Real-Word Spelling Correction using Google Web 1T 3-grams

Aminul Islam
Department of Computer Science
University of Ottawa
Ottawa, ON, K1N 6N5, Canada
mdislam@site.uottawa.ca

Diana Inkpen
Department of Computer Science
University of Ottawa
Ottawa, ON, K1N 6N5, Canada
diana@site.uottawa.ca

Abstract

We present a method for detecting and correcting multiple real-word spelling errors using the Google Web 1T 3-gram data set and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm. Our method is focused mainly on how to improve the detection recall (the fraction of errors correctly detected) and the correction recall (the fraction of errors correctly amended), while keeping the respective precisions (the fraction of detections or amendments that are correct) as high as possible. Evaluation results on a standard data set show that our method outperforms two other methods on the same task.

1 Introduction

Real-word spelling errors are words in a text that occur when a user mistakenly types a correctly spelled word when another was intended. Errors of this type may be caused by the writer's ignorance of the correct spelling of the intended word or by typing mistakes. Such errors generally go unnoticed by most spellcheckers as they deal with words in isolation, accepting them as correct if they are found in the dictionary, and flagging them as errors if they are not. This approach would be sufficient to detect the non-word error *myss* in "It doesn't know what the *myss* is all about." but not the real-word error *muss* in "It doesn't know what the *muss* is all about." To detect the latter, the spell-checker needs to make use of the surrounding context such as, in this case, to recognise that *fuss* is more likely to occur than *muss* in the context of *all about*. Ironically, errors of this type may even be caused by spelling checkers in the correction of non-word spelling errors when the *auto-correct* feature in some word-processing software sometimes silently change a non-word to the wrong real word (Hirst and Budanitsky, 2005), and sometimes when correcting a flagged error, the user accidentally make a wrong selection from the choices offered (Wilcox-O'Hearn et al., 2008).

An extensive review of real-word spelling correction is given in (Pedler, 2007; Hirst and Budanitsky, 2005) and the problem of spelling correction more generally is reviewed in (Kukich, 1992).

The Google Web 1T data set (Brants and Franz, 2006), contributed by Google Inc., contains English word *n*-grams (from unigrams to 5-grams) and their observed frequency counts calculated over 1 trillion words from web page text collected by Google in January 2006. The text was tokenised following the Penn Treebank tokenisation, except that hyphenated words, dates, email addresses and URLs are kept as single tokens. The sentence boundaries are marked with two special tokens <S> and </S>. Words that occurred fewer than 200 times were replaced with the special token <UNK>. Table 1 shows the data sizes of the Web 1T corpus. The n-grams themselves must appear at least 40 times to be included in the Web 1T corpus[1]. It is expected that this data will be useful for statistical language modeling, e.g.,

Table 1: Google Web 1T Data Sizes

Number of	Number	Size on disk (in KB)
Tokens	1,024,908,267,229	N/A
Sentences	95,119,665,584	N/A
Unigrams	13,588,391	185,569
Bigrams	314,843,401	5,213,440
Trigrams	977,069,902	19,978,540
4-grams	1,313,818,354	32,040,884
5-grams	1,176,470,663	33,678,504

[1]Details of the Google Web 1T data set can be found at www.ldc.upenn.edu/Catalog/docs/LDC2006T13/readme.txt

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1241–1249,
Singapore, 6-7 August 2009.

for machine translation or speech recognition, as well as for other uses.

In this paper, we present a method for detecting and correcting multiple real-word spelling errors using the Google Web 1T 3-gram data set, and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm (details are in section 3.1). By *multiple* errors, we mean that if we have n words in the input sentence, then we try to detect and correct at most n-1 errors. We do not try to detect and correct an error, if any, in the first word as it is not computationally feasible to search in the Google Web 1T 3-grams while keeping the first word in the 3-gram as a variable. Our intention is to focus on how to improve the detection recall (the fraction of errors correctly detected) or correction recall (the fraction of errors correctly amended) while maintaining the respective precisions (the fraction of detections or amendments that are correct) as high as possible. The reason behind this intention is that if the recall for any method is around 0.5, this means that the method fails to detect or correct around 50 percent of the errors. As a result, we can not completely rely on these type of methods, for that we need some type of human interventions or suggestions to detect or correct the rest of the undetected or uncorrected errors. Thus, if we have a method that can detect or correct almost 80 percent of the errors, even generating some extra candidates that are incorrect is more helpful to the human.

