Online Spelling Correction for Query Completion

Huizhong Duan[*]

University of Illinois at Urbana-Champaign
201 N Goodwin Ave
Urbana, IL 61801 USA

duan9@illinois.edu

Bo-June (Paul) Hsu

Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA

paulhsu@microsoft.com

ABSTRACT

In this paper, we study the problem of online spelling correction for query completions. Misspelling is a common phenomenon among search engines queries. In order to help users effectively express their information needs, mechanisms for automatically correcting misspelled queries are required. Online spelling correction aims to provide spell corrected completion suggestions as a query is incrementally entered. As latency is crucial to the utility of the suggestions, such an algorithm needs to be not only accurate, but also efficient.

To tackle this problem, we propose and study a generative model for input queries, based on a noisy channel transformation of the intended queries. Utilizing spelling correction pairs, we train a Markov n-gram transformation model that captures user spelling behavior in an unsupervised fashion. To find the top spell-corrected completion suggestions in real-time, we adapt the A* search algorithm with various pruning heuristics to dynamically expand the search space efficiently. Evaluation of the proposed methods demonstrates a substantial increase in the effectiveness of online spelling correction over existing techniques.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval – *query formulation*; H.4.m [**Information Systems and Applications**]: Miscellaneous

General Terms

Algorithms, Performance, Experimentation

Keywords

Spelling correction, query completion, transformation model, A* search

1. INTRODUCTION

Misspelling is a common phenomenon in search engine queries. According to Cucerzan and Brill [9], more than 10% of search engines queries are misspelled. This is even more severe for tail queries, of which more than 20% are misspelled [5]. Misspellings occur for a variety of reasons. When typing quickly, users may add or drop letters unintentionally. Accidentally hitting an adjacent key on the keyboard, also known as the fat-finger syndrome [1], is also common, especially on mobile devices with small virtual keyboards. In addition to typographical errors, some

errors result from the challenge of spelling itself. With inconsistent spelling rules [2], ambiguous word breaking boundaries, and constant introduction of new words, spelling presents a formidable challenge to foreign and native speakers alike. Table 1 summarizes different types of misspellings and provides examples of each.

Table 1. Types of misspellings

Cause	Misspelling	Correction
Typing quickly	exxit mispell	exit misspell
Keyboard adjacency	importamt	important
Inconsistent rules	concieve conceirge	conceive concierge
Ambiguous word breaking	silver light	silverlight
New words	kinnect	kinect

To assist users in expressing their information needs, it is important for search engines to automatically generate corrections for misspelled queries. Two such mechanisms are in common use. The first corrects a query after it is submitted to the search engine. For confident corrections, the search engine can search the corrected query directly. As the entire query string is given, we refer to such an approach as *offline spelling correction*. The second technique provides corrections to the query completion suggestions as the query is being entered. Specifically, the search engine responds to each keystroke with a list of query suggestions that best correct and complete the partial query. Compared with offline spelling correction, this task not only needs to address incomplete queries, but also requires lower latency to be effective. We refer to this task as *online spelling correction*.

Online correction has many merits that cannot be achieved by offline correction. First, it keeps users informed of potential errors as they type. Thus, spelling errors and the resulting ambiguities can be eliminated even before issuing the query. Second, it helps users express their information needs. As the quality and quantity of suggestions from current search engines degrade dramatically with misspelled partial queries, the ability to suggest popular completions from corrected partial queries can improve the effectiveness of the suggestions. Third, it saves users effort in inputting queries. With more comprehensive suggestions that correct potential errors, users are more likely to find the target query in the suggestion list. Selecting a corrected suggestion reduces not only the number of keystrokes required to input a query, but often also the additional click on the search result page to confirm the correction.

It is worth noting that although many search engines today apply online corrections to query completion suggestions, their abilities

---

are fairly limited. For instance, neither of the two major US search engines, Google and Bing, suggests any completion for "importamt" and "miunderstand" at the time of this submission, which are only one letter away from "important" and "misunderstand", respectively.

In this paper, we model search queries with a generative model, where the intended query is transformed through a noisy channel into a potentially misspelled query. The distribution from which the target query is selected is estimated from the search engine query log based on frequency. Thus, we are more likely to suggest more popular queries. For the noisy channel, which describes the distribution of spelling errors, we follow the joint-sequence modeling framework [4] to define the probability of transforming the original query into the observed character sequence. Specifically, we treat the desired and realized queries as a sequence of substring transformation units, or *transfemes* for convenience. Thus, we can decompose the probability of the overall transformation sequence as a product of the transfeme probabilities, each conditioned on the previous transfemes. By applying the Markov assumption and experimenting with the length of the transfeme units, we can build transformation models of varying complexities. Figure 1 shows an example segmentation of the input and output queries into a sequence of transfemes.

s c	h	w a r		z e	n e g	g	e r
s	h	w a r	t z e	n e g			e r
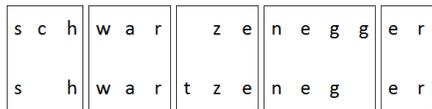
Figure 1: Example segmentation into transfemes

To estimate the conditional probabilities of the transfemes, we train a smoothed transfeme n-gram language model from a set of correction query pairs. As the query pairs are not co-segmented in the training data, we apply the expectation-maximization (EM) algorithm to segment the data with the objective of maximizing the model probability. Since different users misspell at different rates, we further propose a mixture model to address the problem of overly-aggressive corrections.

As suggestion latency is crucial to the usefulness of query suggestions from potentially misspelled partial queries, we further explore efficient data structures and algorithms to dynamically search for the top query completions under the proposed transformation model. In particular, we apply the A* search algorithm against a trie built from the query log, where each node is further annotated with the best score of all descendent queries. We also experiment with different pruning heuristics to reduce search latency. As all query pairs have a non-zero transformation probability under this model, we further investigate probability thresholding techniques to reduce irrelevant suggestions.

The rest of the paper is organized as follows. After surveying the related work in Section 2, we introduce the generative model for online correction in Section 3. Next, we present the proposed transformation model and search algorithm in Sections 4 and 5. We follow up with experiment results in Section 6 before concluding in Section 7.

2. RELATED WORK

Research on spelling correction has a long history [10, 13, 15]. Edit distance, initially proposed by Damerau [10] and Levenshtein [13], has been widely used in generic spelling correction. More recent work on offline spelling correction tends to focus on search engine queries [7, 9, 12, 14, 16]. Cucerzan and Brill [9] studied spelling correction as an iterative process to exploit the information in query logs. Li et al. [14] explored distributional similarity of query terms to estimate the error model. Chen et al. [7] leveraged web search results to improve the performance of spelling correction on rare queries. Sun et al. [16] explored click-through data to identify user correction pairs, and applied them to build a phrase-based error model. Gao et al. [12] proposed the use of a general ranker as a generalization of the traditional noisy channel model in spelling correction, and implemented it with a distributed infrastructure to incorporate large scale data. As offline spelling correction is just a special case of online spelling correction, we consider the performance of both conditions when evaluating our system.

