple relevant suggestions, hence MAP is a good performance metric that averages precision values taking into account also their rank. The significance of the improvements with respect to the baseline is computed using a non-parametric Randomization test ($p < 0.01$) [34].

4.2 System Setup

The system is written in Java language and it is based on two main data structures, the inverted and forward indexes. In order to get better performance, these indexes are built using Redis[1], a NoSql database in memory. The inverted index is used to find the set of relevant documents R, as defined in Equation (7), whereas the forward index is used to store the candidate suggestions extracted from the corpus. The forward index is implemented with a trie to allow an efficient auto-completion feature. The inverted index has been created with the output of a Natural Language pipeline that exploits OpenNLP[2]. The pipeline is exploited twice: 1) to tokenize the text, remove stop words[3], and stem the keywords; and 2) to extract the noun phrases.

1 https://redis.io/
2 https://opennlp.apache.org/
3 https://code.google.com/p/stop-words/

Also the forward index has been built on noun phrases extracted with OpenNLP. The retrieval model used in the first step of the suggestion algorithm relies on a uni-gram language model. We exploit both indexes also for implementing the probabilistic query suggestion algorithm (**BL**). The implementation of the baseline differs from the original paper [3] with respect to two points: 1) we choose only the n-grams whose prefix matches the partial term of the query and 2) we exploit the same stop-word list of our method. The dimension of the result set $|R|$, which consists of the relevant documents retrieved using only the context as query, has been empirically set to 10 in order to obtain the better trade-off between the effectiveness of the model and the efficiency of the system. All parameters ($\lambda(t)$, $\lambda(np)$, and $\lambda(X)$) introduced in Equation (10) have been set up to 1 in order to give equal importance to all the factors. We have set up $\mu = 800$ by considering the different values evaluated in [31] and choosing that value most suitable to our case, while α has been set up to 175 according to the results reported in [35].

4.3 Wikipedia title completion task

In order to provide a standard and reproducible environment for assessing our method we set up an *in vitro* evaluation exploiting Wikipedia. In absence of a standard benchmark, we decided to build a testbed that simulates the process of a real user that has a clear and non ambiguous information need during the formulation of the query. Since a complete query can always be split in a left and a right part, we treat the left part as the context of our model, while the right part is the *relevant suggestion* unknown to the system. Then, the only relevant candidate suggestion is the one that completes the original user query. Although this assumption is not always true (there could be more suggestions suitable for the same left part) and represents a lower bound of a system performance, it is a quite common protocol in the context of log-based query completion [21–23]. We decided to simulate the user queries with Wikipedia page titles. Then, we exploited Wikipedia twice: as a corpus from which to extract the candidate suggestions and for building the set of queries used in the evaluation. Specifically, we adopted the structured version of Wikipedia, called DBpedia, available in RDF format[4]. In order to build the corpus, we processed all the extended abstracts[5] available in the English version of DBpedia. The dataset comprises 4,636,225 articles, which results in a total of 2,451,109 unique terms and 29,711,747 noun phrases. In this phase we retained only the content of the abstracts, while we discarded the titles: in this way the task of completing a partial title is more challenging since the whole sequence of terms may not occur in the corpus. We built a testbed of artificial queries by exploiting categories associated to Wikipedia articles. For this purpose, we randomly chose 1,000 $\langle article, category \rangle$ pairs from a total of 18,731,756. The category represents the context of a query (Q_c). Then, by splitting the title of the Wikipedia article in a left and a right part, we generate the prefix Q_p (the left part) and its relevant suggestion (the right part). In this way we simulate the process of suggesting a specific query, given a general concept. Since an effective QAC method should correctly auto-complete a partial query using only few characters, we built three independent sets of queries using the selected 1,000 queries and setting the prefix length $||Q_p||$ equal to 1, 2 and 3, respectively. Table 3 shows some examples of queries built for this task. There are two different components that can contribute to the overall performance of the proposed method: the use of phrases for completing the prefix and the ranking model based on probabilistic factor graph. In order to understand the contribution of each component individually and to assess the overall performance of the probabilistic factor graph model, we decided to compare our method and the baseline on the same set of candidate suggestions: noun phrase and n-gram[6]. In this way, we can analysis the impact of the probabilistic factor graph with respect to the probabilistic model employed in [3]. This resulted in four different systems: the baseline (**BL**) and probabilistic factor graph (**FG**) either with n-grams (**ng**) or with noun phrases (**np**). We evaluated the systems on different prefix lengths. The evaluation is conducted considering the list of the top 10 suggestions.