This paper is organized as follow: Section 2 presents a brief overview of the related work. Our proposed method is described in Section 3. Evaluation and experimental results are discussed in Section 4. We conclude in Section 5.

2 Related Work

Work on real-word spelling correction can roughly be classified into two basic categories: methods based on semantic information or human-made lexical resources, and methods based on machine learning or probability information. Our proposed method falls into the latter category.

2.1 Methods Based on Semantic Information

The 'semantic information' approach first proposed by Hirst and St-Onge (1998) and later developed by Hirst and Budanitsky (2005) detected semantic anomalies, but was not restricted to checking words from predefined confusion sets. This approach was based on the observation that the words that a writer intends are generally semantically related to their surrounding words, whereas some types of real-word spelling errors are not, such as (using Hirst and Budanitsky's example), "It is my sincere hole (hope) that you will recover swiftly." Such "malapropisms" cause "a perturbation of the cohesion (and coherence) of a text." Hirst and Budanitsky (2005) use semantic distance measures in WordNet (Miller et al., 1993) to detect words that are potentially anomalous in context - that is, semantically distant from nearby words; if a variation in spelling results in a word that was semantically closer to the context, it is hypothesized that the original word is an error (a "malapropism") and the closer word is its correction.

2.2 Methods Based on Machine Learning

Machine learning methods are regarded as lexical disambiguation tasks and confusion sets are used to model the ambiguity between words. Normally, the machine learning and statistical approaches rely on pre-defined confusion sets, which are sets (usually pairs) of commonly confounded words, such as {*their*, *there*, *they're*} and {*principle*, *principal*}. The methods learn the characteristics of typical context for each member of the set and detect situations in which one member occurs in context that is more typical of another. Such methods, therefore, are inherently limited to a set of common, predefined errors, but such errors can include both content and function words. Given an occurrence of one of its confusion set members, the spellchecker's job is to predict which member of that confusion set is the most appropriate in the context. Golding and Roth (1999), an example of a machine-learning method, combined the Winnow algorithm with weighted-majority voting, using nearby and adjacent words as features. Another example of a machine-learning method is that of Carlson et al. (2001).

2.3 Methods Based on Probability Information

Mays et al. (1991) proposed a statistical method using word-trigram probabilities for detecting and correcting real-word errors without requiring predefined confusion sets. In this method, if the trigram-derived probability of an observed sentence is lower than that of a sentence obtained by replacing one of the words with a spelling varia-

tion, then we hypothesize that the original is an error and the variation is what the user intended.

Wilcox-O'Hearn et al. (2008) analyze the advantages and limitations of Mays et al. (1991)'s method, and present a new evaluation of the algorithm, designed so that the results can be compared with those of other methods, and then construct and evaluate some variations of the algorithm that use fixed-length windows. They consider a variation of the method that optimizes over relatively short, fixed-length windows instead of over a whole sentence (except in the special case when the sentence is smaller than the window), while respecting sentence boundaries as natural breakpoints. To check the spelling of a span of d words requires a window of length d+4 to accommodate all the trigrams that overlap with the words in the span. The smallest possible window is therefore 5 words long, which uses 3 trigrams to optimize only its middle word. They assume that the sentence is bracketed by two BoS and two EoS markers (to accommodate trigrams involving the first two and last two words of the sentence). The window starts with its left-hand edge at the first BoS marker, and the Mays et al. (1991)'s method is run on the words covered by the trigrams that it contains; the window then moves d words to the right and the process repeats until all the words in the sentence have been checked. As Mays et al. (1991)'s algorithm is run separately in each window, potentially changing a word in each, Wilcox-O'Hearn et al. (2008)'s method as a side-effect also permits multiple corrections in a single sentence.