One of the earliest forms of auto-completion is the tab completion feature found in many command prompts. It is later extended by Darragh et al. [11] to support the prediction of general text. Recently, Chaudhuri and Kaushik [6] proposed a technique to further extended auto-completion to tolerate errors. Particularly, they made use of a simple edit distance model and performed a fuzzy search over database records to find completions. To the best of our knowledge, this is the only prior work that addresses the problem of online spelling correction. Unfortunately, with a predetermined cap on edit distance and linear lookup time with increasing data size, the algorithm is not sufficiently robust and scalable for online spelling correction for query completion.

Our approach to spelling correction is largely inspired by previous work in grapheme to phoneme transformation. Chen [8] studied conditional and joint maximum entropy models for grapheme to phoneme conversion. Taylor [17] used a hidden Markov model, where the graphemes are observations of the hidden phoneme states. Bisani and Ney [4] proposed a joint-sequence model for modeling grapheme to phoneme transformation, where graphemes and phonemes are viewed as a joint sequence generated with a Markov model. In this work, we adapt the joint sequence modeling of Bisani and Ney to model the transformation from the intended query to the observed sequence. However, whereas grapheme to phoneme conversions are strongly constrained by pronunciation rules, typographical errors do not impose any constraint on possible transformations, increasing the difficulty in model training.

3. ONLINE SPELLING CORRECTION

In offline spelling correction, we want to find the correctly spelled query $\hat{c}$ with the highest probability of yielding the potentially misspelled input query q. By applying Bayes' rule, we can alternatively express the task as:

$$\hat{c} = \underset{c}{\operatorname{argmax}}\, p(c|q) = \underset{c}{\operatorname{argmax}}\, p(q|c)p(c) \qquad (1)$$

In this noisy channel model formulation, $p(c)$ is a query language model that describes the prior probability of c as the intended user query. $p(q|c) = p(c \rightarrow q)$ is the transformation model that represents the probability of observing the query q when the original user intent is to enter the query c.

For online spelling correction, we are given only the prefix $\bar{q}$ of the potentially misspelled input query q. Thus, the objective is to find the correctly spelled query $\hat{c}$ that maximizes the probability of yielding any query q that extends the given partial query $\bar{q}$. More formally, we want to find:

$$\hat{c} = \underset{c,\, q:q=\bar{q}\cdots}{\operatorname{argmax}}\, p(c|q) = \underset{c,\, q:q=\bar{q}\cdots}{\operatorname{argmax}}\, p(q|c)p(c) \qquad (2)$$

where $q = \bar{q} \cdots$ denotes that $\bar{q}$ is a prefix of q. In this formulation, we can view offline spelling correction as just a constrained special case of the more generic online spelling correction.

In this work, as search engines typically only suggest previously seen queries as completions, we model the query prior $p(c)$ using the maximum likelihood estimation of the distribution of queries from the query log. Consequently, this model can only correct misspelled query prefixes to previously observed queries. In future work, we plan to extend this approach to the use of an n-gram language model for the query prior $p(c)$ to allow corrections to previously unseen queries.

To simplify the transformation model, we segment the conversion from c to q as a sequence of substring transformation units, or *transfemes*. For example, the transformation $schwarzenegger \rightarrow shwartzeneger$ can be segmented into the transfeme sequence: $sch \rightarrow sh$, $war \rightarrow war$, $ze \rightarrow tze$, $negg \rightarrow neg$, $er \rightarrow er$. By describing the sequence with a transfeme n-gram language model, we can decompose the transformation model into a set of conditional transfeme probabilities. This allows us to not only train the model from segmented correction pairs, but also generalize the model to previously unseen transformations.

Factoring the solution into the query language model and transfeme transformation model enables the two models to be updated independently. As the query language model needs to reflect the constantly changing trends across topics, the query histogram and corresponding trie data structure can be updated frequently with low cost. On the other hand, the transformation model describes the user spelling behavior when entering queries. As this behavior does not change rapidly, we do not need to retrain the model as often. Instead, as the pattern of spelling errors depends heavily on the keyboard layout (English vs. French), keyboard size (standard vs. thumb-sized), and interface (physical buttons vs. virtual keyboard), we can build a separate transformation model for each text entry environment.

As we can see from Equation (2), by relaxing q to be any query that extends the partial query $\bar{q}$, online spelling correction significantly increases the theoretical search space. However, with appropriate data structures and algorithms, we can actually perform this search *faster* than offline spelling correction, as we demonstrate in Section 5.

4. TRANSFORMATION MODEL

The transformation model presented in this work, including the EM training, pruning, and smoothing algorithms, largely mirrors the joint sequence model for grapheme to phoneme conversion in speech recognition, as described in Bisani and Ney [4]. In the following sections, we define the transformation model as applied to spelling correction, summarize the EM training algorithm, and present additional considerations specific to spelling correction.

4.1 Model Definition

We decompose a transformation from the intended query c to the observed query q as a sequence of substring transformation units. As this model is inspired by joint sequence modeling in grapheme to phoneme conversion [4], we name such substring transformation units *transfemes*. For example, the transformation britney $\rightarrow$ britny can be segmented into the transfeme sequence $\{br \rightarrow br, i \rightarrow i, t \rightarrow t, ney \rightarrow ny\}$, where only the last transfeme, $ney \rightarrow ny$, involves a correction.

