A Graph Approach to Spelling Correction in Domain-Centric Search

Zhuowei Bao
University of Pennsylvania
Philadelphia, PA 19104, USA
zhuowei@cis.upenn.edu

Benny Kimelfeld
IBM Research–Almaden
San Jose, CA 95120, USA
kimelfeld@us.ibm.com

Yunyao Li
IBM Research–Almaden
San Jose, CA 95120, USA
yunyaoli@us.ibm.com

Abstract

Spelling correction for keyword-search queries is challenging in restricted domains such as personal email (or desktop) search, due to the scarcity of query logs, and due to the specialized nature of the domain. For that task, this paper presents an algorithm that is based on statistics from the corpus data (rather than the query log). This algorithm, which employs a simple graph-based approach, can incorporate different types of data sources with different levels of reliability (e.g., email subject vs. email body), and can handle complex spelling errors like splitting and merging of words. An experimental study shows the superiority of the algorithm over existing alternatives in the email domain.

1 Introduction

An abundance of applications require *spelling correction*, which (at the high level) is the following task. The user intends to type a chunk $\mathbf{q}$ of text, but types instead the chunk $\mathbf{s}$ that contains *spelling errors* (which we discuss in detail later), due to uncareful typing or lack of knowledge of the exact spelling of $\mathbf{q}$. The goal is to restore $\mathbf{q}$, when given $\mathbf{s}$. Spelling correction has been extensively studied in the literature, and we refer the reader to comprehensive summaries of prior work (Peterson, 1980; Kukich, 1992; Jurafsky and Martin, 2000; Mitton, 2010). The focus of this paper is on the special case where $\mathbf{q}$ is a *search query*, and where $\mathbf{s}$ instead of $\mathbf{q}$ is submitted to a search engine (with the goal of retrieving documents that match the search query $\mathbf{q}$). Spelling correction for search queries is important, because a significant portion of posed queries may be misspelled (Cucerzan and Brill, 2004). Effective spelling correction has a major effect on the experience and effort of the user, who is otherwise required to ensure the exact spellings of her queries. Furthermore, it is critical when the exact spelling is unknown (e.g., person names like *Schwarzenegger*).

1.1 Spelling Errors

The more common and studied type of spelling error is *word-to-word* error: a single word w is misspelled into another single word w'. The specific spelling errors involved include omission of a character (e.g., `atachment`), inclusion of a redundant character (e.g., `attachement`), and replacement of characters (e.g., `attachemnt`). The fact that w' is a misspelling of (and should be corrected to) w is denoted by $w' \rightarrow w$ (e.g., `atachment` $\rightarrow$ `attachment`).

Additional common spelling errors are *splitting* of a word, and *merging* two (or more) words:

- `attach ment` $\rightarrow$ `attachment`
- `emailattachment` $\rightarrow$ `email attachment`

Part of our experiments, as well as most of our examples, are from the domain of (personal) email search. An email from the *Enron email collection* (Klimt and Yang, 2004) is shown in Figure 1. Our running example is the following misspelling of a search query, involving multiple types of errors.

`sadeep kohli excellatach ment` $\rightarrow$
`sandeep kohli excel attachment` (1)

In this example, correction entails fixing `sadeep`, splitting `excellatach`, fixing `excell`, merging `atach ment`, and fixing `atachment`. Beyond the complexity of errors, this example also illustrates other challenges in spelling correction for search. We need to identify not only that `sadeep` is misspelled, but also that `kohli` is correctly spelled. Just having `kohli` in a dictionary is not enough.

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pages 905–914,
Portland, Oregon, June 19-24, 2011.

Subject: Follow-Up on Captive Generation
From: sandeep.kohli@enron.com
X-From: Sandeep Kohli
X-To: Stinson Gibner@ECT, Vince J Kaminski@ECT

Vince/Stinson,

Please find below two attachemnts. The Excell spreadsheet shows some calculations... The seond attachement (Word) has the wordings that I think we can send in to the press...

I am availabel on mobile if you have questions o clarifications...

Regards,
Sandeep.

Figure 1: Enron email (misspelled words are underlined)

For example, in `kohli coupons` the user may very well mean `kohls coupons` if Sandeep Kohli has nothing to do with coupons (in contrast to the store chain Kohl's). A similar example is the word `nail`, which is a legitimate English word, but in the context of email the query `nail box` is likely to be a misspelling of `mail box` (unless nail boxes are indeed relevant to the user's email collection). Finally, while the word `kohli` is relevant to some email users (e.g., Kohli's colleagues), it may have no meaning at all to other users.

1.2 Domain Knowledge

The common approach to spelling correction utilizes statistical information (Kernighan et al., 1990; Schierle et al., 2007; Mitton, 2010). As a simple example, if we want to avoid maintaining a manually-crafted dictionary to accommodate the wealth of new terms introduced every day (e.g., `ipod` and `ipad`), we may decide that `atachment` is a misspelling of `attachment` due to both the (relative) proximity between the words, and the fact that `attachment` is significantly more popular than `atachment`. As another example, the fact that the expression `sandeep kohli` is frequent in the domain increases our confidence in `sadeep kohli` $\rightarrow$ `sandeep kohli` (rather than, e.g., `sadeep kohli` $\rightarrow$ `sudeep kohli`). One can further note that, in email search, the fact that Sandeep Kohli sent multiple excel attachments increases our confidence in `excell` $\rightarrow$ `excel`.