Wilcox-O'Hearn et al. (2008) show that the trigram-based real-word spelling-correction method of Mays et al. (1991) is superior in performance to the WordNet-based method of Hirst and Budanitsky (2005), even on content words ("malapropisms"), especially when supplied with a realistically large trigram model. Wilcox-O'Hearn et al. (2008) state that their attempts to improve the method with smaller windows and with multiple corrections per sentence were not successful, because of excessive false positives.

Verberne (2002) proposed a trigram-based method for real-word errors without explicitly using probabilities or even localizing the possible error to a specific word. This method simply assumes that any word trigram in the text that is attested in the British National Corpus (Burnard, 2000) is correct, and any unattested trigram is a likely error. When an unattested trigram is observed, the method then tries the spelling variations of all words in the trigram to find attested trigrams to present to the user as possible corrections. The evaluation of this method was carried out on only 7100 words of the Wall Street Journal corpus, with 31 errors introduced (i.e., one error in every approximately 200 words) obtaining a recall of 0.33 for correction, a precision of 0.05 and a F-measure of 0.086.

3 Proposed Method

The proposed method first tries to determine some probable candidates and then finds the best one among the candidates or sorts them based on some weights. We consider a string similarity function and a normalized frequency value function in our method. The following sections present a detailed description of each of these functions followed by the procedure to determine some probable candidates along with the procedure to sort the candidates.

3.1 Similarity between Two Strings

We use the longest common subsequence (LCS) (Allison and Dix, 1986) measure with some normalization and small modifications for our string similarity measure. We use the same three different modified versions of LCS that we (Islam and Inkpen, 2008) used, along with another modified version of LCS, and then take a weighted sum of these[2]. Kondrak (2005) showed that edit distance and the length of the longest common subsequence are special cases of n-gram distance and similarity, respectively. Melamed (1999) normalized LCS by dividing the length of the longest common subsequence by the length of the longer string and called it longest common subsequence ratio (LCSR). But LCSR does not take into account the length of the shorter string which sometimes has a significant impact on the similarity score.

Islam and Inkpen (2008) normalized the *longest common subsequence* so that it takes into account the length of both the shorter and the longer string and called it *normalized longest common subse-*

[2]We (Islam and Inkpen, 2008) use modified versions because in our experiments we obtained better results (precision and recall) for schema matching on a sample of data than when using the original LCS, or other string similarity measures.

quence (NLCS) which is:

$$v_1 = NLCS(s_i, s_j) = \frac{len(LCS(s_i, s_j))^2}{len(s_i) \times len(s_j)} \quad (1)$$

While in classical LCS, the common subsequence needs not be consecutive, in spelling correction, a consecutive common subsequence is important for a high degree of matching. We (Islam and Inkpen, 2008) used *maximal consecutive longest common subsequence* starting at character 1, $MCLCS_1$ and *maximal consecutive longest common subsequence* starting at any character n, $MCLCS_n$. $MCLCS_1$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching must be from first character (character 1) for both strings. $MCLCS_n$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching may start from any character (character n) for both of the strings. We normalized $MCLCS_1$ and $MCLCS_n$ and called it *normalized* $MCLCS_1$ ($NMCLCS_1$) and *normalized* $MCLCS_n$ ($NMCLCS_n$), respectively.

$$v_2 = NMCLCS_1(s_i, s_j) = \frac{len(MCLCS_1(s_i, s_j))^2}{len(s_i) \times len(s_j)} \quad (2)$$

$$v_3 = NMCLCS_n(s_i, s_j) = \frac{len(MCLCS_n(s_i, s_j))^2}{len(s_i) \times len(s_j)} \quad (3)$$