Given a sequence of transfemes $s = t_1 t_2, \ldots, t_{l^s}$, we can expand the probability of the sequence using the chain rule. As there are

multiple ways to segment a transformation in general, we further model the transformation probability $p(c \rightarrow q)$ as the sum of all possible segmentations. Formally,

$$p(c \rightarrow q) = \sum_{s \in S(c \rightarrow q)} p(s)$$
$$= \sum_{s \in S(c \rightarrow q)} \prod_{i \in [1, l^s]} p(t_i | t_1, \ldots, t_{i-1}) \quad (3)$$

where $S(c \rightarrow q)$ is the set of all possible joint segmentations of c and q. Further applying the Markov assumption that a transfeme only depends on the previous $M - 1$ transfemes, similar to an n-gram language model, we obtain:

$$p(c \rightarrow q) = \sum_{s \in S(c \rightarrow q)} \prod_{i \in [1, l^s]} p(t_i | t_{i-M+1}, \ldots, t_{i-1}) \quad (4)$$

We define the length of a transfeme $t = c_t \rightarrow q_t$, as:

$$|t| = \max\{|c_t|, |q_t|\} \quad (5)$$

In general, a transfeme can be arbitrarily long. To constrain the complexity of the transformation model, we limit the maximum length of a transfeme to L. With both n-gram approximation and transfeme length constraint, we obtain the final model with parameters M and L:

$$p(c \rightarrow q) = \sum_{\substack{s \in S(c \rightarrow q): \ i \in [1, l^s] \\ \forall t \in s, |t| \leq L}} \prod p(t_i | t_{i-M+1}, \ldots, t_{i-1}) \quad (6)$$

In the special case of $M = 1$ and $L = 1$, the transformation model degenerates to a model similar to weighted edit distance. With $M = 1$, we assume that the transfemes are generated independently of one another. As each transfeme contains substrings of at most one letter, we can model the standard Levenshtein edit operations [13]: insertions $\varepsilon \rightarrow \alpha$, deletions $\alpha \rightarrow \varepsilon$, and substitutions $\alpha \rightarrow \beta$, where ε denotes the empty string. However, unlike many edit distance models, the weights in the transformation model represent normalized probabilities estimated from data, not just arbitrary score penalties. Thus, the transformation model not only captures the underlying patterns of spelling errors, but also allows us to compare the probabilities of different completion suggestions in a mathematically principled way. Figure 2 contains an example of such a transformation.
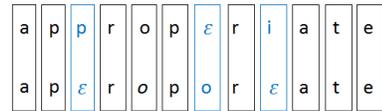
Figure 2: Example transformation with $L = 1$

With $L = 1$, transpositions are penalized twice, even though it occurs as easily as other edit operations. Similarly, phonetic spelling errors, such as ph $\rightarrow$ f, often involve multiple characters. Modeling these transfemes as single character edit operations not only over-penalizes the transformation, but also pollutes the model as it increases the probabilities of edit operations, such as p $\rightarrow$ f, that would otherwise have very low probabilities. By increasing L, we increase the allowable length of the transfemes. Thus, the model is able to capture more meaningful transformation units and reduce probability contamination that result from decomposing intuitively atomic substring transformations. Figure 3 compares an example transformation with $L = 1$ and $L = 2$.

Figure 3: Comparing transformations with $L = 1$ and $L = 2$

Instead of increasing L, we can also improve the modeling of errors spanning multiple characters by increasing M, the number of transfemes the model probabilities are conditioned on. Consider the example from Figure 3 with $L = 1$. When $M = 1$, no context is considered in the generation of each transfeme. When $M = 2$, the probability of each transfeme is dependent on its previous transfeme. As a result, we are able to capture the fact that $h \to \varepsilon$ has a much higher probability when following the transfeme $p \to f$. As a more interesting example, ie is often misspelled as ei. A unigram model ($M = 1$) is not able to express such an error. A bigram model ($M = 2$) captures this pattern by assigning higher probability to the transfeme $e \to i$ when following $i \to e$. A trigram model ($M = 3$) can further identify exceptions to this pattern when preceded by a c, as cei is more common than cie. As M and L capture similar behavior in our transformation model, we study the effect of different combinations of M and L in Section 6.

4.2 Model Estimation

To learn the patterns of user spelling errors, we use a parallel corpus of input and output query pairs, where the input represents the intended query with correct spelling and the output corresponds to the potentially misspelled transformation of the input. If such data is pre-segmented into transfemes, we can derive the transformation model directly using maximum likelihood estimation (MLE). However, such labeled training data is generally too costly to obtain in large scale. Thus, we devise an expectation-maximization (EM) algorithm to estimate the parameters in the transformation model from partially observed data.

Given a set of observed training pairs $O = \{O^k\}$, where $O^k = c^k \to q^k$, we can write the log likelihood of the training data as:

$$\log \mathcal{L}(\Theta; O) = \sum_k \log p(c^k \to q^k | \Theta)$$
$$= \sum_k \log \sum_{s^k \in S(O^k)} p(s^k | \Theta) \tag{7}$$

where $\Theta = \{p(t|t_{-M+1}, \dots, t_{-1})\}$ is the set of model parameters. $s^k = t_1^k t_2^k, \dots, t_{l^s}^k$, the joint segmentation of each training pair $c^k \to q^k$ into a sequence of transfemes, is the unobserved variable. By applying the EM algorithm [3], we can iteratively find the parameter set Θ that maximizes the log likelihood.

For $M = 1$ and $L = 1$, where each transfeme of length up to 1 is generated independently, we derive the following update formulas:

$$p(s; \Theta) = \prod_{i \in [1, l^s]} p(t_i; \Theta) \tag{8}$$

$$e(t; \Theta) = \sum_k \sum_{s^k \in S(O^k)} \frac{p(s^k; \Theta)}{\sum_{s' \in S(O^k)} p(s'; \Theta)} \#(t, s^k) \tag{9}$$

$$p(t; \Theta') = \frac{e(t; \Theta)}{\sum_{t'} e(t'; \Theta)} \tag{10}$$

where $\#(t, s)$ is the count of transfeme t in the segmentation sequence s, $e(t; \Theta)$ is the expected partial count of the transfeme t with respect to the transformation model Θ, and Θ' is the updated model. $e(t; \Theta)$, also known as the evidence for t, can be computed efficiently using a forward-backward algorithm [4].

We can extend the EM training algorithm to higher order transformation models ($M > 1$), where the probability of each transfeme now depends on the previous $M - 1$ transfemes. Other than having to take into account the transfeme history context when accumulating the partial counts, the general EM procedure is essentially the same. Specifically, we have:

$$p(s; \Theta) = \prod_{i \in [1, l^s]} p(t_i | t_{i-M+1}^{i-1}; \Theta) \tag{11}$$

$$e(t, h; \Theta) = \sum_k \sum_{s^k \in S(O^k)} \frac{p(s^k; \Theta)}{\sum_{s' \in S(O^k)} p(s'; \Theta)} \#(t, h, s^k) \tag{12}$$

$$p(t|h; \Theta') = \frac{e(t, h; \Theta)}{\sum_{t'} e(t', h; \Theta)} \tag{13}$$

where h is a transfeme sequence representing the history context, and $\#(t, h, s)$ is the occurrence count of transfeme t following the context h in the segmentation sequence s. Though more complicated, $e(t, h; \Theta)$, the evidence for t in the context of h, can still be computed efficiently using the forward-backward algorithm.