A source of statistics widely used in prior work is the query log (Cucerzan and Brill, 2004; Ahmad and Kondrak, 2005; Li et al., 2006a; Chen et al., 2007; Sun et al., 2010). However, while query logs are abundant in the context of Web search, in many other search applications (e.g. email search, desktop search, and even small-enterprise search) query logs are too scarce to provide statistical information that is sufficient for effective spelling correction. Even an email provider of a massive scale (such as GMail) may need to rely on the (possibly tiny) query log of the single user at hand, due to privacy or security concerns; moreover, as noted earlier about `kohli`, the statistics of one user may be relevant to one user, while irrelevant to another.

The focus of this paper is on spelling correction for search applications like the above, where query-log analysis is impossible or undesirable (with email search being a prominent example). Our approach relies mainly on the corpus data (e.g., the collection of emails of the user at hand) and external, generic dictionaries (e.g., English). As shown in Figure 1, the corpus data may very well contain misspelled words (like query logs do), and such noise is a part of the challenge. Relying on the corpus has been shown to be successful in spelling correction for *text cleaning* (Schierle et al., 2007). Nevertheless, as we later explain, our approach can still incorporate query-log data as features involved in the correction, as well as means to refine the parameters.

1.3 Contribution and Outline

As said above, our goal is to devise spelling correction that relies on the corpus. The corpus often contains various types of information, with different levels of reliability (e.g., n-grams from email subjects and sender information, vs. those from email bodies). The major question is how to effectively exploit that information while addressing the various types of spelling errors such as those discussed in Section 1.1. The key contribution of this work is a novel graph-based algorithm, MaxPaths, that handles the different types of errors and incorporates the corpus data in a uniform (and simple) fashion. We describe MaxPaths in Section 2. We evaluate the effectiveness of our algorithm via an experimental study in Section 3. Finally, we make concluding remarks and discuss future directions in Section 4.

2 Spelling-Correction Algorithm

In this section, we describe our algorithm for spelling correction. Recall that given a search query

$\mathbf{s}$ of a user who intends to phrase $\mathbf{q}$, the goal is to find $\mathbf{q}$. Our corpus is essentially a collection $\mathbf{D}$ of unstructured or *semistructured documents*. For example, in email search such a document is an email with a title, a body, one or more recipients, and so on. As conventional in spelling correction, we devise a scoring function $\text{score}_{\mathbf{D}}(\mathbf{r} \mid \mathbf{s})$ that estimates our confidence in $\mathbf{r}$ being the correction of $\mathbf{s}$ (i.e., that $\mathbf{r}$ is equal to $\mathbf{q}$). Eventually, we suggest a sequence $\mathbf{r}$ from a set $\mathcal{C}_{\mathbf{D}}(\mathbf{s})$ of *candidates*, such that $\text{score}_{\mathbf{D}}(\mathbf{r} \mid \mathbf{s})$ is maximal among all the candidates in $\mathcal{C}_{\mathbf{D}}(\mathbf{s})$. In this section, we describe our graph-based approach to finding $\mathcal{C}_{\mathbf{D}}(\mathbf{s})$ and to determining $\text{score}_{\mathbf{D}}(\mathbf{r} \mid \mathbf{s})$.

We first give some basic notation. We fix an alphabet $\mathbf{\Sigma}$ of *characters* that does not include any of the conventional whitespace characters. By $\mathbf{\Sigma}^*$ we denote the set of all the *words*, namely, finite sequences over $\mathbf{\Sigma}$. A *search query* $\mathbf{s}$ is a sequence $w_1, \ldots, w_n$, where each w_i is a word. For convenience, in our examples we use whitespace instead of comma (e.g., `sandeep kohli` instead of `sandeep, kohli`). We use the *Damerau-Levenshtein* edit distance (as implemented by the *Jazzy* tool) as our primary edit distance between two words $r_1, r_2 \in \mathbf{\Sigma}^*$, and we denote this distance by $ed(r_1, r_2)$.

2.1 Word-Level Correction

We first handle a restriction of our problem, where the search query is a single word w (rather than a general sequence $\mathbf{s}$ of words). Moreover, we consider only candidate suggestions that are words (rather than sequences of words that account for the case where w is obtained by merging keywords). Later, we will use the solution for this restricted problem as a basic component in our algorithm for the general problem.

Let $\mathbf{U}_{\mathbf{D}} \subseteq \mathbf{\Sigma}^*$ be a finite *universal lexicon*, which (conceptually) consists of all the words in the corpus $\mathbf{D}$. (In practice, one may want add to $\mathbf{D}$ words of auxiliary sources, like English dictionary, and to filter out noisy words; we did so in the site-search domain that is discussed in Section 3.) The set $\mathcal{C}_{\mathbf{D}}(w)$ of candidates is defined by

$$\mathcal{C}_{\mathbf{D}}(w) \stackrel{\text{def}}{=} \{w\} \cup \{w' \in \mathbf{U}_{\mathbf{D}} \mid ed(w, w') \leq \delta\}\,.$$

for some fixed number δ. Note that $\mathcal{C}_{\mathbf{D}}(w)$ contains w even if w is misspelled; furthermore, $\mathcal{C}_{\mathbf{D}}(w)$ may contain other misspelled words (with a small edit distance to w) that appear in $\mathbf{D}$.