4.3.1 Results and analysis

The scores of our method (**FG**) and that of the baseline (**BL**) using the two different phrase extraction strategies are reported in Table 4 and, for each metric, the best scores are marked in **bold**. All improvements are statistically significant, with $p < 0.01$, except for those of **FG-np** compared with **FG-ng** in the case of $||Q_p|| = 1$. First, our

4 http://data.dws.informatik.uni-mannheim.de/dbpedia/2014/en/long_abstracts_en.nt.bz2

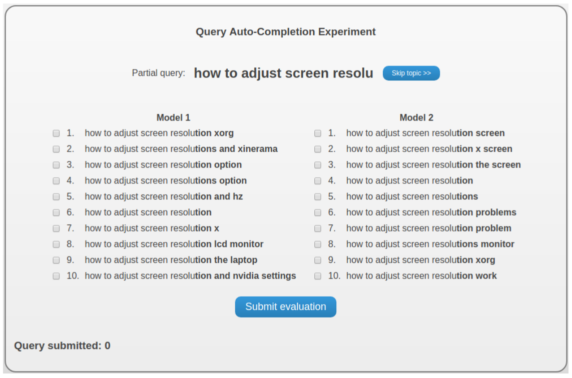
5 The first paragraphs of the Wikipedia articles.

6 The phrase extraction method adopted by the baseline [3]

Table 4: Comparison of the two QAC models (**BL** and **FG**) on Wikipedia abstract dataset using the two different phrase extraction strategies (*n-gram* and *noun phrase*). The scores (in %) are computed using different prefix lengths ($1 \le ||Q_p|| \le 3$).

	$\|\|Q_p\|\| = 1$				$\|\|Q_p\|\| = 2$				$\|\|Q_p\|\| = 3$			
	MRR	**SR@1**	**SR@5**	**SR@10**	**MRR**	**SR@1**	**SR@5**	**SR@10**	**MRR**	**SR@1**	**SR@5**	**SR@10**
BL-ng	05.25	02.93	08.28	12.63	16.43	10.68	23.19	31.94	26.47	19.34	36.11	42.08
BL-np	10.17	04.95	16.87	26.97	29.16	18.15	44.42	58.92	48.03	34.79	66.33	78.43
FG-ng	16.83	08.18	28.35	39.43	33.55	20.96	49.39	65.75	51.31	36.83	70.33	80.29
FG-np	**16.84**	**08.43**	**28.48**	**40.81**	**35.54**	**21.77**	**53.31**	**70.40**	**53.24**	**38.76**	**72.37**	**82.16**

method based on factor graph and noun phrases obtains significant improvements compared to the standard baseline method with respect to all metrics. Interestingly, in the challenging case with a prefix consisting of only a character ($||Q_p|| = 1$), our method achieves a **SR@10** of 40.81, which means that the relevant suggestion is among the first top 10 list in more that 40% of cases. Moreover, as the prefix length grows, the performance gap between our approach and the baseline increases. For instance, with a prefix consisting of only a character ($||Q_p|| = 1$), our method achieves a **MRR** of 16.84, while the standard baseline obtains a **MRR** of 5.25. The delta between the two MRR values is equal to 11.59. Instead, with a prefix of three characters ($||Q_p|| = 3$), the two methods achieve a **MRR** of 53.24 and 26.47, respectively. In this case, the delta is 26.77. This trend is consistent across all the metrics. This proves that our full method (**FG-np**), in presence of a context, requires to type only few characters in order to correctly auto-complete a partial query. The results show that both the factor graph model and the extraction of noun phrases contribute to the improvement of performance. The use of phrases instead of n-gram resulted in better performance in both the baseline and the proposed method. However, this factor contributes only partially to the general better performance of our method. Indeed, the probabilistic factor graph with n-gram (**FG-ng**) outperforms both the baseline with n-gram (**BL-ng**) and the baseline with noun phrases (**BL-fg**). This proves the effectiveness of the probabilistic factor graph method in capturing the semantic similarities between the candidate suggestions and the context. Moreover, it is interesting to point out that when the prefix is short ($||Q_p|| = 1$), the use of noun phrase do not really contribute to significant improvements; these are obtained only at the increase of the prefix length.