As the number of model parameters increases with M, we initialize the model parameters using the converged values from the lower order model to achieve faster convergence. Specifically,

$$p(t|h^M; \Theta^M) \equiv p(t|h^{M-1}; \Theta^{M-1}) \tag{14}$$

where h^M is a sequence of $M - 1$ transfemes representing the context, and h^{M-1} is h^M without the oldest context transfeme.

Extending the training procedure to $L > 1$ further complicates the forward-backward computation. But the general form of the EM algorithm remains the same.

4.3 Model Pruning

One challenge with a direct implementation of the above algorithms is that as we increase the model parameters M and L, the number of potential parameters in the transformation model increases exponentially. Assuming an alphabet size of 50, a $M = 1, L = 1$ model contains $(50 + 1)^2$ parameters, as each component in $t = c_t \to q_t$ can take on any of the 50 symbols or ε. But a $M = 3, L = 2$ model may contain up to $(50^2 + 50 + 1)^{2 \cdot 3} \approx 2.8 \times 10^{20}$ parameters! Although most parameters are never observed in the data, model pruning techniques are still beneficial to reduce the overall search space, during both training and decoding, and to reduce overfitting, as infrequent transfeme n-grams are likely to be noise.

In this work, we employ two pruning strategies in each iteration of the training algorithm. First, we remove transfeme n-grams with expected partial counts below a threshold τ^e. Second, we trim out transfeme n-grams with estimated conditional probabilities below a threshold τ^p. The thresholds τ^e and τ^p are tuned against a held-

out development set. By filtering out transfemes with low confidence, we significantly reduce the number of active parameters in the model and speed up the running time of training and decoding.

4.4 Model Smoothing

As with any maximum likelihood estimation techniques, the EM algorithm has a tendency to overfit the training data when the number of model parameters is large, for example when $M > 1$. The standard technique in n-gram language modeling to address this problem is to apply smoothing when computing the conditional probabilities. In our work, we study two smoothing techniques: Jelinek-Mercer (JM) and absolute discounting (AD).

In JM smoothing, the probability of a transfeme is given by the linear interpolation of its maximum likelihood estimation at order M (using partial counts) and its smoothed probability from a lower order distribution:

$$p^{\text{JM}}(t|h^M) = (1 - \alpha)\frac{e(t, h^M)}{\sum_{t'} e(t', h^M)} + \alpha\, p^{\text{JM}}(t|h^{M-1}) \quad (15)$$

where $\alpha \in (0,1)$ is the linear interpolation parameter. Note that $p^{\text{JM}}(t|h^M)$ and $p^{\text{JM}}(t|h^{M-1})$ are probabilities from different distributions within the same model. That is, in computing the M-gram model, we also compute the partial counts and probabilities for all lower-order m-grams, where $m \leq M$.

AD smoothing operates by discounting the partial counts of the transfemes. The removed probability mass is then redistributed to the lower order model:

$$p^{\text{AD}}(t|h^M) = \frac{\max(e(t, h^M) - d, 0)}{\sum_{t'} e(t', h^M)} + \alpha(h^M)p^{\text{AD}}(t|h^{M-1}) \quad (16)$$

where d is the discount and $\alpha(h^M)$ is computed such that $\sum_t p^{\text{AD}}(t|h^M) = 1$. Note that since the partial count $e(t, h^M)$ can be arbitrarily small, it is not possible to choose a value of d such that $e(t, h^M)$ will always be larger than d. Consequently, we will trim the model if $e(t, h^M) \leq d$. For both smoothing techniques, all parameters are tuned on a held-out development set.

4.5 Mixture Models

When training from a dataset consisting of only query correction pairs, the resulting model is likely to over-correct. To address this issue, we prepare another dataset of correctly spelled query pairs and propose two ways of using the two datasets for training.

The first approach simply concatenates the two datasets together when estimating the transformation model. We refer to this method as *data mixture*. The second technique trains two transformation models from the two datasets individually. It is easy to see that the model trained from correctly spelled queries will only assign non-zero probabilities to transfemes with identical input and output, as all the transformation pairs are identical. We linearly interpolate the two models as the final model:

$$p(t) = (1 - \lambda)\, p\big(t; \Theta^{\text{misspelled}}\big) + \lambda\, p\big(t; \Theta^{\text{identical}}\big) \quad (17)$$

We label this approach as *model mixture*, where we can view each transfeme as probabilistically generated from one of the two distributions, according to the interpolation factor λ. As with all other modeling parameters, λ is tuned on a held-out development set.

4.6 Discussions

Observant readers may have noticed that the transformation model estimates the joint probability of the input and output substrings in

a transfeme. As the transformation probability is later multiplied with the query language model in the generative formulation for online and offline spelling corrections, we are essentially double counting the input query probability. A solution to this problem is to normalize the transformation model for each input substring after training, so as to obtain a conditional model. Although this solution is theoretically sound, initial experiments have failed to improve the performance. As is common in speech recognition, where a "fudge factor" is introduced to balance the language model score against the acoustic model, we reformulate the optimization as:

$$\hat{c} = \underset{c}{\arg\max}\, p(q|c)p(c) \approx \underset{c}{\arg\max}\, p(c \to q)p(c)^\gamma \quad (18)$$

where $p(c \to q)$ is still the transformation model probability, and γ is the fudge factor controlling the additional probability mass of the query language model. Empirically, this approach turns out to be very effective in our experiments, although it lacks a theoretical foundation. We plan to continue exploring this issue in future work.

5. SEARCH

With a query language model and a transformation model, we are able to compute the probability of any query q given an input query c. However, our task is to find the input query $\hat{c}$ with highest probability efficiently, so as to enable offline spelling correction. More generally for online spelling correction, we want to find the top k completions of an observed query prefix $\bar{q}$. To achieve this, we propose to apply the A* search algorithm against a trie representing the query language model. Below we first introduce the modified trie data structure that we use to store the queries and their probabilities. We then present the A* search algorithm, followed by discussions on the pruning and thresholding techniques necessary to improve the efficiency and quality of the suggestions.