Islam and Inkpen (2008) did not consider consecutive common subsequences ending at the last character, though $MCLCS_n$ sometimes covers this, but not always. We argue that the consecutive common subsequence ending at the last character is as significant as the consecutive common subsequence starting at the first character. So, we introduce the *maximal consecutive longest common subsequence* ending at the last character, $MCLCS_z$ (Algorithm 1). Algorithm 1, takes two strings as input and returns the shorter string or the maximal consecutive portions of the shorter string that consecutively matches with the longer string, where matching must end at the last character for both strings. We normalize $MCLCS_z$ and call it *normalized* $MCLCS_z$ ($NMCLCS_z$).

$$v_4 = NMCLCS_z(s_i, s_j) = \frac{len(MCLCS_z(s_i, s_j))^2}{len(s_i) \times len(s_j)} \quad (4)$$

We take the weighted sum of these individual values v_1, v_2, v_3, and v_4 to determine string similarity score, where α_1, α_2, α_3, α_4 are weights and $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. Therefore, the similarity of the two strings, $S \in [0, 1]$ is:

$$S(s_i, s_j) = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 \quad (5)$$

We heuristically set equal weights for our experiments[3]. Theoretically, $v_3 \geq v_2$ and $v_3 \geq v_4$. To give an example, consider $s_i = albastru$ and $s_j = alabasteru$, then

$LCS(s_i, s_j) = albastru$
$MCLCS_1(s_i, s_j) = al$
$MCLCS_n(s_i, s_j) = bast$
$MCLCS_z(s_i, s_j) = ru$
$NLCS(s_i, s_j) = 8^2/(8 \times 10) = 0.8$
$NMCLCS_1(s_i, s_j) = 2^2/(8 \times 10) = 0.05$
$NMCLCS_n(s_i, s_j) = 4^2/(8 \times 10) = 0.2$
$NMCLCS_z(s_i, s_j) = 2^2/(8 \times 10) = 0.05$

The string similarity, $S = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 = 0.25 \times 0.8 + 0.25 \times 0.05 + 0.25 \times 0.2 + 0.25 \times 0.05 = 0.275$

3.2 Normalized Frequency Value

We determine the normalized frequency value of each candidate word for a single position with respect to all other candidates for the same position. If we find n replacements of a word w_i which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$, and their frequencies $\{f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in}\}$, where f_{ij} is the frequency of a 3-gram (where any candidate word w_{ij} is a member of the 3-gram), then we determine the normalized frequency value of any candidate word w_{ij}, represented as $F(w_{ij}) \in (0, 1]$, as the frequency of the 3-gram having w_{ij} over the maximum frequency among all the candidate words for that position:

$$F(w_{ij}) = \frac{f_{ij}}{\max(f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in})} \quad (6)$$

3.3 Determining Candidate Words

Our task is to correct real-word spelling error from an input text using Google Web 1T 3-gram data set. Let us consider an input text W which

[3]We use equal weights in several places in this paper in order to keep the system unsupervised. If development data would be available, we could adjust the weights.

Algorithm 1: $MCLCS_z$ (Maximal Consecutive LCS ending at the last character)

input : s_i, s_j /* s_i and s_j are input strings where $|s_i| \leq |s_j|$ */
output: str /* str is the Maximal Consecutive LCS ending at the last character */

```
str ← NULL
c ← 1
while |s_i| ≥ c do
    x ← SubStr(s_i, −c, 1)    /* returns cth character of s_i from the end */
    y ← SubStr(s_j, −c, 1)    /* returns cth character of s_j from the end */
    if x = y then
        str ← SubStr(s_i, −c, c)
    else
        return str
    end
    increment c
end
```