Table 1: Feature set $\mathsf{WF}_{\mathbf{D}}$ in email search

Basic Features
$ed(w, w')$: weighted Damerau-Levenshtein edit distance
$ph(w, w')$: 1 if w and w' are phonetically equal, 0 otherwise
$\text{english}(w')$: 1 is w' is in English, 0 otherwise
Corpus-Based Features
$logfreq(w')$): logarithm of #occurrences of w' in the corpus
Domain-Specific Features
$\text{subject}(w')$: 1 if w' is in some "Subject" field, 0 otherwise
$\text{from}(w')$: 1 if w' is in some "From" field, 0 otherwise
$\text{xfrom}(w')$: 1 if w' is in some "X-From" field, 0 otherwise

We now define $\text{score}_{\mathbf{D}}(w' \mid w)$. Here, our corpus $\mathbf{D}$ is translated into a set $\mathsf{WF}_{\mathbf{D}}$ of *word features*, where each feature $f \in \mathsf{WF}_{\mathbf{D}}$ gives a scoring function $\text{score}_f(w' \mid w)$. The function $\text{score}_{\mathbf{D}}(w' \mid w)$ is simply a linear combination of the $\text{score}_f(w' \mid w)$:

$$\text{score}_{\mathbf{D}}(w' \mid w) \stackrel{\text{def}}{=} \sum_{f \in \mathsf{WF}_{\mathbf{D}}} a_f \cdot \text{score}_f(w' \mid w)$$

As a concrete example, the features of $\mathsf{WF}_{\mathbf{D}}$ we used in the email domain are listed in Table 1; the resulting $\text{score}_f(w'|w)$ is in the spirit of the *noisy channel model* (Kernighan et al., 1990). Note that additional features could be used, like ones involving the *stems* of w and w', and even query-log statistics (when available). Rather than manually tuning the parameters a_f, we learned them using the well known *Support Vector Machine*, abbreviated *SVM* (Cortes and Vapnik, 1995), as also done by Schaback and Li (2007) for spelling correction. We further discuss this learning step in Section 3.

We fix a natural number k, and in the sequel we denote by $\text{top}_{\mathbf{D}}(w)$ a set of k words $w' \in \mathcal{C}_{\mathbf{D}}(w)$ with the highest $\text{score}_{\mathbf{D}}(w' \mid w)$. If $|\mathcal{C}_{\mathbf{D}}(w)| < k$, then $\text{top}_{\mathbf{D}}(w)$ is simply $\mathcal{C}_{\mathbf{D}}(w)$.

2.2 Query-Level Correction: MaxPaths

We now describe our algorithm, MaxPaths, for spelling correction. The input is a (possibly misspelled) search query $\mathbf{s} = s_1, \ldots, s_n$. As done in the word-level correction, the algorithm produces a set $\mathcal{C}_{\mathbf{D}}(\mathbf{s})$ of suggestions and determines the values

Algorithm 1 MaxPaths

Input: a search query $\mathbf{s}$
Output: a set $\mathcal{C}_\mathbf{D}(\mathbf{s})$ of candidate suggestions $\mathbf{r}$, ranked by $\text{score}_\mathbf{D}(\mathbf{r} \mid \mathbf{s})$

1: Find the *strongly plausible tokens*
2: Construct the *correction graph*
3: Find top-k *full paths* (with the largest weights)
4: Re-rank the paths by *word correlation*

$\text{score}_\mathbf{D}(\mathbf{r} \mid \mathbf{s})$, for all $\mathbf{r} \in \mathcal{C}_\mathbf{D}(\mathbf{s})$, in order to rank $\mathcal{C}_\mathbf{D}(\mathbf{s})$. A high-level overview of MaxPaths is given in the pseudo-code of Algorithm 1. In the rest of this section, we will detail each of the four steps in Algorithm 1. The name MaxPaths will become clear towards the end of this section.

We use the following notation. For a word $w = c_1 \cdots c_m$ of m characters c_i and integers $i < j$ in $\{1, \ldots, m+1\}$, we denote by $w_{[i,j)}$ the word $c_i \cdots c_{j-1}$. For two words $w_1, w_2 \in \mathbf{\Sigma}^*$, the word $w_1 w_2 \in \mathbf{\Sigma}^*$ is obtained by concatenating w_1 and w_2. Note that for the search query $\mathbf{s} = s_1, \ldots, s_n$ it holds that $s_1 \cdots s_n$ is a single word (in $\mathbf{\Sigma}^*$). We denote the word $s_1 \cdots s_n$ by $\lfloor \mathbf{s} \rfloor$. For example, if $s_1 =$ `sadeep` and $s_2 =$ `kohli`, then $\mathbf{s}$ corresponds to the query `sadeep kohli` while $\lfloor \mathbf{s} \rfloor$ is the word `sadeepkohli`; furthermore, $\lfloor \mathbf{s} \rfloor_{[1,7)} =$ `sadeep`.

2.2.1 Plausible Tokens

To support merging and splitting, we first identify the possible *tokens* of the given query $\mathbf{s}$. For example, in `excellatach ment` we would like to identify `excell` and `atach ment` as tokens, since those are indeed the tokens that the user has in mind. Formally, suppose that $\lfloor \mathbf{s} \rfloor = c_1 \cdots c_m$. A *token* is a word $\lfloor \mathbf{s} \rfloor_{[i,j)}$ where $1 \leq i < j \leq m+1$. To simplify the presentation, we make the (often false) assumption that a token $\lfloor \mathbf{s} \rfloor_{[i,j)}$ uniquely identifies i and j (that is, $\lfloor \mathbf{s} \rfloor_{[i,j)} \neq \lfloor \mathbf{s} \rfloor_{[i',j')}$ if $i \neq i'$ or $j \neq j'$); in reality, we should define a token as a triple $(\lfloor \mathbf{s} \rfloor_{[i,j)}, i, j)$. In principle, every token $\lfloor \mathbf{s} \rfloor_{[i,j)}$ could be viewed as a possible word that user meant to phrase. However, such liberty would require our algorithm to process a search space that is too large to manage in reasonable time. Instead, we restrict to *strongly plausible tokens*, which we define next.