Query Auto-Completion Experiment

Partial query: how to adjust screen resolu

Model 1	Model 2
1. how to adjust screen resolution xorg	1. how to adjust screen resolution screen
2. how to adjust screen resolutions and xinerama	2. how to adjust screen resolution x screen
3. how to adjust screen resolution option	3. how to adjust screen resolution the screen
4. how to adjust screen resolutions option	4. how to adjust screen resolution
5. how to adjust screen resolution and hz	5. how to adjust screen resolutions
6. how to adjust screen resolution	6. how to adjust screen resolution problems
7. how to adjust screen resolution x	7. how to adjust screen resolution problem
8. how to adjust screen resolution lcd monitor	8. how to adjust screen resolutions monitor
9. how to adjust screen resolution the laptop	9. how to adjust screen resolution xorg
10. how to adjust screen resolution and nvidia settings	10. how to adjust screen resolution work

Query submitted: 0

Figure 3: A screenshot from the web-app developed for the human assessment. The user has to tick the correct completions given the partial query. The names of the systems are blinded and the two suggestion lists are showed randomly in order to avoid biases.

4.4 User study

The aim of this experiment is to assess the quality of the suggestions produced in response to a real user query in terms of meaningfulness and relevance. The user study follows an evaluation protocol similar to that described in [3]. The evaluation has been performed on two different datasets:

Ubuntuforums.org More than 100,000 discussion threads and 25 queries collected from Ubuntuforums.org [36]. In this evaluation, we exploited the Ubuntuforums.org documents also as a corpus for extracting the suggestions.

Yahoo! Webscope (L13) A selection of 50 queries generated from Yahoo! Webscope (L13)[7]. In this case, we exploited Wikipedia abstracts as a corpus for the extraction of query suggestions, with the same set up explained in Subsection 4.3. From the original sample of 4,496 search queries, we removed those with terms not

7 http://research.yahoo.com/Academic_Relations

Table 5: Comparison between the two methods on three queries extracted from Yahoo! Webscope (L13) dataset; suggestions are generated from the corpus of Wikipedia abstracts.

Query (Q_c+Q_p)	Baseline (**BL**)	Factor Graph (**FG**)
land rover **engi**	land rover engi*neering*	land rover engi*ne*
	land rover engi*ne*	land rover engi*ne plant*
	land rover engi*nes*	land rover engi*ne line*
	land rover engi*nes the engines*	land rover engi*ne production*
	land rover engi*nes have been used for land*	land rover engi*neering centres*
	land rover engi*nes some land*	
	...	
chinese ice **sculp**	chinese ice sculp*ture*	chinese ice sculp*ture*
	chinese ice sculp*tures*	chinese ice sculp*tor*
	chinese ice sculp*ture festival chinese*	chinese ice sculp*ture festival*
	chinese ice sculp*ture festival*	chinese ice sculp*ted vase*
	chinese ice sculp*tures throughout the city*	chinese ice sculp*tor nicolas coustou*
	chinese ice sculp*tures at sun island*	chinese ice sculp*ture decoration technology*
	...	
lexmark fax solutions **softw**		lexmark fax solutions softw*are*
		lexmark fax solutions softw*are integrates*
		lexmark fax solutions softw*are application*
		lexmark fax solutions softw*are company*
		lexmark fax solutions softw*are providers*
		lexmark fax solutions softw*are division*

covered in Wikipedia, those that contained only punctuations, and those made up of only one keyword. We end up with a sample of 2,811 queries from which we randomly picked up the final set.

For each query, we collected at most the top 10 suggestions from each system. Then, 25 assessors selected by colleagues in our department assessed the suggestions produced by the systems in terms of meaningfulness and relevance with respect to both the query context and the prefix. The assessment is a binary value (relevant/non relevant) associated to each suggestion for each query. The presentation order of the two lists of suggestions was randomly selected. Figure 3 shows a screenshot from the webapp we developed for the human assessment. Each query was assessed by three different annotators, and the final judgment was decided by a majority vote. In this evaluation we perform a comparison between our method (**FG**) and the baseline (**BL**) only in their original formulation. This is because in this *in-vivo* experiment we want to evaluate the perception of the users about the two complete systems.

4.4.1 Results and analysis

Table 6 reports the results of the user study conducted on the two datasets: Yahoo! Webscope (L13) and Ubuntuforums.org. This evaluation confirms the overall better performance of the **FG** method. It can be noted that in general the performance of the two systems on Ubuntuforums.org dataset are very low. On average, the baseline returns a relevant suggestion at the fifth position (MRR=0.23), while our method is able to rank a relevant suggestion between the second and the third position (MRR=0.43). These results are corroborated by the SR@1 figures: **BL** and **FG** are able to return a relevant suggestion at the top position for 2 ($SR@1 = 0.08$) and 4 ($SR@1 = 0.16$) out of 25 queries, respectively. A better trend can be observed on the Yahoo! Webscope (L13) dataset. Here, **BL** gives a relevant result at one of the top 3 positions ($MRR = 0.40$), while **FG** at one of the top 2 ($MRR = 0.66$). Our system is able to give a relevant suggestion on 24 out of the 50 queries ($SR@1 = 0.48$), while the baseline is successful only on 10 queries ($SR@1 = 0.20$). On both datasets, MAP values are in line with and corroborate the MRR values. Generally, the performance of both systems improve dramatically if we consider the success rate at the top 5 and top 10 suggestions. However, it is worth to notice that our system is always able

Table 6: Comparison of the two QAC methods on Yahoo! Webscope (L13) and Ubuntuforums.org datasets. Statistically significant improvements ($p < 0.01$) are reported (†).

