A Generalized Hidden Markov Model with Discriminative Training for Query Spelling Correction

Yanen Li, Huizhong Duan and ChengXiang Zhai

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801
{yanenli2, duan9, czhai}@illinois.edu

ABSTRACT

Query spelling correction is a crucial component of modern search engines. Existing methods in the literature for search query spelling correction have two major drawbacks. First, they are unable to handle certain important types of spelling errors, such as concatenation and splitting. Second, they cannot efficiently evaluate all the candidate corrections due to the complex form of their scoring functions, and a heuristic filtering step must be applied to select a working set of top-K most promising candidates for final scoring, leading to non-optimal predictions. In this paper we address both limitations and propose a novel generalized Hidden Markov Model with discriminative training that can not only handle all the major types of spelling errors, including splitting and concatenation errors, in a single unified framework, but also efficiently evaluate all the candidate corrections to ensure the finding of a globally optimal correction. Experiments on two query spelling correction datasets demonstrate that the proposed generalized HMM is effective for correcting multiple types of spelling errors. The results also show that it significantly outperforms the current approach for generating top-K candidate corrections, making it a better first-stage filter to enable any other complex spelling correction algorithm to have access to a better working set of candidate corrections as well as to cover splitting and concatenation errors, which few existing method in academic literature can correct.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query Alteration*

Keywords

Query Spelling Correction, Generalized Hidden Markov Models, Discriminative Training for HMMs

1. INTRODUCTION

The ability to automatically correct potentially misspelled queries has become an indispensable component of modern search engines. People make errors in spelling frequently. Particularly, search engine users are more likely to commit misspellings in their queries as they are in most scenarios exploring unfamiliar contents. Automatic spelling correction for queries helps the search engine to better understand the users' intents and can therefore improve the quality of search experience. However, query spelling is not an easy task, especially under the strict efficiency constraint. In Table 1 we summarize major types of misspellings in real search engine queries. Users not only make typos on single words, (insertion, deletion and substitution), but can also easily mess up with word boundaries (concatenation and splitting). Moreover, different types of misspelling could be committed in the same query, making it even harder to correct. Unfortunately, no existing query spelling correction approaches in the literature are able to correct all major types of errors, especially for correcting splitting and concatenation errors. To the best of my knowledge, the only work that can potentially address this problem is [24] in which a Conditional Random Field (CRF) model is proposed to handle a broad set of query refinements. However, this work considers query correction and splitting/merging as different tasks, hence it is unable to correct queries with mixed types of errors, such as substitution and splitting errors in one query. In fact splitting and merging are two important error types in query spelling correction, and a major research challenge of query spelling correction is to accurately correct all major types of errors simultaneously. On the other hand, while modern industrial systems such as Google can handle this problem relatively well, it's unclear what they have done, especially what resources they have employed in order to solve the challenges.

Table 1: Major Types of Query Spelling Errors

Type		Example	Correction
In-Word	Insertion	esspresso	espresso
	Deletion	vollyball	volleyball
	Substitution	comtemplate	contemplate
	Mis-use	capital hill	capitol hill
Cross-Word	Concatenation	intermilan	inter milan
	Splitting	power point	powerpoint

Another major difficulty in automatic query spelling correction is the huge search space. Theoretically, any sequence of characters could potentially be the correction of a misspelled query. It is clearly intractable to enumerate and evaluate all possible sequences for the purpose of finding the correct query. Thus a more feasible strategy is to search in a space of all combinations of candidate words that are in a neighborhood of each query word based on editing distance. The assumption is that a user's spelling error of each single word is unlikely too dramatic, thus the correction is most likely in the neighborhood by editing distance. Unfortunately, even in this restricted space, the current approaches still cannot enumerate and evaluate all the candidates because their scoring functions involve complex features that are expensive to compute. As a result, a separate filtering step must first be used to prune the search space so that the final scoring can be done on a small working set of candidates. Take [7] as a two-stage method example, in the first stage, a Viterbi or A* search algorithm is used to generate a small set of most promising candidates, and in the second stage different types of features of the candidates are computed and a ranker is employed to score the candidates. However, this two-stage strategy has a major drawback in computing the complete working set. Since the filtering stage uses a non-optimal objective function to ensure efficiency, it is quite possible that the best candidate is filtered out in the first stage, especially because we cannot afford a large working set since the correction must be done online while a user is entering a query. The inability of searching the complete space of candidates leads to non-optimal correction accuracy.

In this paper, we propose a generalized Hidden Markov Model (gHMM) for query spelling correction that can address deficiencies of the existing approaches discussed above. The proposed gHMM can model all major types of spelling errors, thus enabling consideration of multiple types of errors in query spelling correction. In the proposed gHMM, the hidden states represent the correct forms of words, and the outcomes are the observed (potentially) misspelled terms. In addition, each state is associated with a type, indicating merging, splitting or in-word transformation operation. The proposed HMM is generalized in the sense that it would allow adjustment of both emission probabilities and transition probabilities to accommodate the non-optimal parameter estimation. Unfortunately, such an extension of HMM makes it impossible to use a standard EM algorithm for parameter estimation. To solve this problem, we propose a perceptron-based discriminative training method to train the parameters in the HMM.

Moreover, a Viterbi-like search algorithm for top-K paths is designed to efficiently obtain a small number of highly confident correction candidates. This algorithm can handle splitting/merging of multiple words. It takes into account major types of local features such as error model, language model, and state type information. The error model is trained on a large set of query correction pairs from the web. And web scale language model is obtained by leveraging the Microsoft Web N-gram service [1].

We conducted extensive evaluation on our proposed gHMM. For this purpose, we have constructed a query correction dataset from real search logs, which has been made publicly available. Experimental results verify that the gHMM can effectively correct all major types of query spelling errors. It also reveal that the gHMM can run as efficient as the common used noisy channel model, while it achieves much better results for obtaining the candidate space of query corrections. Therefore, in addition to being used as standing alone query correction module, the proposed gHMM can also be used as a more effective first-stage filtering module to more effectively support any other complicated scoring functions such as those using complex global features.