A token $w = \lfloor \mathbf{s} \rfloor_{[i,j)}$ is *plausible* if w is a word of $\mathbf{s}$, or there is a word $w' \in \mathcal{C}_\mathbf{D}(w)$ (as defined in Section 2.1) such that $\text{score}_\mathbf{D}(w' \mid w) > \epsilon$ for some fixed number ϵ. Intuitively, w is plausible if it is an original token of $\mathbf{s}$, or we have a high confidence in our word-level suggestion to correct w (note that the suggested correction for w can be w itself). Recall that $\lfloor \mathbf{s} \rfloor = c_1 \cdots c_m$. A *tokenization* of $\mathbf{s}$ is a sequence $j_1, \ldots, j_l$, such that $j_1 = 1$, $j_l = m+1$, and $j_i < j_{i+1}$ for $1 \leq i < l$. The tokenization $j_1, \ldots, j_l$ *induces* the tokens $\lfloor \mathbf{s} \rfloor_{[j_1,j_2)}, \ldots, \lfloor \mathbf{s} \rfloor_{[j_{l-1},j_l)}$. A tokenization is *plausible* if each of its induced tokens is plausible. Observe that a plausible token is not necessarily induced by any plausible tokenization; in that case, the plausible token is useless to us. Thus, we define a *strongly plausible token*, abbreviated *sp-token*, which is a token that is induced by some plausible tokenization. As a concrete example, for the query `excellatach ment`, the sp-tokens in our implementation include `excellatach`, `ment`, `excell`, and `atachment`.

As the first step (line 1 in Algorithm 1), we find the sp-tokens by employing an efficient (and fairly straightforward) dynamic-programming algorithm.

2.2.2 Correction Graph

In the next step (line 2 in Algorithm 1), we construct the *correction graph*, which we denote by $\mathcal{G}_\mathbf{D}(\mathbf{s})$. The construction is as follows.

We first find the set $\text{top}_\mathbf{D}(w)$ (defined in Section 2.1) for each sp-token w. Table 2 shows the sp-tokens and suggestions thereon in our running example. This example shows the actual execution of our implementation within email search, where $\mathbf{s}$ is the query `sadeep kohli excellatach ment`; for clarity of presentation, we omitted a few sp-tokens and suggested corrections. Observe that some of the corrections in the table are actually misspelled words (as those naturally occur in the corpus).

A node of the graph $\mathcal{G}_\mathbf{D}(\mathbf{s})$ is a pair $\langle w, w' \rangle$, where w is an sp-token and $w' \in \text{top}_\mathbf{D}(w)$. Recall our simplifying assumption that a token $\lfloor \mathbf{s} \rfloor_{[i,j)}$ uniquely identifies the indices i and j. The graph $\mathcal{G}_\mathbf{D}(\mathbf{s})$ contains a (directed) edge from a node $\langle w_1, w_1' \rangle$ to a node $\langle w_2, w_2' \rangle$ if w_2 immediately follows w_1 in $\lfloor \mathbf{q} \rfloor$; in other words, $\mathcal{G}_\mathbf{D}(\mathbf{s})$ has an edge from $\langle w_1, w_1' \rangle$ to $\langle w_2, w_2' \rangle$ whenever there exist indices i, j and k, such that $w_1 = \lfloor \mathbf{s} \rfloor_{[i,j)}$ and $w_2 = \lfloor \mathbf{s} \rfloor_{[j,k)}$. Observe that $\mathcal{G}_\mathbf{D}(\mathbf{s})$ is a *directed acyclic graph* (DAG).

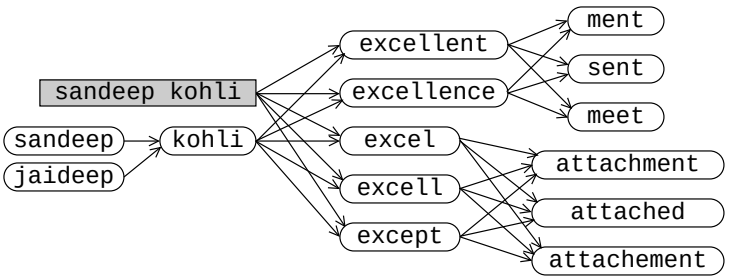


Figure 2: The graph $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$

For example, Figure 2 shows $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ for the query `sadeep kohli excellatach ment`, with the sp-tokens w and the sets $\text{top}_{\mathbf{D}}(w)$ being those of Table 2. For now, the reader should ignore the node in the grey box (containing `sandeep kohli`) and its incident edges. For simplicity, in this figure we depict each node $\langle w, w' \rangle$ by just mentioning w'; the word w is in the first row of Table 2, above w'.