after tokenization[4] has m words, i.e., $W = \{w_1, w_2, \ldots, w_m\}$. Our method aims to correct m-1 spelling errors, for all m-1 word positions, except for the first word position, as we do not try to correct the first word. We use a slight different way to correct the first word (i.e., w_2) and the last word (i.e., w_m) among those m-1 words, than for the rest of the words. First, we discuss how we find the candidates for a word (say w_i, where $2<i<m$) which is not either w_2 or w_m. Then, we discuss the procedure to find the candidates for either w_2 or w_m. Our method could have worked for the first word too. We did not do it here due to efficiency reasons. Google 3-grams are sorted based on the first word, then the second word, and so on. Based on this sorting, all Google 3-grams are stored in 97 different files. All the 97 Google 3-gram files could have been needed to access a single word, instead of accessing just one 3-gram file as we do for any other words. This is because when the first word needs to be corrected, it might be in any file among those 97 Google 3-gram files. No error appears in the first position among 1402 inserted malapropisms. The errors start appearing from the second position till the last position.

[4]We need to tokenize the input sentence to make the 3-grams formed using the tokens returned after the tokenization consistent with the Google 3-grams. The input sentence is tokenized in a manner similar to the tokenization of the Wall Street Journal portion of the Penn Treebank. Notable exceptions include the following:

- Hyphenated word are usually separated, and hyphenated numbers usually form one token.
- Sequences of numbers separated by slashes (e.g., in dates) form one token.
- Sequences that look like urls or email addresses form one token.

3.3.1 Determining Candidate Words for w_i $(2 < i < m)$

We use the following steps:

1. We define the term *cut off frequency* for word w_i or word w_{i+1} as the frequency of the 3-gram $w_{i-1}\ w_i\ w_{i+1}$ in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0. The intuition behind using the *cut off frequency* is the fact that, if the word is misspelled, then the correct one should have a higher frequency than the misspelled one. Thus, using the *cut off frequency*, we isolate a large number of candidates that we do not need to process.

2. We find all the 3-grams (where only w_i is changed while w_{i-1} and w_{i+1} are unchanged) having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 3-gram $w_{i-1}\ w_{ij}\ w_{i+1}$.

3. We determine the *cut off frequency* for word w_{i-1} or word w_i as the frequency of the 3-gram $w_{i-2}\ w_{i-1}\ w_i$ in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

4. We find all the 3-grams (where only w_i is changed while w_{i-2} and w_{i-1} are unchanged) having frequency greater than the *cut off frequency* of w_i (determined in step 3). Let us consider that we find n replacements of w_i which are $R_2 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies

$F_2 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 3-gram $w_{i-2}\ w_{i-1}\ w_{ij}$.

5. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5) and then assign a weight using the following equation (7) only to the words that return the string similarity value greater than 0.5.

$$weight = \beta S(w_i, w_{ij}) + (1-\beta)F(w_{ij}) \quad (7)$$

Equation (7) is used to ensure a balanced weight between the string similarity function and the normalized frequency value function where β refers to how much importance we give to the string similarity function with respect to the normalized frequency value function[5].

6. For each $w_{ij} \in R_2$, we calculate the string similarity between w_{ij} and w_i using equation (5), and then assign a weight using the equation (7) only to the words that return the string similarity value greater than 0.5.

7. We sort the words found in step 5 and in step 6 that were given weights, if any, in descending order by the assigned weights and keep only one word as candidate word[6].

3.3.2 Determining Candidate Words for w_2

We use the following steps:

1. We determine the *cut off frequency* for word w_2 as the frequency of the 3-gram $w_1\ w_2\ w_3$ in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_2 as 0.

2. We find all the 3-grams (where only w_2 is changed while w_1 and w_3 are unchanged) having frequency greater than the *cut off frequency* of w_2 (determined in step 1). Let us consider that we find n replacements of w_2 which are $R_1 = \{w_{21}, w_{22}, \cdots, w_{2n}\}$, and their frequencies $F_1 = \{f_{21}, f_{22}, \cdots, f_{2n}\}$, where f_{2j} is the frequency of the 3-gram $w_1\ w_{2j}\ w_3$.