2. RELATED WORK

Spelling correction has a long history [8]. Traditional spellers focused on dealing with non-word errors caused by misspelling a known word as an invalid word form. A common strategy at that time was to utilize a trusted lexicon and certain distance measures, such as Levenshtein distance [13]. The size of lexicon in traditional spellers is usually small due to the high cost of manual construction of lexicon. Consequently, many valid word forms such as human names and neologisms are rarely included in the lexicon. Later, statistical generative models were introduced for spelling correction, in which the error model and n-gram language model are identified as two critical components. Brill and Moore demonstrated that a better statistical error model is crucial for improving a speller's accuracy [3]. But building such an error model requires a large set of manually annotated word correction pairs, which is expensive to obtain. Whitelaw *et al.* alleviated this problem by leveraging the Web to automatically discover the misspelled/corrected word pairs [12].

With the advent of the Web, the research on spelling correction has received much more attention, particularly on the correction of search engine queries. Many research challenges are raised, which are non-existent in traditional settings of spelling correction. More specifically, there are many more types of spelling errors in search queries, such as misspelling, concatenation/splitting of query words, and misuse of legitimate yet inappropriate words. Research in this direction includes utilizing large web corpora and query log [4, 5, 2], training phrase-based error model from clickthrough data [10] and developing additional features [7]. However, two important challenges are under addressed in these approaches, i.e., correcting splitting and concatenation errors, and ensuring complete search in the candidate space to evaluate an effective scoring function.

Query alteration/refinement is a broader topic which naturally subsumes query spelling correction. Beside correcting the misspelled query, query alteration/refinement also need to modify the ineffective query so that it could be more suitable for the search intent. For this purpose, many research topics have been studied. Query expansion expands the query with additional terms to enrich the query formulation [14, 15, 16]. Query segmentation divides a query into semantically meaningful sub-units [17, 18]. Other query reformulation methods intend to replace the inappropriate query terms with effective keywords to bridge the vocabulary gaps [19]. Particularly, there is research attempt [24] to use a unified model to do a broad set of query refinements such as correction, segmentation and even stemming. However, it treats query correction and splitting/merging as separate tasks, which is not true for real world queries. Also, it has very limited ability for query correction. For example, it only allows one letter difference in deletion/insertion/substitution errors.

Query spelling correction also shares similarities with many other NLP tasks, such as speech recognition and machine translation. In many of these applications, HMM has been found very useful [21, 20]. Our generalized HMM model for query spelling correction is novel in that it models all the major types of misspellings in a unified framework. A discrete training method is also proposed to train the parameters in the model. It is demonstrated that the gHMM model we use is very effective for the task of query spelling correction.

3. PROBLEM SETUP AND CHALLENGES

Formally, let Σ be the alphabet of a language and $L \subset \Sigma^+$ be a large lexicon of the language. We define the query spelling correction problem as:

Given a query $q \in \Sigma^+$, generate top-K most effective corrections $Y = (y^1, y^2, ..., y^k)$ where $y^i \in L^+$ is a candidate correction, and Y is sorted according to the probability of y^i being the correct spelling of the target query.

It is worth noting that from a search engine perspective, the ideal output Y should be sorted according to the probability of y^i retrieving the most satisfying results in search. However, in practice it is very difficult to measure the satisfaction as unlike in *ad hoc* retrieval where the query is given in its correct form, here the real query is unknown. As a result, different corrections could simply lead to queries with different meanings and it would be very subjective to determine which query actually satisfies the user. In this paper, we are mostly concerned with the lexical and semantic correctness of queries with the assumption that correction of mis-spelled query terms most likely would lead to improved retrieval accuracy.

The problem of query spelling correction is significantly harder than the traditional spelling correction. Previous researches show that approximately 10-15% of search queries contain spelling errors [5]. First, it is difficult to cover all the different types of errors. The spelling errors generally fall into one of the following four categories: (1) in-word transformation, e.g. insertion, deletion, misspelling of characters. This type of error is most frequent in web queries, and it is not uncommon that up to 3 or 4 letters are misspelled; (2) mis-use of valid word, e.g. "persian golf" → "persian gulf". It is also a type of in-word tranformation errors; (3) concatenation of multiple words, e.g. "unitedstatesofamerica" → "united states of america"; (4) splitting a word into parts, e.g. "power point slides" → "powerpoint slides". Among all these types, the splitting and concatenation errors are especially challenging to correct. Indeed, no existing approaches in the academic literature can correct these two types of errors. Yet, it's important to correct all types of errors because users might commit different types of errors or even commit these errors at the same time. A main goal of this work is to develop a new HMM framework that can model and correct all major types of errors including splitting and concatenation.

Second, it is difficult to ensure complete search of all the candidate space because the candidate space is very large. The existing work addresses this challenge by using a two-stage method, which searches for a small set of candidates with simple scoring functions and do re-ranking on top of these candidates. Unfortunately, the simple scoring function used in the first stage cannot ensure that the nominated candidate corrections in the first stage always contain the best correction, thus no matter how effective the final scoring function is, we may miss the best correction simply because of the use of two separate stages. In this paper, we address this challenge by developing a generalized HMM that can both be efficiently scored to ensure complete search in the candidate space and accurately correct all types of errors in a unified way.

4. A GENERALIZED HMM FOR QUERY SPELLING CORRECTION

Our algorithm accepts a query as input, and then generates a small list of ranked corrections as output by a generalized Hidden Markov Model (gHMM). It is trained by a discriminative method with labeled spelling examples. Given a query, it scores candidate spelling corrections in a one-stage fashion and outputs the top-K corrections, without using a re-ranking strategy. Other components of our algorithm include a large clean lexicon, the error model and the language model. In this section we will focus on the gHMM model structure, the discriminative training of it, as well as the efficient computation of spelling corrections.