2.2.3 Top-k Paths

Let $P = \langle w_1, w'_1 \rangle \rightarrow \cdots \rightarrow \langle w_k, w'_k \rangle$ be a path in $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$. We say that P is *full* if $\langle w_1, w'_1 \rangle$ has no incoming edges in $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$, and $\langle w_k, w'_k \rangle$ has no outgoing edges in $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$. An easy observation is that, since we consider only strongly plausible tokens, if P is full then $w_1 \cdots w_k = \lfloor \mathbf{s} \rfloor$; in that case, the sequence $w'_1, \ldots, w'_k$ is a suggestion for spelling correction, and we denote it by $crc(P)$. As an example, Figure 3 shows two full paths P_1 and P_2 in the graph $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ of Figure 2. The corrections $crc(P_i)$, for $i = 1, 2$, are `jaideep kohli excellent ment` and `sandeep kohli excel attachement`, respectively.

To obtain corrections $crc(P)$ with high quality, we produce a set of k full paths with the largest weights, for some fixed k; we denote this set by $\text{topPaths}_{\mathbf{D}}(\mathbf{s})$. The *weight* of a path P, denoted $\text{weight}(P)$, is the sum of the weights of all the nodes and edges in P, and we define the weights of nodes and edges next. To find these paths, we use a well known efficient algorithm (Eppstein, 1994).

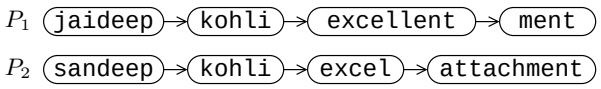


Figure 3: Full paths in the graph $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ of Figure 2

Consider a node $u = \langle w, w' \rangle$ of $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$. In the construction of $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$, zero or more merges of (part of) original tokens have been applied to obtain the token w; let $\#\text{merges}(w)$ be that number. Consider an edge e of $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ from a node $u_1 = \langle w_1, w'_1 \rangle$ to $u_2 = \langle w_2, w'_2 \rangle$. In $\mathbf{s}$, either w_1 and w_2 belong to different words (i.e., there is a whitespace between them) or not; in the former case define $\#\text{splits}(e) = 0$, and in the latter $\#\text{splits}(e) = 1$. We define:

$$\text{weight}(u) \stackrel{\text{def}}{=} \text{score}_{\mathbf{D}}(w' \mid w) + a_m \cdot \#\text{merges}(w)$$
$$\text{weight}(e) \stackrel{\text{def}}{=} a_s \cdot \#\text{splits}(e)$$

Note that a_m and a_s are negative, as they penalize for merges and splits, respectively. Again, in our implementations, we learned a_m and a_s by means of SVM.

Recall that $\text{topPaths}_{\mathbf{D}}(\mathbf{s})$ is the set of k full paths (in the graph $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$) with the largest weights. From $\text{topPaths}_{\mathbf{D}}(\mathbf{s})$ we get the set $\mathcal{C}_{\mathbf{D}}(\mathbf{s})$ of candidate suggestions:

$$\mathcal{C}_{\mathbf{D}}(\mathbf{s}) \stackrel{\text{def}}{=} \{crc(P) \mid P \in \text{topPaths}_{\mathbf{D}}(\mathbf{s})\}.$$

2.2.4 Word Correlation

To compute $\text{score}_{\mathbf{D}}(\mathbf{r}|\mathbf{s})$ for $\mathbf{r} \in \mathcal{C}_{\mathbf{D}}(\mathbf{s})$, we incorporate *correlation* among the words of $\mathbf{r}$. Intuitively, we would like to reward a candidate with pairs of words that are likely to co-exist in a query. For that, we assume a (symmetric) numerical function $crl(w'_1, w'_2)$ that estimates the extent to which the words w'_1 and w'_2 are correlated. As an example, in the email domain we would like $crl(\texttt{kohli}, \texttt{excel})$ to be high if Kohli sent many emails with excel attachments. Our implementation of $crl(w'_1, w'_2)$ essentially employs *pointwise mutual information* that has also been used in (Schierle et al., 2007), and that

Table 2: $\text{top}_{\mathbf{D}}(w)$ for sp-tokens w

sadeep	kohli	excellatach	ment	excell	atachment
sandeep	kohli	excellent	ment	excel	attachment
jaideep		excellence	sent	excell	attached
			meet	except	attachement

compares the number of documents (emails) containing w'_1 and w'_2 separately and jointly.

Let $P \in \text{topPaths}_{\mathbf{D}}(\mathbf{s})$ be a path. We denote by $crl(P)$ a function that aggregates the numbers $crl(w'_1, w'_2)$ for nodes $\langle w_1, w'_1 \rangle$ and $\langle w_2, w'_2 \rangle$ of P (where $\langle w_1, w'_1 \rangle$ and $\langle w_2, w'_2 \rangle$ are not necessarily neighbors in P). Over the email domain, our $crl(P)$ is the minimum of the $crl(w'_1, w'_2)$. We define $\text{score}_{\mathbf{D}}(P) = \text{weight}(P) + crl(P)$. To improve the performance, in our implementation we learned again (re-trained) all the parameters involved in $\text{score}_{\mathbf{D}}(P)$.