5.1 Trie

As the search algorithm starts from the beginning of a query and incrementally traverses potential corrections one letter at a time, we use a prefix tree (trie) to represent all queries in the query log. Figure 4b shows a trie built over the set of strings in Figure 4a. To avoid ambiguity, we end each string with an implicit $ character. Thus in the trie, all leaf nodes are associated with a complete query. Internal nodes do not represent complete strings. For each node in the trie, we store the largest probability among all queries represented by its descendant leaf nodes. As this represents the largest value among all queries starting with the prefix associated with the node, we can apply it an admissible heuristic function for A* search.
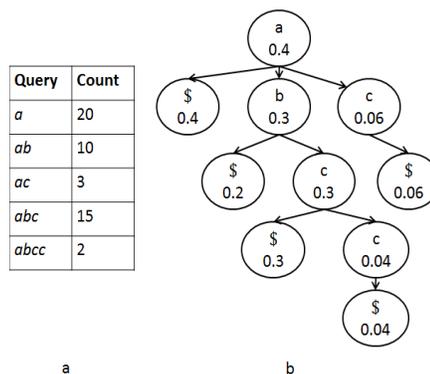
Figure 4: Trie with highest probabilities

5.2 A* Search

We use the A* search algorithm to find the top k corrected query completions for the prefix $\bar{q}$, given the query trie T and transformation model Θ. We represent each intermediate search path as a quadruplet <*Pos*, *Node*, *Hist*, *Prob*>, corresponding to the current position in the query prefix $\bar{q}$, the current node in trie T, the transformation history so far, and the probability of this search path, respectively. The full algorithm is presented Figure 5.

Input:	Query trie T, transformation model Θ, integer k, query prefix $\bar{q}$
Output:	Top k completion suggestions of $\bar{q}$

A	List l = new List()
B	PriorityQueue pq = new PriorityQueue()
C	pq.Enqueue(new Path(0, T.Root, [], 1))
D	while (!pq.Empty())
E	Path π = pq.Dequeue()
F	if (π.Pos < |$\bar{q}$|) // *Transform input query*
G	foreach (Transfeme t in GetTransformations(π, $\bar{q}$, T, Θ))
H	int i = π.Pos + t.Output.Length
I	Node n = π.Node.FindDescendant(t.Input)
J	History h = π.Hist + t
K	Prob p = π.Prob × (n.Prob / π.Node.Prob) × $P(t, \pi$.Hist; $\Theta)$
L	pq.Enqueue(new Path(i, n, h, p))
M	else // *Extend input query*
N	if (π.Node.IsLeaf())
O	l.Add(π.Node.Query)
P	if (l.Count ≥ k)
Q	return l
R	else
S	foreach (Transfeme t in GetExtensions(π, T, Θ))
T	int i = π.Pos + t.Output.Length
U	Node n = π.Node.FindDescendant(t.Input)
V	History h = π.Hist + t
W	Prob p = π.Prob × (n.Prob / π.Node.Prob)
X	pq.Enqueue(new Path(i, n, h, p))
Y	return l

Figure 5: A* search algorithm for online spelling correction

The algorithm works by maintaining a priority queue of intermediate search paths, ranked by decreasing probabilities. We initialize the queue with the initial path <0, T.Root, [], 1> (line C). While there is still a path on the queue, we dequeue it and check if there are still characters unaccounted for in the input prefix $\bar{q}$ (lines F). If so, we iterate over all transfeme expansions that transform substrings starting from the current node in the trie to substrings yet unaccounted for in the query prefix (line G). For each transfeme expansion, we add a corresponding path to the trie (line L). The probability of the path is updated to include adjustments to the heuristic future score and the probability of the transfeme given the previous history (line K).

As we expand the search path, we will eventually reach a point where all the characters in the input query have been consumed. The first path in the search that meets this criterion represents a partial correction to the partial input query $\bar{q}$. At this point, the search transitions from correcting potential errors in the partial input to extending the partial correction to complete queries. In this scenario (line M), if the path is associated with a leaf node in the trie (line N), indicating that we have reached the end of a complete query, we add the corresponding query to the suggestion list (line O) and return if we have sufficient suggestions (line P). Otherwise, we iterate over all transfemes that extend from the current node (line S) and add them to the priority queue (line X). As the transformation score is not affected by extensions to the partial query, we only update the score to reflect the changes in

the heuristic future score (line W). When we run out of search paths to expand, we return the current list of correction completions (line Y).

The heuristic future score we use in the A* algorithm, as applied in line K and W, is the probability value stored with each node in the trie. As this value represents the largest probability among all queries reachable from this path, it is an admissible heuristic that guarantees that the algorithm will indeed find the top suggestions.

One problem with this heuristic function is that it does not penalize the untransformed part of the input query. Therefore, we can design a better heuristic by taking into consideration the upper bound of the transformation probability $p(c \rightarrow q)$. Formally,

$$\text{heuristic}^*(\pi) = \max_{c \in \pi.\text{Node.Queries}} p(c)$$
$$\times \max_{c'} p(c' \rightarrow q_{[\pi.\text{Pos},|q|]} | \pi.\text{Hist}; \Theta) \quad (19)$$

where $q_{[\pi.\text{Pos},|q|]}$ is the substring of q from position π.Pos to $|q|$. For each query, we pre-compute the second maximization in the equation for all positions of q using dynamic programming.

The A* search algorithm can also be configured to perform exact match for offline spelling correction by simply substituting the probabilities in line W with line K. In effect, we continue to penalize transformations involving additional unmatched letters even after finding a prefix match.

It is worth noting that a search path can theoretically grow to infinite length, as ε is allowed to appear as either the source or target of a transfeme. In practice, this does not happen as the probability of such transformation sequences will be very low and will not be further expanded in the search algorithm.

A translation model with larger L parameter (L bounds the length of transfemes) significantly increases the number of potential search paths. As we need to consider all possible transfemes with length less or equal to L when expanding each path, models with larger L are less efficient.

5.3 Pruning

To further improve the efficiency of A* search, we need to limit the search space and prune unpromising paths early. In practice, carefully designed beam pruning methods can usually achieve significant improvement in efficiency without causing much loss in accuracy. In our work, we employ two pruning techniques: absolute pruning and relative pruning.

For absolute pruning, we limit the number of paths to be explored at each position in the target query q. As mentioned earlier, the complexity of our search algorithm is theoretically unbounded due to ε transfemes. However, by applying absolute pruning, we can bound the complexity of the algorithm by $O(|q|LK)$, where K is the number of paths allowed at each position in q.

With relative pruning, we only explore the paths that have probabilities higher than a certain percentage of the maximum probability at each position. The threshold values are carefully designed to achieve the best efficiency without causing a significant drop in accuracy. In practice we find relative pruning to be generally more effective for pruning unpromising paths. In our system, we make use of both absolute pruning and relative pruning to improve search efficiency and accuracy.

5.4 Thresholding

From the perspective of user interface, it is not always a good idea to show a predefined number of suggestions for every query.