	Ret	Rel	MAP	MRR	SR@1	SR@5	SR@10
Yahoo! Webscope (L13)							
BL	8.9	2.7	0.36	0.40	0.20	0.70	0.92
FG	5.9	3.1	**0.63**†	**0.66**†	**0.48**†	**0.90**	**1.00**
Ubuntuforums.org							
BL	9.9	1.6	0.20	0.23	0.08	0.40	0.80
FG	9.7	3.7	**0.45**†	**0.43**†	**0.16**	**0.92**†	**1.00**

to give at least one relevant suggestion in the completion list ($SR@10 = 1$), while the baseline fails on 4 queries of the Yahoo! dataset and on 5 queries of the Ubuntuforums.org dataset, respectively. We ascribe the lower performance of both systems on Ubuntuforums.org dataset to the noisy nature of this corpus, which contains posts written by users that did not undergo any review process. Table 6 reports the average number of returned suggestions (Ret) and the average number of relevant suggestions (Rel) for each dataset. From this figures we can see that **FG** is able to give more relevant completions than **BL**, which in general produces more suggestions. The higher number of completions generated by **FG** on the dataset of Ubuntuforums.org is due to the fact that the posts on this dataset are on average longer than Wikipedia abstracts. Then, when the algorithm retrieves the top *n* relevant documents, it has more text from which to extract the candidate suggestions. Finally, Table 5 shows some examples of top six suggestions generated from Wikipedia abstracts for three queries extracted from the Yahoo! Webscope (L13) dataset. From the first query, “land rover engi”, it can be observed that the n-gram model **BL** produces, among interesting suggestions, also meaningless results, like “land rover engines have been used for land”, which contains both verbs and stop-words. On the other hand, by working on noun phrases, the **FG** method is able to produce completions that have a semantics on their own. The use of the backward n-gram mechanism shows its benefits in the second query “chinese ice sculp”. For example, considering the completing noun phrase “sculptor nicolas coustou”, a traditional n-gram mechanism would have produced also “sculptor nicolas” as a candidate, which is an incomplete and redundant suggestion. The last is an example of query for which the baseline did not produce any result. Indeed, **BL** computes the correlation between the context and the suggestion as the number of terms that belong to the intersection between the context and the suggestion. If such a number is zero, the final score will be zero for each suggestion extracted from the corpus. We overcome such limitation by applying the scoring function to all those suggestions extracted from the top *n* relevant documents that match the query context. The set of relevant documents is retrieved by a language model that does not force the AND between the query terms. For this reason our method is able to suggest completions even when not all the query terms co-occur within the same document. Finally, the first two examples show as the **BL** method usually returns more suggestions for each partial query.

5 Conclusions and future work

This paper described a novel corpus-based query auto-completion mechanism based on factor graphs for modeling concepts and their relationships. The proposed method showed several advantages. First, it was always able to generate suggestions for a partial query. Moreover, the evaluations highlighted that in most cases the proposed method provides at least one relevant suggestion. Second, a thoroughly *in vivo* evaluation showed the robustness of our method, which exhibited a stable behavior regardless the length of the prefix. Third, the results of the user study attested the good quality of the completions in terms of meaningfulness and relevance to the query context. The proposed model is flexible and can be easily extended by defining new factor nodes. Thus, we are planning to define new factors in order to infuse further information into the model. Specifically, in our model we can encode session information to predict the user intent in order to provide a personalized service. In this context, we would like to integrate also a suggestion diversification method. Another promising future work is to extend our model by introducing information coming from external sources like knowledge graphs and/or query logs in order to design a hybrid approach. Finally, we plan to introduce embedding representation of the concepts from the text using, for instance, deep learning models [37, 38] in order to add new factors into factor graph to enhance the concept similarity capabilities of the QAC method proposed in this paper.

Acknowledgements This work was supported by the ADAPT Centre for Digital Content Technology, funded under the SFI - Science Foundation Ireland Research Centres Programme (Grant SFI 13/RC/2106) and is co-funded under the European Regional Development Fund and by the EU2020 - European Unions Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No.: EU2020-713567.