Finally, as the top suggestions we take $crc(P)$ for full paths P with highest $\text{score}_{\mathbf{D}}(P)$. Note that $crc(P)$ is not necessarily injective; that is, there can be two full paths $P_1 \neq P_2$ satisfying $crc(P_1) = crc(P_2)$. Thus, in effect, $\text{score}_{\mathbf{D}}(\mathbf{r} \mid \mathbf{s})$ is determined by the best evidence of $\mathbf{r}$; that is,

$$\text{score}_{\mathbf{D}}(\mathbf{r} \mid \mathbf{s}) \stackrel{\text{def}}{=} \max\{\text{score}_{\mathbf{D}}(P) \mid crc(P) = \mathbf{r} \wedge P \in \text{topPaths}_{\mathbf{D}}(\mathbf{s})\}.$$

Note that our final scoring function essentially views P as a *clique* rather than a path. In principle, we could define $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ in a way that we would extract the maximal cliques directly without finding $\text{topPaths}_{\mathbf{D}}(\mathbf{s})$ first. However, we chose our method (finding top paths first, and then re-ranking) to avoid the inherent computational hardness involved in finding maximal cliques.

2.3 Handling Expressions

We now briefly discuss our handling of frequent n-grams (expressions). We handle n-grams by introducing new nodes to the graph $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$; such a new node u is a pair $\langle \mathbf{t}, \mathbf{t}' \rangle$, where $\mathbf{t}$ is a sequence of n consecutive sp-tokens and $\mathbf{t}'$ is a n-gram. The weight of such a node u is rewarded for constituting a frequent or important n-gram. An example of such a node is in the grey box of Figure 2, where `sandeep kohli` is a bigram. Observe that `sandeep kohli` may be deemed an important bigram because it occurs as a sender of an email, and not necessarily because it is frequent.

An advantage of our approach is avoidance of over-scoring due to *conflicting n-grams*. For example, consider the query `textile import expert`, and assume that both `textile import` and `import export` (with an "o" rather than an "e") are frequent bigrams. If the user referred to the bigram `textile import`, then `expert` is likely to be correct. But if she meant for `import export`, then `expert` is misspelled. However, only one of these two options can hold true, and we would like `textile import export` to be rewarded only once—for the bigram `import export`. This is achieved in our approach, since a full path in $\mathcal{G}_{\mathbf{D}}(\mathbf{s})$ may contain either a node for `textile import` or a node for `import export`, but it cannot contain nodes for both of these bigrams.

Finally, we note that our algorithm is in the spirit of that of Cucerzan and Brill (2004), with a few inherent differences. In essence, a node in the graph they construct corresponds to what we denote here as $\langle w, w' \rangle$ in the special case where w is an actual word of the query; that is, no re-tokenization is applied. They can split a word by comparing it to a bigram. However, it is not clear how they can split into non-bigrams (without a huge index) and to handle simultaneous merging and splitting as in our running example (1). Furthermore, they translate bigram information into edge weights, which implies that the above problem of over-rewarding due to conflicting bigrams occurs.

3 Experimental Study

Our experimental study aims to investigate the effectiveness of our approach in various settings, as we explain next.

3.1 Experimental Setup

We first describe our experimental setup, and specifically the datasets and general methodology.

Datasets. The focus of our experimental study is on *personal email search*; later on (Section 3.6), we will consider (and give experimental results for) a totally different setting—site search over `www.ibm.com`, which is a massive and open domain. Our dataset (for the email domain) is obtained from

the Enron email collection (Bekkerman et al., 2004; Klimt and Yang, 2004). Specifically, we chose the three users with the largest number of emails. We refer to the three email collections by the last names of their owners: *Farmer*, *Kaminski* and *Kitchen*. Each user mailbox is a separate domain, with a separate corpus $\mathbf{D}$, that one can search upon. Due to the absence of real user queries, we constructed our dataset by conducting a user study, as described next.

For each user, we randomly sampled 50 emails and divided them into 5 disjoint sets of 10 emails each. We gave each 10-email set to a unique human subject that was asked to phrase two search queries for each email: one for the entire email content (*general query*), and the other for the `From` and `X-From` fields (*sender query*). (Figure 1 shows examples of the `From` and `X-From` fields.) The latter represents queries posed against a specific field (e.g., using "advanced search"). The participants were not told about the goal of this study (i.e., spelling correction), and the collected queries have no spelling errors. For generating spelling errors, we implemented a typo generator.[1] This generator extends an online typo generator (Seobook, 2010) that produces a variety of spelling errors, including skipped letter, doubled letter, reversed letter, skipped space (merge), missed key and inserted key; in addition, our generator produces inserted space (split). When applied to a search query, our generator adds random typos to each word, independently, with a specified probability p that is 50% by default. For each collected query (and for each considered value of p) we generated 5 misspelled queries, and thereby obtained 250 instances of misspelled general queries and 250 instances of misspelled sender queries.

Methodology. We compared the accuracy of MaxPaths (Section 2) with three alternatives. The first alternative is the open-source *Jazzy*, which is a widely used spelling-correction tool based on (weighted) edit distance. The second alternative is the spelling correction provided by *Google*. We provided Jazzy with our unigram index (as a dictionary). However, we were not able to do so with Google, as we used remote access via its Java API (Google, 2010); hence, the Google tool is unaware of our domain, but is rather based on its own statistics (from the World Wide Web). The third alternative is what we call WordWise, which applies word-level correction (Section 2.1) to each input query term, independently. More precisely, WordWise is a simplified version of MaxPaths, where we forbid splitting and merging of words (i.e., only the original tokens are considered), and where we do not take correlation into account.