Showing more suggestions incurs a cost, as users spend more time looking at them instead of completing their task. Moreover, showing irrelevant suggestions risks annoying users. Therefore, we need to make a binary decision for each suggestion on whether it should be shown to the user. Ideally, we want to measure the distance between the target query q and the suggested correction c. The larger the distance, the more risk we take to include it in the suggestions. One way to approximate the distance is to compute the log of the inverse transformation probability, averaged over the number of characters in the query:

$$\text{risk}(c, q) = \frac{1}{|q|} \log \frac{1}{p(c \to q)} \qquad (20)$$

This risk function is not very effective in practice, as the input query q usually consists of several words, of which only one is misspelled. It is unintuitive to average the risk over all letters in the query. Instead, we can first segment q into words and measure the risk at the word level. Specifically, we measure the risk of each word separately using the above formula and define the final risk function as the fraction of words in q having a risk value above a given threshold.

6. EXPERIMENT

6.1 Datasets

Our primary focus in this work is to build a transformation model that is able to capture all the misspelling behaviors of users. To obtain such behaviors, we make use of the click logs of search engine recourse links. Recourse links are provided when the offline correction mechanisms of search engines detect a potential misspelling. For example, in Google (Figure 6a), a recourse link is shown in the sentence "Did you mean: *important*". When the user clicks on this link, it indicates that the user agrees with the correction. Therefore, the search engine will use the suggested query to rerun the search. Similarly recourse links are provided in Bing as well (Figure 6b). By recording such clicks, we accumulate a set of high quality corrections that represent real user spelling behaviors.

It is worth noting that although the recourse links are provided by an offline spelling correction system, it does not mean that our ability will be limited to that of the offline system. First, our model captures the underlying patterns of spelling corrections instead of memorizing corrections at the word or query level. For instance, in the example from Figure 6, a possible pattern is that *im* tends to be misspelled as *in*. Second, our logs consist of recourse link clicks from multiple sites. As the spellers of different search engines behave differently, we can learn from a diverse set of correction pairs.
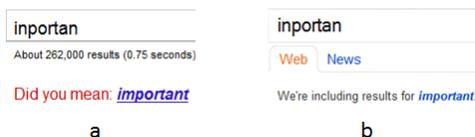
Figure 6. Examples of recourse links

There are also other ways to obtain records of spelling corrections. For example, by analyzing the webpage metadata for near-miss spellings, such those between title and anchor text, we can extract possible spelling corrections. Similarly, such corrections can also be obtained using click-through data from the query log, where a query-document mismatch would indicate a spelling error. In our work, we view the extraction of correction

records as a logical step that precedes transformation modeling. Our model can be easily extended to incorporate all sources of spelling correction pairs.

Our dataset for training the transformation model contains 1.4 million recourse link clicks. The statistics of the training data are shown in Table 2. Around 80% of all queries and 70% of all unique queries are correctly spelled. 1/10 of the training data is held out for parameter tuning.

Table 2. Statistics of training data

	Correctly Spelled	Misspelled	Total
Unique	101,640 (70%)	44,226 (30%)	145,866
Total	1,126,524 (80%)	283,854 (20%)	1,410,378

The query log we use for estimating the query popularity model consists of 21 million unique queries. Our test set is a human annotated set which contains 9,959 unique queries. Table 3 provides the statistics of the test data. The distribution over correctly spelled and misspelled queries is similar to that of the training data. 1/10 of the test data is also held out for tuning additional parameters, e.g. the coefficient λ for the mixture model. The remainder of the test queries is referred to as "all queries" in our evaluation results. The subset of misspelled queries within all queries is referred to as "misspelled queries".

Table 3. Statistics of the test data

	Correctly Spelled	Misspelled	Total
Unique	7585 (76%)	2374 (24%)	9959

6.2 Evaluation Metrics

We evaluate our methods with the following metrics:

R@N: Recall@N is the number of correct suggestions in the top ranked N suggestions generated by the system divided by the total number of suggestions in the ground truth. Since in our ground truth, each query has exactly one correction, the total number of suggestions is the same as the number of queries. Intuitively, Recall@N indicates the percentage of queries that the system can correct within the top N suggestions. Therefore, it is a very natural measurement for performance of correction. We take R@1 as our primary evaluation metric in experiment. Recall@N on all queries is also referred to as accuracy in other works [12].

P@N: Precision@N is the number of correct suggestions in the top ranked N suggestions generated by the system divided by the smaller value of N or the total number of suggestions generated by the system. Precision reveals the quality of suggestions generated by the system. Penalty is given to generating more incorrect suggestions. Note that this definition is different from another widely used definition of P@N, where the denominator is fixed to be N. Our definition can be interpreted as the *precision* of a system that limits its number of outputs to N at most. It is also worth noting that while the micro average and macro average for recall are the same, it is not the case for precision. For precision, we take the micro average because for queries where the system provides no suggestion, precision is not well defined.

R@N and P@N are metrics for measuring offline spelling correction. We use these metrics to evaluate our system in the *exact match mode*. Next, we introduce two metrics for measuring the performance of online spelling correction.

MKS: Minimal Keystrokes measures the minimal number of key presses the user has to make in order to issue the target search query. This metric simulates the scenario of users entering queries to search engines. Suppose the user's query is *inportan* and the correct query is *important*. The user types in each letter in *inportan* sequentially. In the case that no suggestion is available, the user types in all the letters in *inportan* and presses the Enter key. Then the user can click on the recourse link provided by the offline speller. Therefore, the total number of keystrokes the user makes is the length of *inportan*, plus 1 for the Enter key, and 1 for the recourse link click. When suggestions are provided while the user is typing, she can use arrow keys to select a query from the suggestion list. For example, after typing in *inpo*, the user sees that *important* appears at the fifth position in the suggestion list. Thus, she can select the query by pressing the Down Arrow key 5 times, followed by the Enter key. In this case, the number of keystrokes is the number of letters the user enters (4) plus the number of arrow keys the user hits (5), plus 1 for the Enter key. If the user continues typing the rest of the query, she may see *important* increase to rank one for the input *inpor*. In this case, the number of keystrokes is 5+1+1=7, which is the minimal number of keystrokes (MKS). A good correction mechanism should have low MKS. In our experiments, we consider superstrings of the target query as positive matches, too. That is, in the case that "*important people*" is suggested instead of "*important*", we still treat it as a match.

PMKS: PMKS refers to penalized MKS, which adds a penalty to MKS for each suggestion generated by the system, as it takes effort for users to examine them for correctness. In this work we heuristically assign 0.1 keystrokes as the penalty for showing each suggestion. Thus, reading each query suggestion costs one tenth the effort of pressing a key. The essential idea of minimizing effort in MKS and PMKS is of independent research interest and could be applied to a wide range of research studies.