References

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

[2] Fei Cai and Maarten de Rijke. A survey of query auto completion in information retrieval. *Foundations and Trends in Information Retrieval*, 10(4):273–363, 2016.

[3] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 795–804. ACM, 2011.

[4] P Deepak, Sutanu Chakraborti, and Deepak Khemani. Query suggestions for textual problem solution repositories. In *ECIR*, pages 569–581. 2013.

[5] Michael Bendersky and W Bruce Croft. Modeling higher-order term dependencies in information retrieval using query hypergraphs. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 941–950. ACM, 2012.

[6] Catherine L Smith, Jacek Gwizdka, and Henry Feild. The use of query auto-completion over the course of search sessions with multifaceted information needs. *Information Processing & Management*, 53(5):1139–1155, 2017.

[7] Xi Niu and Diane Kelly. The use of query suggestions during information search. *Information Processing & Management*, 50(1):218–234, 2014.

[8] Diane Kelly, Amber Cushing, Maureen Dostert, Xi Niu, and Karl Gyllstrom. Effects of popularity and quality on the usage of query suggestions during information search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 45–54. ACM, 2010.

[9] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the 2004 International Conference on Current Trends in Database Technology*, pages 588–596, 2004.

[10] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM, 2006.

[11] M. Barouni-Ebrahimi and Ali A. Ghorbani. A novel approach for frequent phrase mining in web search engine query streams. In *CNSR*, pages 125–132. IEEE Computer Society, 2007.

[12] Yanan Li, Bin Wang, Sheng Xu, Peng Li, and Jintao Li. Querytrans: Finding similar queries based on query trace graph. In *Web Intelligence*, pages 260–263. IEEE, 2009.

[13] Yang Song and Li-wei He. Optimal rare query suggestion with implicit user feedback. In *Proceedings of the 19th international conference on World wide web*, pages 901–910. ACM, 2010.

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

[15] Hao Ma, Haixuan Yang, Irwin King, and Michael R Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 709–718. ACM, 2008.

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

[17] Eugene Kharitonov, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. Intent models for contextualising and diversifying query suggestions. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2303–2308. ACM, 2013.

[18] F. Cai, S. Liang, and M. de Rijke. Prefix-Adaptive and Time-Sensitive Personalized Query Auto Completion. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2452–2466, 2016.

[19] Fei Cai, Ridho Reinanda, and Maarten De Rijke. Diversifying Query Auto-Completion. *ACM Transactions on Information Systems*, 34(4):25:1–25:33, June 2016.

[20] Liangda Li, Hongbo Deng, Jianhui Chen, and Yi Chang. Learning parametric models for context-aware query auto-completion via hawkes processes. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 131–139, New York, NY, USA, 2017. ACM.

[21] Bhaskar Mitra. Exploring session context using distributed representations of queries and reformulations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. ACM, 2015.

[22] Bhaskar Mitra and Nick Craswell. Query auto-completion for rare prefixes. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1755–1758. ACM, 2015.

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

[24] Fei Cai and Maarten de Rijke. Learning from homologous queries and semantically related terms for query auto completion. *Information Processing & Management*, 52(4):628–643, 2016.

[25] Giovanni Di Santo, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Comparing approaches for query autocompletion. In Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 775–778. ACM, 2015.

[26] Holger Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 364–371. ACM, 2006.

[27] David Maxwell, Peter Bailey, and David Hawking. Large-scale generative query autocompletion. In Bevan Koopman, Guido Zuccon, and Mark James Carman, editors, *Proceedings of the 22nd Australasian Document Computing Symposium, ADCS 2017, Brisbane, QLD, Australia, December 7-8, 2017*, pages 9:1–9:8. ACM, 2017.

[28] Meher T. Shaikh, Maria Soledad Pera, and Yiu-Kai Ng. A probabilistic query suggestion approach without using query logs. In *25th International Conference on Tools with Artificial Intelligence*, pages 633–639. IEEE Computer Society, 2013.

[29] Youngho Kim, Jangwon Seo, W. Bruce Croft, and David A. Smith. Automatic suggestion of phrasal-concept queries for literature search. *Information Processing and Management*, 50(4):568–583, 2014.

[30] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[31] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214, April 2004.

[32] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2006.

[33] David J.C. MacKay and Linda C. Bauman Peto. A hierarchical dirichlet language model. *Natural Language Engineering*, 1:1–19, 1994.

[34] Mark D Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 623–632. ACM, 2007.

[35] Yuanhua Lv and ChengXiang Zhai. Positional language models for information retrieval. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 299–306, New York, NY, USA, 2009. ACM.

[36] Sumit Bhatia and Prasenjit 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.

[37] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[38] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, August 2013.