3. For each $w_{2j} \in R_1$, we calculate the string similarity between w_{2j} and w_2 using equation (5), and then assign a weight using the following equation only to the words that return the string similarity value greater than 0.5.

$$weight = \beta S(w_2, w_{2j}) + (1-\beta)F(w_{2j})$$

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only one word as candidate word.

3.3.3 Determining Candidate Words for w_m

We use the following steps:

1. We determine the *cut off frequency* for word w_m as the frequency of the 3-gram $w_{m-2}\ w_{m-1}\ w_m$ in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_m as 0.

2. We find all the 3-grams (where only w_m is changed while w_{m-2} and w_{m-1} are unchanged) having frequency greater than the *cut off frequency* of w_m (determined in step 1). Let us consider that we find n replacements of w_m which are $R_2 = \{w_{m1}, w_{m2}, \cdots, w_{mn}\}$ and their frequencies $F_2 = \{f_{m1}, f_{m2}, \cdots, f_{mn}\}$, where f_{mj} is the frequency of the 3-gram $w_{m-2}\ w_{m-1}\ w_{mj}$.

3. For each $w_{mj} \in R_2$, we calculate the string similarity between w_{mj} and w_m using equation (5) and then assign a weight using the following equation only to the words that return the string similarity value greater than 0.5.

$$weight = \beta S(w_m, w_{mj}) + (1-\beta)F(w_{mj})$$

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only one word as the candidate word.

[5] We give more importance to string similarity function with respect to frequency value function throughout the section of 'determining candidate words' to have more candidate words so that the chance of including the target word into the set of candidate words gets higher. For this reason, we heuristically set β=0.85 in equation (7) instead of setting β=0.5.

[6] Sometimes the top candidate word might be either a plural form or a past participle form of the original word. Or even it might be a high frequency function word (e.g., *the*). We omit these type of words from the candidacy.

4 Evaluation and Experimental Results

We used as test data the same data that Wilcox-O'Hearn et al. (2008) used in their evaluation of Mays et al. (1991) method, which in turn was a replication of the data used by Hirst and St-Onge (1998) and Hirst and Budanitsky (2005) to evaluate their methods.

The data consisted of 500 articles (approximately 300,000 words) from the 1987−89 *Wall Street Journal* corpus, with all headings, identifiers, and so on removed; that is, just a long stream of text. It is assumed that this data contains no errors; that is, the *Wall Street Journal* contains no malapropisms or other typos. In fact, a few typos (both non-word and real-word) were noticed during the evaluation, but they were small in number compared to the size of the text.

Malapropisms were randomly induced into this text at a frequency of approximately one word in 200. Specifically, any word whose base form was listed as a noun in WordNet (but regardless of whether it was used as a noun in the text; there was no syntactic analysis) was potentially replaced by any spelling variation found in the lexicon of the *ispell* spelling checker[7]. A *spelling variation* was defined as any word with an *edit distance* of 1 from the original word; that is, any single-character insertion, deletion, or substitution, or the transposition of two characters, that results in another real word. Thus, none of the induced malapropisms were derived from closed-class words, and none were formed by the insertion or deletion of an apostrophe or by splitting a word. The data contained 1402 inserted malapropisms.

Because it had earlier been used for evaluating Mays et al. (1991)'s trigram method, which operates at the sentence level, the data set had been divided into three parts, without regard for article boundaries or text coherence: sentences into which no malapropism had been induced; the original versions of the sentences that received malapropisms; and the malapropized sentences. In addition, all instances of numbers of various kinds had been replaced by tags such as <INTEGER>, <DOLLAR VALUE>, and <PERCENTAGE VALUE>. Actual (random) numbers or values were restored for these tags. Some spacing anomalies around punctuation marks were corrected. A detailed description of this data can be found in (Hirst, 2008; Wilcox-O'Hearn et al., 2008).