6.3 Experimental Results

In this subsection, we study the performance of our proposed system. We conduct all experiments on both the all queries and misspelled queries test sets to demonstrate the overall performance as well as the ability to handle misspelled queries.

We first compare our system with existing baselines in Table 4. The first baseline we include is the edit distance model used by Chaudhuri and Kaushik [6]. To the best of our knowledge, this is the only existing research study on online spelling correction. Our system outperforms the edit distance model in terms of all evaluation metrics. Significance test (*t*-test) shows that the improvement of our system is significant (*p*-value < 0.05) for all measurements except R@10. This indicates that most misspellings are not very severe; therefore the edit distance model is able to rank the best correction among the top 10 suggestions. However, the edit distance model is not able to further distinguish corrections within the same edit distance. We further observe that although we see a big gap in R@1 for misspelled queries, the overall performance difference for all queries is less than that of the misspelled queries. This is expected as the edit distance model will always rank identical transformation on top (if it exists in the query log).

We also include Google's online query spelling suggestions[1] as a baseline. As it is unclear how Google's online spelling suggestion can be configured to run in exact match mode, we only measure

---

[1] Based on results collected on August 4, 2010.

its performance with respect to the online correction metric, i.e. MKS. Not surprisingly, Google outperforms the simple edit-distance model. On average users save 0.38 keystrokes per query using Google's spelling suggestions over that of the edit distance model. For misspelled queries, nearly 1 keystroke is saved. Yet, our system further outperforms Google's suggestion system on MKS with a statistically significant 1.1 and 1.5 keystrokes savings on all queries and misspelled queries, respectively. It is worth noting that a larger search space (query log in our case) may result in worse performance. Since the size of Google's search space is unknown, we cannot jump to the conclusion that our system outperforms Google's spelling suggestion system.

Table 4. Comparison of performance with baseline systems

	All Queries			Misspelled Queries		
	R@1	R@10	MKS	R@1	R@10	MKS
EditDist	0.899	0.973	13.39	0.579	0.887	14.53
Google	N/A	N/A	13.01	N/A	N/A	13.49
Proposed	**0.918**	**0.976**	**11.86**	**0.677**	**0.900**	**11.96**

We also see in this experiment that the MKS metric is fairly consistent with Recall. Higher recall values always correspond to lower MKS. This validates the use of MKS as a performance metric.
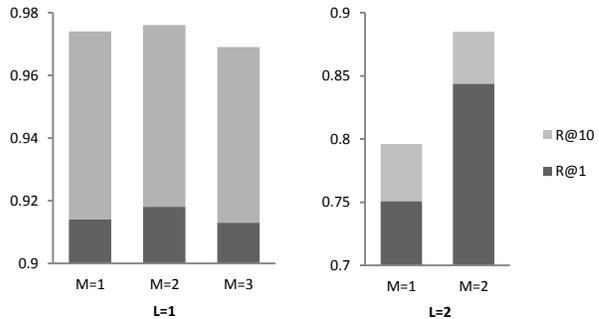
Figure 7: Performance with varying *L* and *M*

To further understand how the proposed method works, we study the performance of the transformation model with different configurations of *L* and *M*. Figure 7 shows the effect of the transfeme Markov order *M* at $L = 1$ and $L = 2$. As we increase *M* from 1 to 2, we see a consistent increase in performance; but from 2 to 3, the performance decreased instead. This is contradictory with our intuition that higher order models result in better performance. We believe that this is because higher order models are more likely to suffer from data sparseness. Thus, with more training data, we may find higher order models to further improve the performance over $M = 2$. We also observe that for a fixed *M*, increasing *L* actually decreases the performance. We hypothesize that this may be due to overfitting, as increasing *L* significantly increases the number of model parameters. As larger *L* also significantly increases the cost of search, it is impractical for real-time scenarios. Under the current setting, our best result is achieved with $L = 1, M = 2$. Thus for all subsequent experiments, we fix the configuration to $L = 1, M = 2$.

To confirm the effect of smoothing, we experiment with two smoothing methods and compare their performance. In Figure 8

we see that absolute discounting (AD) outperforms Jelinek-Mercer (JM) smoothing over every evaluation metric for both the all queries and misspelled queries test sets. This is in line with previous language modeling research that found discounting based smoothing to outperform simple interpolation techniques. This experiment confirms our hypothesis that employing proper smoothing methods substantially increases the performance of the transformation model.
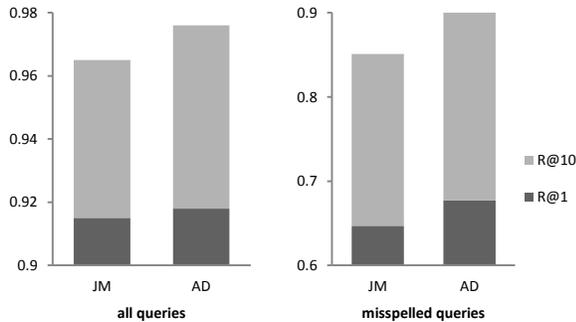
Figure 8: Performance of transformation models with different smoothing methods

We present the effectiveness of our proposed methods for avoiding over correction in Table 5. As we can see, the non-mixture model, which is trained with misspelled queries only, performs well for misspelled queries. However, the overall performance is not good because it tends to alter queries that are already correctly spelled. Both the data mixture and model mixture approaches improve the overall performance by reducing such overcorrections. For the all queries set, they perform equally well. For misspelled queries, model mixture performs just as well as the non-mixture model. However the performance of the data mixture approach drops significantly. From an application perspective, it is the misspelled queries for which users need suggestions the most. Users are able to enter queries that they can spell no matter what our system suggests. In this sense, the model mixture approach is more preferable than the data mixture approach. Moreover, by estimating the two models separately, the model mixture approach can be updated more easily.