Our emphasis is on *correcting misspelled queries*, rather than *recognizing correctly spelled queries*, due to the role of spelling in a search engine: we wish to provide the user with the correct query upon misspelling, but there is no harm in making a suggestion for correctly spelled queries, except for visual discomfort. Hence, by default *accuracy* means the number of properly corrected queries (within the top-k suggestions) divided by the number of the misspelled queries. An exception is in Section 3.5, where we study the accuracy on correct queries.

Since MaxPaths and WordWise involve parameter learning (SVM), the results for them are consistently obtained by performing 5-folder *cross validation* over each collection of misspelled queries.

3.2 Fixed Error Probability

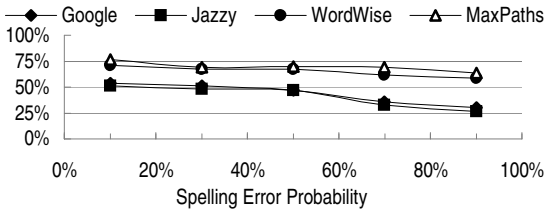
Here, we compare MaxPaths to the alternatives when the error probability p is fixed (0.5). We consider only the Kaminski dataset; the results for the other two datasets are similar. Figure 4(a) shows the accuracy, for general queries, of top-k suggestions for $k = 1$, $k = 3$ and $k = 10$. Note that we can get only one (top-1) suggestion from Google. As can be seen, MaxPaths has the highest accuracy in all cases. Moreover, the advantage of MaxPaths over the alternatives increases as k increases, which indicates potential for further improving MaxPaths.

Figure 4(b) shows the accuracy of top-k suggestions for sender queries. Overall, the results are similar to those of Figure 4(a), except that top-1 of both WordWise and MaxPaths has a higher accuracy in sender queries than in general queries. This is due to the fact that the dictionaries of person names and email addresses extracted from the `X-From` and `From` fields, respectively, provide strong features for the scoring function, since a sender query refers to these two fields. In addition, the accuracy of MaxPaths is further enhanced by exploiting the cor-

[1] The queries and our typo generator are publicly available at `https://dbappserv.cis.upenn.edu/spell/`.

(a) General queries (Kaminski)

(b) Sender queries (Kaminski)



(c) Varying error probability (Kaminski)

Figure 4: Accuracy for Kaminski (misspelled queries)

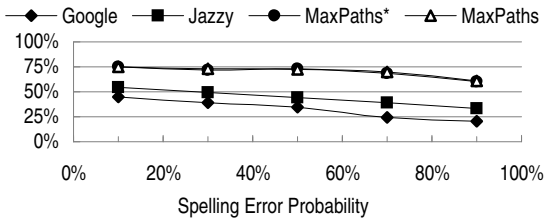
relation between the first and last name of a person.

3.3 Impact of Error Probability

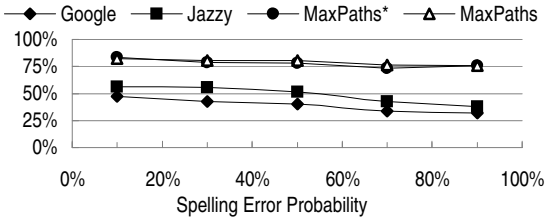
We now study the impact of the complexity of spelling errors on our algorithm. For that, we measure the accuracy while the error probability p varies from 10% to 90% (with gaps of 20%). The results are in Figure 4(c). Again, we show the results only for Kaminski, since we get similar results for the other two datasets. As expected, in all examined methods the accuracy decreases as p increases. Now, not only does MaxPaths outperform the alternatives, its decrease (as well as that of WordWise) is the mildest—13% as p increases from 10% to 90% (while Google and Jazzy decrease by 23% or more). We got similar results for the sender queries (and for each of the three users).

3.4 Adaptiveness of Parameters

Obtaining the labeled data needed for parameter learning entails a nontrivial manual effort. Ideally, we would like to learn the parameters of MaxPaths in one domain, and use them in similar domains.



(a) General queries (Farmer)



(b) Sender queries (Farmer)

Figure 5: Accuracy for Farmer (misspelled queries)

More specifically, our desire is to use the parameters learned over one corpus (e.g., the email collection of one user) on a second corpus (e.g., the email collection of another user), rather than learning the parameters again over the second corpus. In this set of experiments, we examine the feasibility of that approach. Specifically, we consider the user Farmer and observe the accuracy of our algorithm with two sets of parameters: the first, denoted by MaxPaths in Figures 5(a) and 5(b), is learned within the Farmer dataset, and the second, denoted by MaxPaths*, is learned within the Kaminski dataset. Figures 5(a) and 5(b) show the accuracy of the top-1 suggestion for general queries and sender queries, respectively, with varying error probabilities. As can be seen, these results mean good news—the accuracies of MaxPaths* and MaxPaths are extremely close (their curves are barely distinguishable, as in most cases the difference is smaller than 1%). We repeated this experiment for Kitchen and Kaminski, and got similar results.

3.5 Accuracy for Correct Queries

Next, we study the accuracy on *correct* queries, where the task is to recognize the given query as correct by returning it as the top suggestion. For each of the three users, we considered the $50 + 50$ (general + sender) collected queries (having no spelling errors), and measured the *accuracy*, which is the percentage of queries that are equal to the top sug-

Table 3: Accuracy for Correct Queries

Dataset	Google	Jazzy	MaxPaths
Kaminski (general)	90%	98%	94%
Kaminski (sender)	94%	98%	94%
Farmer (general)	96%	98%	96%
Farmer (sender)	96%	96%	92%
Kitchen (general)	86%	100%	92%
Kitchen (sender)	94%	100%	98%

gestion. Table 3 shows the results. Since Jazzy is based on edit distance, it almost always gives the input query as the top suggestion; the misses of Jazzy are for queries that contain a word that is not the corpus. MaxPaths is fairly close to the upper bound set by Jazzy. Google (having no access to the domain) also performs well, partly because it returns the input query if no reasonable suggestion is found.