4.1 The gHMM Model Structure

We propose a generalized HMM Model to model the spelling correction problem. We call it a generalized HMM because there are several important differences between it and the standard HMM model which will be explained later. Without loss of generality, let an input query be $q = q_{[1:n]}$ and a corresponding correction be $y = y_{[1:m]}$ where n, m are the length of the query and correction, which might or might not be equal. Here we introduce hidden state sequence $z = z_{[1:n]} = (s_1, s_2, ..., s_n)$ in which z and q have the same length. An individual state s_i is represented by a phrase corresponding to one or more terms in correction $y_{[1:m]}$. Together the phrase representing z is equal to y. Therefore, finding best-K corrections $Y = (y^1, y^2, ..., y^k)$ is equivalent to finding best-K state sequences $Z = (z^1, z^2, ..., z^k)$. In addition, there is a type t associated with each state, indicating the operation such as substitution, splitting, merging *etc.* Also, in order to facilitate the merging state we introduce a NULL state. The NULL state is represented by an empty string, and it doesn't emit any phrase. There can be multiple consecutive NULL states followed by a merging state. Table 2 summarizes the state types and the spelling errors they correspond to. Having the hidden states defined, the hypothesized process of observing a mis-spelled query is as follows:

1. sample a state s_1 and state type t_1 from the state space Ω and the type set T;
2. emit a word in q_1, or empty string if the s_1 is a NULL state according to the type specific error model;
3. transit to s_2 with type t_2 according to the state transition distribution, and emit another word, or multiple words in $q_{[1:n]}$ if s_2 is a merging state;
4. continue until the whole (potentially) mis-spelled query q is observed.

Figure 1 illustrates our gHMM model with a concrete example. In this example, there are three potential errors with different error types, e.g. "goverment" → "government" (substitution), "home page" → "homepage" (splitting), "illinoisstate" → "illinois state" (concatenation). The state path shown in Figure 1 is one of the state sequences

Table 2: State Types in gHMM

State Type	Operation	Spelling Errors
In-word Transformation	Deletion Insertion Substitution	Insertion Deletion Substitution
Mis-use	Transformation	Word Mis-use
Merging	Merge Multiple Words	Splitting
Splitting	Split one Word to Multiple Words	Concatenation

that can generate the query. Take state s_3 for example, s_3 is represented by phrase *homepage*. Since s_3 is a merging state, it emits a phrase *home page* with probability $P(home\ page|homepage)$. And s_3 is transited from state s_2 with probability $P(s_3|s_2)$. With this model, we are able to come up with arbitrary corrections instead of limiting ourselves to an incomprehensive set of queries from query log. By simultaneously modeling the misspellings on word boundaries, we are able to correct the query in a more integrated manner.

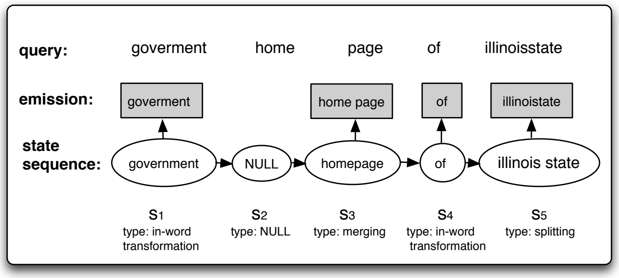

Figure 1: Illustration of the gHMM Model

4.2 Generalization of HMM Scoring Function

For a standard HMM [23], let $\theta = \{A, B, \pi\}$ be the model parameters of the HMM, representing the transition probability, emission probabilities and initial state probabilities respectively. Given a list of query words $q_{[1:n]}$ (obtained by splitting empty spaces), the state sequence $z^* = (s_1^*, s_2^*, ..., s_n^*)$ that best explains $q_{[1:n]}$ can be calculated by:

$$z^* = \arg\max_z P(z|q_{[1:n]}, A, B, \pi) \tag{1}$$

However, theoretically the phrase in a state can be chosen arbitrarily, so estimating $\{A, B, \pi\}$ is such a large space is almost impossible in the standard HMM framework. In order to overcome this difficulty, the generalized Hidden Markov Model proposed in this work generalizes the standard HMM as follows: (1) gHMM introduces state type for each state, which indicates the correction operations and can reduce the search space effectively; (2) it adopts feature functions to parameterize the measurement of probability of a state sequence given a query. Such treatment can not only map the transition and emission probabilities to feature functions with a small set of parameters, but can also add additional feature functions such as the ones incorporating state type information. Another important benefit of the feature function representation is that we can use discriminative training on the model with labeled spelling corrections, which will lead to a more accurate estimation of the parameters.

Formally, in our gHMM model, there is an one-to-one relationship between states in a state sequence and words in the original query. For a given query $q = q_{[1:n]}$ and the sequence of states $z = (s_1, s_2, ..., s_n)$, we define a context h_i for every state in which an individual correction decision is made. The context is defined as $h_i =< s_{i-1}, t_{i-1}, s_i, t_i, q_{[1:n]} >$ where $s_{i-1}, t_{i-1}, s_i, t_i$ are the previous and current state and type decisions and $q_{[1:n]}$ are all query words.

The generalized HMM model measures the probability of a state sequence by defining feature vectors on the context-state pairs. A feature vector is a function that maps a context-state pair to a d-dimensional vector. Each component of the feature vector is an arbitrary function operated on (h, z). Particularly, in this study we define 2 kinds of feature vectors, one is $\phi_j(s_{i-1}, t_{i-1}, s_i, t_i), j = 1...d$, which measures the interdependency of adjacent states. We can map this function to a kind of transition probability measurement. The other kind of feature function, $f_k(s_i, t_i, q_{[1:n]}), k = 1...d'$ measures the dependency of the state and its observation. We can consider it as a kind of emission probability in the standard HMM point of view. Such feature vector representation of HMM is introduced by Collins [22] and successfully applied to the POS tagging problem.

Specifically, we have designed several feature functions as follows: we define a function of $\phi(s_{i-1}, t_{i-1}, s_i, t_i)$ as

$$\phi_1(s_{i-1}, t_{i-1}, s_i, t_i) = logP_{LM}(s_i|s_{i-1}, t_{i-1}, t_i) \tag{2}$$

to measure the language model probabilities of two consecutive states. Where $P_{LM}(s_i|s_{i-1})$ is the bigram probability calculated by using Microsoft Web N-gram Service [1]. The computation of $P_{LM}(s_i|s_{i-1})$ may depend on the state types, such as in a merging state.