Table 5. Performance study on overcorrection

	All Queries			Misspelled Queries		
	R@1	R@10	MKS	R@1	R@10	MKS
Non-Mix	0.893	0.966	11.94	**0.678**	0.899	11.98
Data Mix	**0.918**	0.971	**11.85**	0.669	0.879	11.98
Model Mix	**0.918**	**0.976**	11.86	0.677	**0.900**	**11.96**

In Table 8, we study the effect of the proposed thresholding method for pruning irrelevant suggestions. As we can see, with suggestion pruning, the performance of online spelling correction substantially increases for both the all queries and misspelled queries sets in terms of P@1, P@10 and PMKS. This verifies the effectiveness of our proposed thresholding method. But in terms of R@1, R@10 and MKS, the performance actually decreased. The reason behind this pattern is that the first set of metrics (P@1, P@10 and PMKS) assigns penalty for showing irrelevant suggestions, while the second set of metrics does not. In fact, any

pruning of suggestions can only decrease the recall, as some correct suggestions may be pruned by mistake. From our perspective, showing too many irrelevant corrections has a strong negative effect on the query completion user experience, increasing the risk of losing users. Given that the recall did not significantly decrease, we prune suggestions using risk thresholding in the implementation of our system.
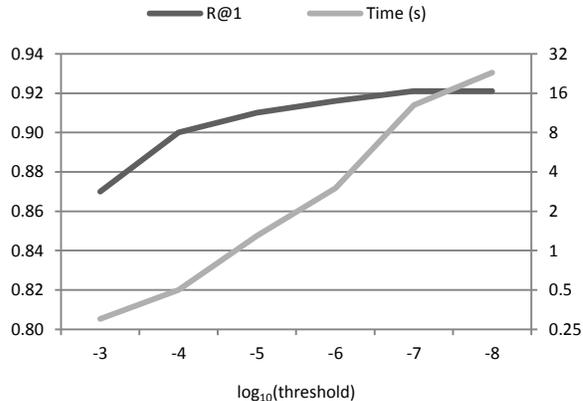
Figure 9: R@1/Running time vs. pruning threshold

Finally, we address the efficiency of our approach. From our experiments, we observe that although a better heuristic function can reduce the running time of the search algorithm, beam pruning is still required to achieve practical performance. In Figure 9 we plot the performance and running times for different relative beam pruning thresholds. Based on our experiments on an unoptimized implementation, we observe that as we relax the pruning threshold, the running time increases exponentially. However, the increase in R@1 is slow and ceases beyond a relative threshold of 10^{-7}.

Table 6. Examples suggestions

Input Query	Top Suggestion
milk shak	milkshake recipes
hwo to tain ur dra	how to train your dragon
alice on wander land	alice in wonderland
mision inpos	mission impossible

In Table 6 we list some example correction pairs identified by our system. None of these input queries are in the training corpus. As we can see, our method is capable of capturing various kinds of spelling errors for multiple word phrases. By updating the query language model frequently, we can keep our online spelling correction system up-to-date with the latest query language.

Table 7. Examples of transfeme probabilities

$M = 1$		$M = 2$	
$p(a \rightarrow u)$	0.0001	$p(a \rightarrow u \mid h \rightarrow h)$	0.0006
$p(u \rightarrow a)$	0.0002	$p(u \rightarrow a \mid a \rightarrow u)$	0.2
$p(e \rightarrow a)$	0.002	$p(e \rightarrow a \mid a \rightarrow u)$	0.007

To further understand the internal mechanism of our model, we list some transfeme probabilities in Table 7. Clearly, for $M = 1$,

Table 8. Effect of Pruning

	all queries						misspelled queries					
	R@1	R@10	P@1	P@10	MKS	PMKS	R@1	R@10	P@1	P@10	MKS	PMKS
w/ pruning	0.916	0.969	**0.927**	**0.304**	11.87	**19.42**	0.669	0.875	**0.704**	**0.241**	12.00	**19.21**
w/o pruning	**0.918**	**0.976**	0.920	0.262	**11.86**	19.60	**0.677**	**0.900**	0.685	0.204	**11.96**	19.56

$p(au \rightarrow ua) = p(a \rightarrow u) \cdot p(u \rightarrow a)$ is much smaller than $p(ae \rightarrow ua)$. But with $M = 2$, $p(au \rightarrow ua|h \rightarrow h)$ is significantly larger than $p(ae \rightarrow ua|h \rightarrow h)$. This is desirable as $au \rightarrow ua$ is a more common mistake (e.g. "haul" vs "hual") than $ae \rightarrow ua$.

7. CONCLUSION

This paper addresses the problem of online spelling correction for search queries by adopting a generative model for query correction. We first propose a transfeme based transformation model that is capable of capturing users' spelling behavior. We estimate the transformation model using clicks on search engine recourse links, which represent user confirmed query misspellings. Next, we study various techniques to optimize the effectiveness of the transformation model.

To efficiently retrieve the query corrections with the highest probability according to the generative model, we propose the use of an algorithm based on A* search. The A* search algorithm is configured to deal with partial queries, so that online search is possible. We study different pruning and thresholding methods to improve the efficiency of the A* search.

Finally, we propose two evaluation metrics for online spelling correction, minimal keystrokes and penalized minimal keystrokes, based on the idea of minimal effort cost for users. We conduct extensive experiments and conclude that the proposed method is both effective and efficient for the task of online spelling correction.

For future work, we plan to explore the use of other sources of spelling correction pairs to more robustly estimate the transformation models. For example, we will consider the use of webpage metadata, including title and anchor texts, to extract correction pairs. We also plan to extend our model by incorporating a large scale language model [18] so that we can suggest query corrections that have never been seen before.

8. REFERENCES

[1] http://en.wikipedia.org/wiki/Fat-finger

[2] http://en.wikipedia.org/wiki/I_before_E_except_after_C

[3] J. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *Technical Report ICSI-TR-97-021*. 1997.

[4] M. Bisani and H. Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, Vol. 50. 2008.

[5] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, J. Yuan. Online expansion of rare queries for sponsored search. In *WWW*, 2009.

[6] S. Chaudhuri and R. Kaushik. Extending auto-completion to tolerate errors. In *SIGMOD*, 2009.

[7] Q. Chen, M. Li, and M. Zhou. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, 2007.

[8] S. F. Chen. Conditional and joint models for grapheme-to-phoneme conversion. In *Eurospeech*, 2003.

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

[10] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communication of ACM*. Vol. 7. 1964.

[11] J. Darragh, I. Witten, and M. James. The reactive keyboard: a predictive typing aid. *Computer*. Vol. 11. 1990.

[12] J. Gao, X. Li, D. Micol, C. Quirk and X. Sun. A large scale ranker-based system for search query spelling correction. In *COLING*, 2010.

[13] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*. 1966.

[14] M. Li, Y. Zhang , M. Zhu , M. Zhou. Exploring distributional similarity based models for query spelling correction. In *ACL*, 2006.

[15] E.M. Rieseman and A.R. Hanson. A contextual postprocessing system for error correction using binary *n*-grams. *IEEE Transactions on Computers*. Vol. 23. 1974.

[16] X. Sun, J. Gao, D. Micol and C. Quirk. Learning phrase-based spelling error models from clickthrough data. In *ACL*, 2010.

[17] P. Taylor. 2005. Hidden Markov models for grapheme to phoneme conversion. In *Eurospeech*, 2005.

[18] K. Wang, X. Li, and J. Gao. Multi-style language model for web scale information retrieval. In *SIGIR*, 2010.