[7]Ispell is a fast screen-oriented spelling checker that shows you your errors in the context of the original file, and suggests possible corrections when it can figure them out. The original was written in PDP-10 assembly in 1971, by R. E. Gorin. The C version was written by Pace Willisson of MIT. Geoff Kuenning added the international support and created the current release.

SUCCESSFUL CORRECTION:
The Iran revelations were particularly disturbing to the Europeans because they came on the heels of the Reykjavik summit between President Reagan and Soviet *reader* → leader [leader] Mikhail Gorbachev.
Even the now sainted Abraham Lincoln was often reviled while in *officer* → office [office], sometimes painted by cartoonists and editorial writers as that baboon in the White House.
FALSE POSITIVE:
· · · by such public displays of interest in *Latinos* → Latin [Latinos], many undocumented · · ·
The southeast Asian nation was one reported *contributor* → contribution [contributor] to the Nicaraguans.
FALSE NEGATIVE:
Kevin Mack, Geldermann president and chief executive officer, didn't return calls for comment on the Clayton *purchaser* [purchase].
U.S. *manufactures* [manufacturers], in short, again are confronting a ball game in which they will be able to play.
TRUE POSITIVE DETECTION, FALSE POSITIVE CORRECTION:
Hawkeye also is known to *rear* → reader [fear] that a bankruptcy-law filing by the parent company, which theoretically shouldn't affect the operations of its member banks, would spark runs on the banks that could drag down the whole entity.
The London Daily News has quoted sources saying as many as 23 British mercenaries were enlisted by KMS to *lid* → slide [aid] the Contras.

Table 2: Examples of successful and unsuccessful corrections. Italics indicate observed word, arrow indicates correction, square brackets indicate intended word.

Some examples of successful and unsuccessful corrections using our proposed method are shown in Table 2.

Table 3 shows our method's results on the described data set compared with the results for the trigram method of Wilcox-O'Hearn et al. (2008)

Detection			correction		
R	P	F_1	R	P	F_1
Lexical cohesion (Hirst and Budanitsky, 2005)					
0.306	0.225	0.260	0.281	0.207	0.238
Trigrams (Wilcox-O'Hearn et al., 2008)					
0.544	0.528	0.536	0.491	0.503	0.497
Multiple 3-grams					
0.890	0.445	0.593	0.763	0.381	0.508

Table 3: A comparison of recall, precision, and F_1 score for three methods of malapropism detection and correction on the same data set.

and the lexical cohesion method of Hirst and Budanitsky (2005). The data shown here for trigram method are not from (Wilcox-O'Hearn et al., 2008), but rather are later results following some corrections reported in (Hirst, 2008). We have not tried optimizing our adjustable parameters: β and αs, because the whole data set was used as testing set by the other methods we compare with. To keep the comparison consistent, we did not use any portion of the data set for training purpose. Having optimized parameters could lead to a better result. The performance is measured using *Recall* (R), *Precision* (P) and F_1:

$$R = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$P = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$F_1 = \frac{2PR}{P + R}$$

The fraction of errors correctly detected is the detection *recall* and the fraction of detections that are correct is the detection *precision*. Again, the fraction of errors correctly amended is the correction *recall* and the fraction of amendments that are correct is the correction *precision*. To give an example, consider a sentence from the data set: "The Philippine president, in her commencement address at the academy, complained that the U.S. was *living* → giving [giving] advice instead of the *aid* → said [aid] it pledged.", where italics indicate the observed word, arrow indicates the correction and the square brackets indicate the intended word. The detection *recall* of this sentence is 1.0 and the *precision* is 0.5. The correction *recall* of this sentence is 1.0 and the *precision* is 0.5. For both cases, the F_1 score is 0.667.

We loose some precision because our method tries to detect and correct errors for all the words (except the first word) in the input sentence, and, as a result, it generates more false positives than the other methods. Even so, we get better F_1 scores than the other competing methods. Accepting 8.3 percents extra incorrect detections, we get 34.6 percents extra correct detections of errors, and similarly, accepting 12.2 percents extra incorrect amendments, we get 27.2 percents extra correct amendments of errors compared with the trigrams method (Wilcox-O'Hearn et al., 2008)[8].