We have also defined a set of functions in the form of $f_k(s_i, t_i, q_{[1:n]})$, which are dependent on the query words and state type, measuring the emission probability of a state. For example, we define

$$f_1(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s_i, q_i) & \text{if } q_i \text{ is in-word transformed} \\ & \text{to } s_i \textbf{ and } q_i \notin \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

as a function measuring the emission probability given the state type is in-word transformation and q_i is out of dictionary. e.g. "goverment" → "government". $P_{err}(s_i, q_i)$ is the emission probability computed by an error model which measures the probability of mis-typing "government" to "goverment". (see Section 5.2).

$$f_2(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s_i, q_i) & \text{if } t_i \text{ is splitting} \\ & \textbf{and } q_i \in \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

to capture the emission probability if the state is of splitting type and q_i is in dictionary. e.g. "homepage" → "home page".

$$f_3(s_i, t_i, q_{[1:n]}) = \begin{cases} logP_{err}(s, q_i) & \text{if } t_i \text{ is Mis-use} \\ & \textbf{and } q_i \in \text{Lexicon } L \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

to get the emission probability if a valid word is transformed to another valid word.

Note that in Equation (3), (4), and (5), we use the same error model $P_{err}(s_i, q_i)$ (see Section 5.2 for detail) to model the emission probabilities from merging, splitting errors *etc.* in the same way as in-word transformation errors. However we assign different weights to the transformation probabilities resulted from different error types via discriminative training on a set of labeled query-correction pairs.

Overall, we have designed a set of feature functions that are all relied on local dependencies, ensuring that the top-K state sequences can be computed efficiently by Dynamic Programming.

After establishing the feature vector representation, the log-probability of a state sequence and its corresponding types $logP(z, t|q_{[1:n]})$ is proportional to:

$$\begin{aligned} Score(z,t) = & \sum_{i=1}^{n}\sum_{j=1}^{d}\lambda_j\phi_j(s_{i-1}, t_{i-1}, s_i, t_i) \quad (6) \\ & + \sum_{i=1}^{n}\sum_{k=1}^{d'}\mu_k f_k(s_i, t_i, q_{[1:n]}) \end{aligned}$$

where λ_j, μ_k are the component coefficients needed to be estimated. And the best state sequence can be found by:

$$z^*t^* = \arg\max_{z,t} Score(z,t) \quad (7)$$

Note that the form of $Score(z, t)$ is similar to the objective function of a Conditional Random Field model [26], but with an important difference that there is no normalization terms in our model. Such difference also enables the efficient search of top-K state sequences (equivalent to top-K corrections) using Dynamic Programming, which will be introduced shortly.

4.3 Discriminative Training

Motivated by ideas introduced in [22], we propose a perceptron algorithm to train the gHMM model. To the best of our knowledge, this is the first attempt to use discriminative approach to train a HMM on the problem of query spelling correction. Now we describe how to estimate the parameters λ_j, μ_k from a set of <query, spelling correction> pairs. The estimation procedure follows the perceptron learning framework. Take the λ_j for example. We first set all the λ_j at random. For each query q, we search for the most likely state sequence with types $z^i_{[1:n_i]}, t^i_{[1:n_i]}$ using the current parameter settings. Such search process is described in Algorithm 2 by setting $K = 1$. After that, if the best decoded sequence is not correct, we update λ_j by simple addition: we promote the amount of λ_j by adding up ϕ_j values computed between the query and labeled correction y', and demote the amount of λ_j by the sum of all ϕ_j values computed between the query and the top-ranked predictions. We repeat this process for several iterations until converge. Finally in step 11 and 12, we average all $\lambda^{o,i}_j$ in each iteration to get the final estimate of λ_j, where $\lambda^{o,i}_j$ is the stored value for the parameter λ_j after i's training example is processed in iteration o. Similar procedures can apply to μ_k. The detailed steps are listed in Algorithm *1*. Note that in step 7 and 8 the feature functions $\phi_j(q^i, y'^i, t'^i)$ and $f_k(q^i, y'^i, t'^i)$ depend on unknown types t'^i that are inferred by computing the best word-level alignment between q^i and y'^i.

Algorithm 1: Discriminative Training of gHMM

```
   input : A set of <query, spelling correction> pairs
           $q^i_{[1:n_i]}, y'^i_{[1:m_i]}$ for $i = 1...n$
   output: Optimal estimate of $\hat{\lambda}_j, \hat{\mu}_k$, where
           $j \in \{1, ..., d\}, k \in \{1, ..., d'\}$
1  Init Set $\hat{\lambda}_j, \hat{\mu}_k$ to random numbers;
2  for $o \leftarrow 1$ to $O$ do
3      for $i \leftarrow 1$ to $n$ do
           /* identify the best state sequence and the
              associated types of the i'th query with the
              current parameters via Algorithm 2:        */
4          $z^i_{[1:n_i]}, t^i_{[1:n_i]} = \arg\max_{u_{[1:n_i]}, t_{[1:n_i]}} Score(u, t)$
           /* where $u_{[1:n_i]} \in \mathbb{S}^{n_i}$, $\mathbb{S}^{n_i}$ is all possible state
              sequences given $q^i_{[1:n_i]}$                    */
5          if $z^i_{[1:n_i]} \neq y'^i_{[1:m_i]}$ then
6              update and store every $\lambda_j, \mu_k$ according to:
7              $\lambda_j = \lambda_j + \sum_{i=1}^{n_i}\phi_j(q^i, y'^i, t'^i) - \sum_{i=1}^{n_i}\phi_j(q^i, z^i, t^i)$
8              $\mu_k = \mu_k + \sum_{i=1}^{n_i} f_k(q^i, y'^i, t'^i) - \sum_{i=1}^{n_i} f_k(q^i, z^i, t^i)$
9          else
10             Do nothing
   /* Average the final parameters by:                   */
11 $\hat{\lambda}_j = \sum_{o=1}^{O}\sum_{i=1}^{n}\lambda^{o,i}_j / nO$, where $j \in \{1, ..., d\}$
12 $\hat{\mu}_k = \sum_{o=1}^{O}\sum_{i=1}^{n}\mu^{o,i}_k / nO$, where $k \in \{1, ..., d'\}$
13 return parameters $\hat{\lambda}_j, \hat{\mu}_k$;
```

This discriminative training algorithm will converge after several iterations. Here we state the following theorem on its convergence:

Theorem 1: For any training example (q^i, y'^i) which is separable with margin δ, then for Algorithm 1:

$$\text{Number of mistakes} \leq \tfrac{R^2}{\delta^2}$$

where R is a constant satisfying

$\forall i, \forall z \in \{G(q^i) - \{y'^i\}\} : ||\Phi(q^i, y'^i, t'^i) - \Phi(q^i, z^i, t^i)|| \leq R,$

where $G(q^i)$ is a set of predictions generated by Algorithm 2, and the Φ functions can be calculated from *Eq.* (6) by ignoring λ and μ. The proof of **Theorem 1** is not shown due to the space limitation.