3.6 Applicability to Large-Scale Site Search

Up to now, our focus has been on email search, which represents a restricted (closed) domain with specialized knowledge (e.g., sender names). In this part, we examine the effectiveness of our algorithm in a totally different setting—*large-scale site search* within `www.ibm.com`, a domain that is popular on a world scale. There, the accuracy of Google is very high, due to this domain's popularity, scale, and full accessibility on the Web. We crawled 10 million documents in that domain to obtain the corpus. We manually collected 1348 misspelled queries from the log of search issued against developerWorks (`www.ibm.com/developerworks/`) during a week. To facilitate the manual collection of these queries, we inspected each query with two or fewer search results, after applying a random permutation to those queries. Figure 6 shows the accuracy of top-k suggestions. Note that the performance of MaxPaths is very close to that of Google—only 2% lower for top-1. For $k = 3$ and $k = 10$, MaxPaths outperforms Jazzy and the top-1 of Google (from which we cannot obtain top-k for $k > 1$).

Figure 6: Accuracy for site search

3.7 Summary

To conclude, our experiments demonstrate various important qualities of MaxPaths. First, it outperforms its alternatives, in both accuracy (Section 3.2) and robustness to varying error complexities (Section 3.3). Second, the parameters learned in one domain (e.g., an email user) can be applied to similar domains (e.g., other email users) with essentially no loss in performance (Section 3.4). Third, it is highly accurate in recognition of correct queries (Section 3.5). Fourth, even when applied to large (open) domains, it achieves a comparable performance to the state-of-the-art Google spelling correction (Section 3.6). Finally, the higher performance of MaxPaths on top-3 and top-10 corrections suggests a potential for further improvement of top-1 (which is important since search engines often restrict their interfaces to only one suggestion).

4 Conclusions

We presented the algorithm MaxPaths for spelling correction in domain-centric search. This algorithm relies primarily on corpus statistics and domain knowledge (rather than on query logs). It can handle a variety of spelling errors, and can incorporate different levels of spelling reliability among different parts of the corpus. Our experimental study demonstrates the superiority of MaxPaths over existing alternatives in the domain of email search, and indicates its effectiveness beyond that domain.

In future work, we plan to explore how to utilize additional domain knowledge to better estimate the correlation between words. Particularly, from available auxiliary data (Fagin et al., 2010) and tools like *information extraction* (Chiticariu et al., 2010), we can infer and utilize *type* information from the corpus (Li et al., 2006b; Zhu et al., 2007). For instance, if `kohli` is of type person, and `phone` is highly correlated with person instances, then `phone` is highly correlated with `kohli` even if the two words do not frequently co-occur. We also plan to explore aspects of corpus maintenance in dynamic (constantly changing) domains.

References

F. Ahmad and G. Kondrak. 2005. Learning a spelling error model from search query logs. In *HLT/EMNLP*.

R. Bekkerman, A. Mccallum, and G. Huang. 2004. Automatic categorization of email into folders: Benchmark experiments on Enron and Sri Corpora. Technical report, University of Massachusetts - Amherst.

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

L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. 2010. SystemT: An algebraic approach to declarative information extraction. In *ACL*, pages 128–137.

C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

S. Cucerzan and E. Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of Web users. In *EMNLP*, pages 293–300.

D. Eppstein. 1994. Finding the k shortest paths. In *FOCS*, pages 154–165.

R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. 2010. Understanding queries in a search database system. In *PODS*, pages 273–284.

Google. 2010. A Java API for Google spelling check service. http://code.google.com/p/google-api-spelling-java/.

D. Jurafsky and J. H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR.

M. D. Kernighan, K. W. Church, and W. A. Gale. 1990. A spelling correction program based on a noisy channel model. In *COLING*, pages 205–210.

B. Klimt and Y. Yang. 2004. Introducing the Enron corpus. In *CEAS*.

K. Kukich. 1992. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439.

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

Y. Li, R. Krishnamurthy, S. Vaithyanathan, and H. V. Jagadish. 2006b. Getting work done on the web: supporting transactional queries. In *SIGIR*, pages 557–564.

R. Mitton. 2010. Fifty years of spellchecking. *Wring Systems Research*, 2:1–7.

J. L. Peterson. 1980. *Computer Programs for Spelling Correction: An Experiment in Program Design*, volume 96 of *Lecture Notes in Computer Science*. Springer.

J. Schaback and F. Li. 2007. Multi-level feature extraction for spelling correction. In *AND*, pages 79–86.

M. Schierle, S. Schulz, and M. Ackermann. 2007. From spelling correction to text cleaning - using context information. In *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 397–404.

Seobook. 2010. Keyword typo generator. http://tools.seobook.com/spelling/keywords-typos.cgi.

X. Sun, J. Gao, D. Micol, and C. Quirk. 2010. Learning phrase-based spelling error models from clickthrough data. In *ACL*, pages 266–274.

H. Zhu, S. Raghavan, S. Vaithyanathan, and A. Löser. 2007. Navigating the intranet with high precision. In *WWW*, pages 491–500.