5 Conclusion

The Google 3-grams proved to be very useful in detecting real-word errors, and finding the corrections. We did not use the 4-grams and 5-grams because of data sparsity. When we tried with 5-grams the results were lower than the ones presented in Section 4. Having sacrificed a bit the precision score, our proposed method achieves a very good detection recall (0.89) and correction recall (0.76). Our attempts to improve the detection recall or correction recall, while maintaining the respective precisions as high as possible are helpful to the human correctors who post-edit the output of the real-word spell checker. If there is no postediting, at least more errors get corrected automatically. Our method could also detect and correct misspelled words, not only malapropisms, without any modification. In future work, we plan to extend our method to allow for deleted or inserted words, and to find the corrected strings in the Google Web 1T n-grams. In this way we will be able to correct grammar errors too. We also plan more experiments using the 5-grams, but backing off to 4-grams and 3-grams when needed.

Acknowledgments

This work is funded by the Natural Sciences and Engineering Research Council of Canada. We want to thank Professor Graeme Hirst from the Department of Computer Science, University of Toronto, for providing the evaluation data set.

[8]We can run our algorithm on subsets of data to check for variance in the results. We cannot test statistical significance compared to the related work (t-test), because we do not have the system from related work to run it on subsets of the data.

References

L. Allison and T.I. Dix. 1986. A bit-string longest-common-subsequence algorithm. *Information Processing Letters*, 23:305–310.

Thorsten Brants and Alex Franz. 2006. Web 1T 5-gram corpus version 1.1. Technical report, Google Research.

Lou Burnard, 2000. *Reference Guide for the British National Corpus (World Edition)*, October. www.natcorp.ox.ac.uk/docs/userManual/urg.pdf.

Andrew J. Carlson, Jeffrey Rosen, and Dan Roth. 2001. Scaling up context-sensitive text correction. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pages 45–50. AAAI Press.

Andrew R. Golding and Dan Roth. 1999. A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130.

Graeme Hirst and Alexander Budanitsky. 2005. Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(1):87–111, March.

Graeme Hirst and David St-Onge, 1998. *WordNet: An electronic lexical database*, chapter Lexical chains as representations of context for the detection and correction of malapropisms, pages 305–332. The MIT Press, Cambridge, MA.

Graeme Hirst. 2008. An evaluation of the contextual spelling checker of microsoft office word 2007, January. http://ftp.cs.toronto.edu/pub/gh/Hirst-2008-Word.pdf.

Aminul Islam and Diana Inkpen. 2008. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data*, 2(2):1–25.

G. Kondrak. 2005. N-gram similarity and distance. In *Proceedings of the 12h International Conference on String Processing and Information Retrieval*, pages 115–126, Buenos Aires, Argentina.

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

Eric Mays, Fred J. Damerau, and Robert L. Mercer. 1991. Context based spelling correction. *Information Processing and Management*, 27(5):517–522.

I. D. Melamed. 1999. Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1):107–130.

G.A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K.J. Miller. 1993. Introduction to wordnet: An on-line lexical database. Technical Report 43, Cognitive Science Laboratory, Princeton University, Princeton, NJ.

Jennifer Pedler. 2007. *Computer Correction of Real-word Spelling Errors in Dyslexic Text*. Ph.D. thesis, Birkbeck, London University.

Suzan Verberne. 2002. Context-sensitive spell checking based on word trigram probabilities. Master's thesis, University of Nijmegen, February-August.

L. Amber Wilcox-O'Hearn, Graeme Hirst, and Alexander Budanitsky. 2008. Real-word spelling correction with trigrams: A reconsideration of the mays, damerau, and mercer model. In Alexander Gelbukh, editor, *Proceedings, 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2008) (Lecture Notes in Computer Science 4919, Springer-Verlag)*, pages 605–616, Haifa, February.