4.4 Query Correction Computation

Once the optimal parameters are obtained by the discriminative training procedure introduced above, the final top-K corrections can be directly computed, avoiding the need for a separate stage of candidate re-ranking. Because the feature functions are only relied on local dependencies, it enables the efficient search of top-K corrections via Dynamic Programming. This procedure involves three major steps: (1) candidate states generation; (2) score function evaluation; (3) filtering.

At the first step, for each word in query q, we generate a set of state candidates with types. The phrase representations in such states are in Lexicon L and within editing distance δ from the query word. Then a set of state sequences are created by combining these states. In addition, for each state sequence we have created, we also create another state sequence by adding a NULL state at the end, facilitating a (potential) following merging state. It is important to note that if the δ is too small, it will compromise the final results due to the premature pruning of state sequences. In this work $\delta = 3$ is chosen in order to introduce adequate possible state sequences.

At the score function evaluation step, we update the scores for each state sequence according to *Eq.* (6). The evaluation is different for sequence with different ending state types. Firstly, for a sequence ending with a NULL state, we don't evaluate the scoring function. Instead, we only need to keep track of the state representation of its previous state. Secondly, for a sequence ending with a merging state, it merges the previous one or more consecutive NULL states. And the scoring function takes into account the information stored in the previous NULL states. For instance, to $\phi_1(s_{i-1}, t_{i-1} = NULL, s_i, t_i = merging)$, we have

$$\phi_1(s_{i-1}, \text{NULL}, s_i, \text{merging}) = logP_{LM}(s_{i-2}|s_i) \tag{8}$$

i.e. skipping the NULL state and pass the previous state representation to the merging state. In this way, we can evaluate the scoring function in multiple consecutive NULL states followed by a merging state, which enables the correction by merging multiple query words. Thirdly, for a sequence ending with a splitting state, the score is accumulated by all bigrams within the splitting state. For example,

$$\begin{aligned}&\phi_1(s_{i-1}, t_{i-1}, s_i, t_i = splitting) \qquad (9)\\ &= logP_{LM}(w_1|s_{i-1}) + \sum_{j=1}^{k-1} logP_{LM}(w_{i+1}|w_i)\end{aligned}$$

where $s_i = w_1 w_2 ... w_k$. On the other hand, the evaluation of $f_k(s_i, t_i, q_{[1:n]})$ is easier because it is not related to previous states. The error model from the state representation to the query word is used to calculate these functions.

At the final step, we filter most of the state sequences and only keep top-K best state sequences in each position corresponding to each query word. In sum, we have proposed and implemented an algorithm via Dynamic Programming (see Algorithm 2) for efficiently computing top-K state sequences (corrections). If there are n words in a query, and the maximum number of candidate states for each query word is M, the computational complexity for finding top-K corrections is $O(n \cdot K \cdot M^2)$.

5. OTHER COMPONENTS

Besides the gHMM model and its discriminative training, our spelling correction algorithm also relies on other important components, such as a large trusted lexicon, the error model, and the language model. In this section we will describe these components.

Algorithm 2: Decoding Top-K Corrections

```
input  : A query $q_{[1:n]}$, parameters $\vec{\lambda}$, $\vec{\mu}$
output: top $K$ state sequences with highest likelihood
/* $Z[i, s_i]$: top K state sequences for sub-query $q_{[1:i]}$
   that ending with state $s_i$. For each $z \in Z[i, s_i]$,
   phrase denotes the representation and score
   denotes the likelihood of $z$ given $q_{[1:i]}$.          */
/* $Z[i]$: top state sequences for all $Z[i, s_i]$.          */
1  Init $Z[0] = \{\}$
2  for $i \leftarrow 1$ to $n$ do
       /* for term $q_i$, get all candidate states          */
3      $S \leftarrow s_i, \forall s_i : edit\_dist(s_i, q_i) \le \delta$, $s_i$ has type $s_i.type$
4      for $s_i \in S$ do
5          for $z \in Z[i-1]$ do
6              $a \leftarrow$ new state sequence
7              $a.phrase \leftarrow z.phrase \cup \{s_i\}$
8              update $a.score$ according to $s_i.type$ and Eq.
               (6), Eq. (8) and Eq. (9)
9              $Z[i, s_i] \leftarrow a$
           /* delay truncation for $NULL$ states            */
10         if $s_i.type \ne NULL$ and $i \ne n$ then
11             sort $Z[i, s_i]$ by score
12             truncate $Z[i, s_i]$ to size $K$
13 sort $Z[n]$ by score
14 truncate $Z[n]$ to size $K$
15 return $Z[n]$;
```

5.1 A Large-scale Trusted Lexicon

As mentioned in Section 3, how to define a proper lexicon L is important to a successful speller. We find that with a clean vocabulary, it will significantly improve the performance of spelling correction. However, to obtain such a clean vocabulary is usually difficult in practice. To do this, we make use of the Wikipedia data. Particularly, we select the top 2 million words from Wikipedia by their word frequencies, and automatically curate the obtained words by removing those frequent but illegitimate words from the vocabulary. This curate process involves checking if the word appears in the title of a Wikipedia article, comparing the bigram probability of other words etc. Finally we obtained 1.2 million highly reliable words in the vocabulary. Please note that all words in this vocabulary are unigrams; numbers and punctuations are not included. And no stemming and other forms of modification are conducted.

5.2 Error Model Training

Error Model. The feature functions $f_k()$ depend on an error model, which measures the probability that one word is misspelled into another. Previous studies have shown that a weighted editing distance model trained with a sufficient large set of correction pairs could achieve a comparable performance with a sophisticated n-gram model [6]. Meanwhile, a higher order model has greater tendency to overfit if the training data is not large enough. Given these considerations, we adopt the Weighted Edit Distance (WED) to estimate the error model. More specifically, we first compute the best character-level alignment of two words, and the WED between these two words is the summation of the WED of all aligned character pairs. We model the character-level WED as the character transformation probability. In addition, null character is included in the vocabulary to accomodate the insertion and deletion operations. In order to compute such a probability, a large set of query-correction pairs is obtained by leveraging the spelling services from Google and Bing; and then this probability is estimated as the expected number of transformation from one character to another in these aligned pairs. The training queries are from the MSN search query log released by the Microsoft Live Labs in 2006 (6.5 million queries). They are submitted to the spelling ser- vices, and the corrections are recorded once consensus is reached.

5.3 Use of Web N-gram Model

Another important factor in selecting and ranking the correction candidates is the prior probability of a correction phrase. It represents our prior belief about how likely a query will be chosen by the user without seeing any input from the user. In this work we make use of the Web n-gram service provided by Microsoft [1]. Web n-gram model intends to model the n-gram probability of English phrases with the parameters estimated from the entire Web data. It also differentiates the sources of the data to build different language models from the title, anchor text and body of Web pages, as well as the queries from query log. In our study, we find the **title** model is the most effective for query spelling correction. We hypothesize that this may be because the training data for query model is much noisier. Particularly, misspelled queries may be included in the training data, which makes it less effective for the task of spelling correction. Despite trained with the Web data, Web n-gram model may also suffer from data sparseness in higher order models. To avoid this issue, we make use of the **bigram** model in building our spelling system.

6. EXPERIMENTS AND RESULTS

In order to test the effectiveness and efficiency of our proposed gHMM model, in this section we conduct extensive experiments on two web query spelling datasets. We first introduce the datasets, and describe the evaluation metrics we use for evaluation. Then we compare our model with other baselines in terms of accuracy and runtime.

6.1 Dataset Preparation

The experiments are conducted on two query spelling correction datasets. One is the TREC dataset based on the publicly available TREC queries (2008 Million Query Track). This dataset contains 5892 queries and the corresponding corrections annotated by the MSR Speller Challenge [1] organizers. There could be more than one plausible corrections for a query. In this dataset only 5.3% of queries are judged as misspelled.

We have also annotated another dataset that contains 4926 MSN queries, where for each query there is at most one correction. Three experts are involved in the annotation process. For each query, we consult the speller from two major search engines (i.e. Google and Bing). If they agree on the returned results (including the case if the query is just unchanged), we take it as the corrected form of the input query. If the results are not the same from the two, as least one human expert will manually annotate the most likely corrected form of the query. Finally, about 13% of queries are judged as misspelled in this dataset, which is close to the error rate of real web queries. This dataset is publicly available to all researchers.

We divide the TREC and MSN datasets into training and test sets evenly. Our gHMM model as well as the baselines are trained on the training sets and finally evaluated on the TREC test set containing 2947 queries and MSN test set containing 2421 queries.

6.2 Evaluation Metrics

We evaluate our system based on the evaluation metrics proposed in Microsoft Speller Challenge [1], including expected precision, expected recall and expected F1 measure.

As used in previous discussions, q is a user query and $Y(q) = (y^1, y^2, , y^k)$ is the set of system output with posterior probabilities $P(y^i|q)$. Let $S(q)$ denote the set of plausible spelling variations annotated by the human experts for q. Expected Precision is computed as:

$$precision = \frac{1}{|Q|} \sum_{q \in Q} \sum_{y \in Y(q)} I_p(y, q) P(y|q) \tag{10}$$

where $I_p(y, q) = 1$ if $y \in S(q)$, and 0 otherwise. And expected recall is defined as:

$$recall = \frac{1}{|Q|} \sum_{q \in Q} \sum_{a \in S(q)} I_r(Y(q), a)/|S(q)| \tag{11}$$

where $I_r(Y(q), a) = 1$ if $a \in Y(q)$ for $a \in S(q)$, and 0 otherwise.
Expected F1 measure can be computed as:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{12}$$

6.3 Overall Effectiveness

We first investigate the overall effectiveness of the gHMM model. For suitable query spelling correction baselines, especially approaches that can handle all types of query spelling errors, we first considered using the CRF model proposed in [24]. This method aims at a broad range of query refinements and hence might be also applicable to query correction. However, we decided not to compare this model for the following reasons. Firstly, we communicated with the authors of [24] and knew that the program is un-reusable. Secondly, as mentioned in Section 1 this work suffers from several drawbacks for query spelling correction: (1) it is unable to correct queries with mixed types of errors, such as substitution and splitting errors in one query, because the model treats query correction and splitting/merging as different tasks; (2) this model only allows 1 character error for substitution/insertion/deletion. And the error model is trained on the <query, correction> examples that only contain 1 character error. Such design is over simplified for real-world queries, in which more than 1 character errors are quite common. In fact, within the queries that contain spelling errors in the MSN dataset, there are about 40.6% of them contain more than 1 character errors. Therefore it is expected model in [24] will have in inferior performance

Because of the reasons stated above, the best baseline method that we can possibly compare with is the system

[1]http://web-ngram.research.microsoft.com/spellerchallenge/

that achieved the best performance in Microsoft Speller Challenge [25] (we call it Lueck-2011). This system relies on candidate corrections from third-party toolkits such as hunspell and Microsoft Wrod Breaker Service [11] , and it re-ranks the candidates by a simple noisy channel model. We communicated with the author and obtained the corrections by running the Web API of this baseline approach. We also include a simple baseline called *Echo*, which is just echoing the original query as the correction response with posterior probability 1. It reflects the basic performance for a naive method. Experiments are conducted on TREC and MSN datasets.

We report the results of all methods in Table 3. In this experiment up to top 10 corrections are used in all approaches. The results in Table 3 indicate that gHMM outperforms Lueck-2011 significantly on recall and F1 on the TREC dataset. Lueck-2011 has a small advantage on precision, possibly due to the better handling the unchanged queries. On the MSN dataset which is considered harder since it has more misspelled queries, gHMM also achieves high precision of 0.910 and recall of 0.966, which are both significantly better than that of the Lueck-2011 (0.896 and 0.921). On another important performance metric, which measures the F1 on misspelled queries (F1 Mis), gHMM outperforms Lueck-2011 by a large margin (0.550 vs. 0.391 on TREC and 0.566 vs. 0.363 on MSN). These results demonstrate that gHMM is very effective for handling all types of spelling errors in search queries overall.

Table 3: gHMM Compared to Baselines

Dataset	Method	Precision	Recall	F1	F1 Mis
TREC	Echo	0.949	0.876	0.911	N/A
	Lueck-2011	**0.963**	0.932	0.947	0.391
	gHMM	0.960	**0.976**	**0.968**	**0.550**
MSN	Echo	0.869	0.869	0.869	N/A
	Lueck-2011	0.896	0.921	0.908	0.363
	gHMM	**0.910**	**0.966**	**0.937**	**0.566**

6.4 Results by Error Types

Further, we also break down the results by error types that are manually classified so that we can see more clearly the distribution of types of spelling errors and how well our gHMM model addressing each type of errors. We present the results of this analysis in Table 4, only with our model on both datasets. Top 40 corrections are used since it achieves the best results. The breakdown results show that most queries are in the group of "no error", which are easier to handle than the other three types. As a result, the overall excellent performance was mostly because the system performed very well on the "no error" group. Indeed, the system has substantially lower precision on the queries with the other three types of errors. The concatenation errors seem to be the hardest to correct, followed by the splitting errors, and the in-word transformation errors (insertion, deletion and substitution, word mis-use) seem to be relatively easier.

6.5 gHMM for Working Set Construction

Since the gHMM can efficiently search in the complete candidate space and compute the top-K spelling corrections in a one-stage manner, it is very interesting to test its effectiveness for constructing a working set of candidate corrections to enable more complex scoring functions to be used for spelling correction. For this purpose, we compare gHMM with the common used noisy channel model, whose parameters, namely error model probabilities and bigram language probabilities are estimated by the procedure mentioned in previous sections. We use recall to measure the completeness of the constructed working set, because it represents the percentage of true corrections given the number of predicted corrections. Table 5 shows the recall according to different number of outputs. It indicates that the recall of gHMM is steadily increasing by a larger number of outputs. By only outputting top-5 corrections, gHMM reaches recall of 0.969 in TREC and 0.964 in MSN. In contrast, the noisy channel model has a substantial gap in term of recall compared to gHMM. This result strongly demonstrates the superior effectiveness of gHMM in constructing a more complete working set of candidate corrections, which can be utilized by other re-ranking approaches which could further improve the correction accuracy.

Table 4: Results by Spelling Error Type

Dataset	Error Type	% Queries	Precision	Recall	F1
TREC	no error	94.9	0.990	0.982	0.986
	transformation	3.3	0.388	0.840	0.531
	concatenation	1.3	0.348	0.877	0.498
	splitting	0.5	0.500	0.792	0.613
MSN	no error	86.9	0.978	1.0	0.989
	transformation	11.1	0.493	0.762	0.599
	concatenation	1.7	0.150	0.600	0.240
	splitting	0.6	0.429	0.571	0.490

Note: % of queries might sum up to more than 100% since there might be multiple types of errors in one query.

Table 5: gHMM vs. Noisy Channel Model on Recall

Dataset	Method	1	5	10	20	40
TREC	N-C	0.869	0.896	0.899	0.901	0.902
	gHMM	**0.887**	**0.969**	**0.976**	**0.981**	**0.983**
MSN	N-C	0.866	0.870	0.873	0.876	0.886
	gHMM	**0.920**	**0.964**	**0.966**	**0.9667**	**0.967**

Note: N-C refers to the noisy channel model

6.6 Efficiency

The runtime requirement of query correction is very stringent. Theoretically, the gHMM with local feature functions can search top-K corrections efficiently by our proposed tok-K Viterbi algorithm. Here we make a directly comparison between the runtime of gHMM and a basic noisy channel that only needs to compute the error model probabilities and bigram language probabilities. Such a basic model is also implemented with Viterbi algorithm. It is run on a Windows server equipped with 2 Quad-core 64 bit 2.4 GHz CPUs and 8 GB RAM. All necessary bigram language probabilities are crawled from Microsoft Web N-gram Service and cached in local memory. We plot the runtime per query (in milliseconds) according to the number of predicted corrections in Figure 2. According to Figure 2, the computation of top-1 correction by gHMM is fast (34 ms) if the number of output is set to 1. It increases as the number of output increases because the search space is increased. Interestingly, the runtime of gHMM and the noisy channel model is of the same order of magnitude. This empirical evidence confirms the theoretical result that top-K spelling corrections can be computed efficiently via our proposed top-K Viterbi algorithm.

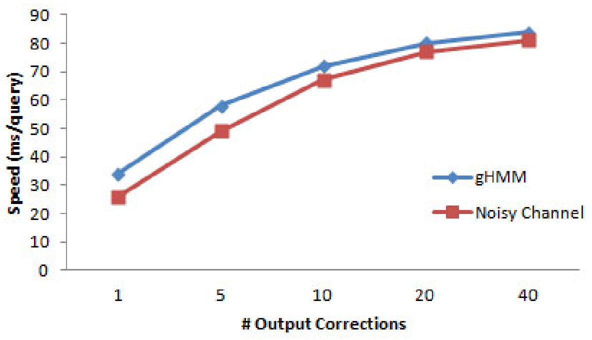

Figure 2: Runtime Comparison

7. DISCUSSION

Finally we discuss how the proposed gHMM model behaves according to the variation of important factors. Such as the number of spelling corrections in the output and the size of lexicon L *etc.*

7.1 Number of Predicted Corrections

Our datasets include all plausible corrections for a query. However it's unknown that how many corrections an algorithm should output to achieve a perfect recall. In this section we investigate how the number of predicted spelling corrections affects the precision and recall. We have carried out experiments on five correction sizes (1,5,10,20,40) on both datasets. Figure 3 summarizes the results. As expected, a bigger number of corrections leads to higher recall, and the most drastical increase of recall lies from 1 correction to 5.

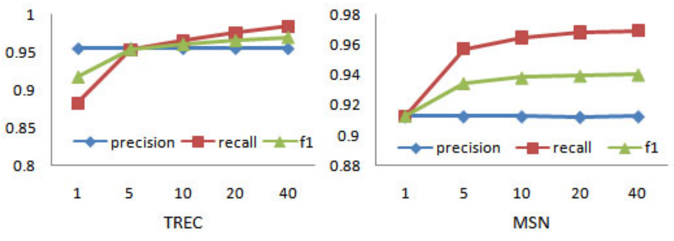

Figure 3: Results by Number of Corrections

7.2 Effect of the Lexicon Size

The size of the trusted lexicon is an important factor influencing the speller correction result. In order to investigate the effect of lexicon size, we conducted a set of experiments on both datasets using different lexicon sizes (ranging from 100,000 to 900,000). Results in Figure 4 show that the effect of lexicon size is significant on precision: the precision increases as the lexicon size increases. However the recall is not sensitive to the lexicon size.

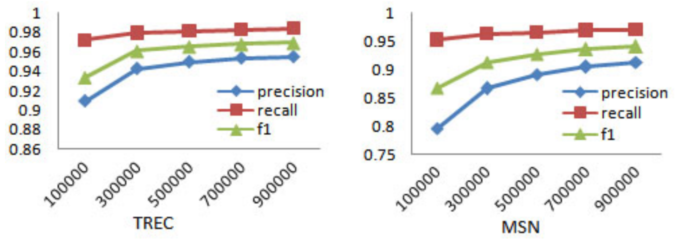

Figure 4: Correction Results by Lexicion Size

7.3 Clean vs. Noisy Lexicon

In Table 6, we show the effects of using a clean lexicon for improving the precision and recall of the spelling system. We can see that for both test sets, there is noticeable improvement in precision. By removing the noise in the automatically constructed lexicon, our system is able to find matches for candidate queries more precisely.

Table 6: Clean vs. Noisy Lexicon

Dataset	Lexicon Type	Precision	Recall	F1
TREC	clean lexicon	0.960	0.983	0.971
	noisy lexicon	0.950	0.984	0.966
MSN	clean lexicon	0.910	0.967	0.938
	noisy lexicon	0.896	0.967	0.930

Note: maximum corrections are set to 40 in this experiment

8. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a novel generalized hidden Markov model (gHMM) for query spelling correction that can address two major deficiencies of previous approaches to query spelling correction, i.e., inability of handling all the major types of spelling errors, and inability of searching efficiently in the complete candidate space. We have also proposed a novel discriminative training method for the gHMM model which enables us to go beyond regular HMM to incorporate useful local features for more effective query spelling correction. Experiment results on two query correction datasets show that gHMM can effectively handle all the major types of spelling errors and outperforms a state-of-the-art baseline by a large margin.

Moreover, our generalized HMM, equipped with the discriminative training, scores the query corrections directly and output a final ranked list of spelling corrections, without needing a filtering stage to prune the candidate space as typically required by an existing method. We have demonstrated that as an efficient one-stage approach, the proposed gHMM can also be used as a filter to construct a more complete working set than the existing noisy channel filter, making it possible to combine it with any complicated spelling correction methods to further improve accuracy. In other words, the proposed gHMM model can serve as a better candidate generation method in a two-stage framework where any sophisticated and potentially more effective spelling correction method can be applied to re-rank the generated candidates for more accurate corrections. In addition, we have also curated a large spelling correction dataset from real-world queries and made it available to the research community.

In this work, we only focused on local feature functions in order to ensure efficient evaluation of all the candidates in the search space. However, some global features, such as the overall editing distance, frequency of the query in a query log can be potentially utilized to further improve the correction accuracy. How to add the global features to the gHMM model, while still ensuring efficient search and evaluation of all the candidates in the search space, is an interesting direction for future work.

9. ACKNOWLEDGMENTS

This paper is based upon work supported in part by a Microsoft grant, the National Science Foundation under grant CNS-1027965, and MIAS - the Multimodal Information Access and Synthesis center at UIUC, part of CCICADA, and a DHS Center of Excellence.

10. REFERENCES

[1] http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx.

[2] F. Ahmad and G. Kondrak. Learning a spelling error model from search query logs. In *HLT/EMNLP*. The Association for Computational Linguistics, 2005.

[3] E. Brill and R. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 2000.

[4] Q. Chen, M. Li, and M. Zhou. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, pages 181–189. ACL, 2007.

[5] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*, 2004.

[6] H. Duan and B.-J. P. Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 117–126. 2011, ACM.

[7] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In C.-R. Huang and D. Jurafsky, editors, *COLING*, pages 358–366. 2010.

[8] K. Kukich. Techniques for automatically correcting words in text. *ACM computing surveys*, 24(4), 1992.

[9] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

[10] X. Sun, J. Gao, D. Micol, and C. Quirk. Learning phrase-based spelling error models from clickthrough data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 266–274, Stroudsburg, PA, USA, 2010.

[11] K. Wang, C. Thrasher, and B.-J. P. Hsu. Web scale nlp: a case study on url word breaking. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 357–366, New York, NY, USA, 2011. ACM.

[12] C. Whitelaw, B. Hutchinson, G. Chung, and G. Ellis. Using the web for language independent spellchecking and autocorrection. In *EMNLP*, 890–899. ACL, 2009.

[13] Levenshtein, V I. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady, 10(8), 707-710*, 1966.

[14] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96. ACM, New York, NY.

[15] Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93. ACM, New York, NY, USA, 160-169.

[16] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving automatic query expansion. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98.

[17] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 347–356. 2008.

[18] Y. Li, B.-J. P. Hsu, C. Zhai, and K. Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 285–294. 2011.

[19] Xuanhui Wang, ChengXiang Zhai. Mining Term Association Patterns from Search Logs for Effective Query Reformulation. In *CIKM'08*. 479-488.

[20] Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics*, Volume 2 (COLING '96). Stroudsburg, PA, USA, 836-841.

[21] B.H. Juang. Hidden Markov models for speech recognition. In Technometrics, Vol. 33, No. 3, 1991.

[22] M. Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02*, pages 1-8, Stroudsburg, PA, USA, 2002.

[23] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[24] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *Proceedings of the 31st annual international ACM SIGIR*, SIGIR '08, pages 379–386. 2008.

[25] G. Luec. A data-driven approach for correcting search quaries. In *Spelling Alteration for Web Search Workshop*, July 2011.

[26] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, San Fransisco, 2001. Morgan Kaufmann.