

# An In-depth Study of the Automatic Detection and Correction of Spelling Mistakes

Sau Kwan Chan  
University of Glasgow  
17 Lilybank Gardens  
Glasgow

chansk@dcs.gla.ac.uk

Ben He  
University of Glasgow  
17 Lilybank Gardens  
Glasgow

ben@dcs.gla.ac.uk

Iadh Ounis  
University of Glasgow  
17 Lilybank Gardens  
Glasgow

ounis@dcs.gla.ac.uk

## ABSTRACT

This paper discusses the issues involved in an information retrieval system when spelling errors are encountered in a query. We look at two classical algorithms that may be used to correct these errors and their consequent effect on the system's performance. The algorithms are the Levenshtein Distance and the Longest Common Subsequence. We experiment on a variety of test data and explore the impact of certain errors on an information retrieval system. We produce an in-depth study of the use of the two algorithms in an information retrieval environment. We evaluate the algorithms using two measures. Using two different data collections, the WT2G and .GOV collections, of differing size, we observe the effects of different types of spelling errors and their successful error correction rate. In particular, we find that the precision/recall results for query terms with spelling errors, regardless of the collection and its size, are reduced by up to half when compared to correctly spelled queries, showing that there is a justifiable need for error correction.

## 1. INTRODUCTION

Information retrieval (IR) systems are ultimately designed and developed to be used by end users. The task of formulating a query to use with the system is not necessarily a simple task and, among other things, users are prone to making errors with the spelling of a query for various reasons. A user may be in a rush and can not afford the time to check the spelling of a query, the user may not be searching with their native language or the user simply does not know the exact spelling of the term. We define a spelling error to be a query term that is not spelt in the same way as the terms in the lexicon of the system. For example, 'retreival' is a spelling mistake of the term 'retrieval'.

It is well known that queries containing spelling errors are common place. The work done by Kukich supports this fact [11]. Kukich showed that there exists approximately 0.2% -

3% spelling errors in English texts. This is regardless of the size of the corpora and will usually cause the IR system's recall rate to decrease.

While it is logical to correct spelling errors, there does not exist a comprehensive study into the effects of using an error correction algorithm in an IR environment. It can be questioned as to whether using string analysis algorithms to correct spelling errors is justifiable in terms of successful results and the costs involved with respect to an IR system. This paper aims to provide an in-depth study on the use of two classical spelling correction algorithms and to ascertain if the benefits do or do not outweigh the costs, regardless of the collection size or content.

The vast differences in collection size brings forward the design issue of whether to correct the errors in the documents of a collection, to correct the given query, or to correct both the documents and the queries. Similar to cross-language retrieval, due to the ease with which a collection can grow in size, the most logical method is to detect and correct spelling errors made in the queries rather than in the collection. We take advantage of the fact that queries tend to consist of a few words, so our algorithm needs only to check these terms making the querying process faster.

While other error correction techniques exist [1, 11], we concentrate on two classical and well established string matching algorithms to detect misspelled terms, namely the Edit Distance and the Longest Common Subsequence algorithms. The results show that both algorithms can correct misspelled terms but the degree of success varied. Our two methods of evaluation provide evidence supporting this. The Edit Distance was found to be more effective than the Longest Common Subsequence with most of the tested type of errors. Our first method of evaluation details the limits of each algorithm in regards to the number of spelling mistakes that can be made before a severe degradation in performance is observed. This is the rate of successful error corrections. We use precision/recall measures as the second evaluation measure. Our analysis shows that using either of these algorithms prevented precision/recall measures to fall to the values of those queries without error correction supporting the need for these algorithms to be used.

In the next two sections, we discuss various types of spelling errors and two classical algorithms that can be used to correct these errors. We look at how they differ in their ap-

proach to correcting erroneous terms. In Section 4, we set out the environment in which our experiments are performed and explain the two evaluation methods used. In addition, we consider how we can keep processor performance high and memory usage low. We evaluate our algorithms in Section 5 and show the results from our study, and discuss the reasons why we got the results we did. Finally, we draw the conclusions of the experiments, discussing to what extent our hypotheses were correct and give possible directions of future developments in Section 6.

## 2. SPELLING ERROR DETECTION AND CORRECTION

There are four different basic types of spelling errors according to Durham, Lam and Saxe [6]. These are:

1. There are extra characters in the term. For example, *'retrievall'* has an extra 'l'.
2. There are letters missing from the term. For example, *'retreval'* where the missing letter is 'i'.
3. Letters in the term are wrong. For example, *'retreeval'* has an 'e' where it should be 'i'.
4. The transposition of letters. For example, *'retrEival'* instead of *'retrIEval'*.

We use these four types as a basis for our study. It is clear that there is no limit to the number of errors made in one term. A term could contain as many or even more errors than correct characters. Hence, a correction algorithm should not be limited by the number of errors made nor by the length of the term.

In addition, a query can have multiple different types of errors within it. For example, a query with the term *'informatoin'* may contain a letter missing (Type 2) as well as a transposition of letters (Type 4) when it should have been *'information'*. Therefore, an IR system that has a spelling error detection and correction functionality should take this into account.

We aim to investigate the effects of having this functionality in regards to querying performance across different corpora and test which correction algorithm is better suited for which type of spelling error.

## 3. LEVENSHTein DISTANCE AND LONGEST COMMON SUBSEQUENCE ALGORITHMS

We use the lexicon of the system as our 'dictionary' of correctly spelled words. This is a logical approach since the system will not propose correction suggestions that are not in the index and so any corrected terms are guaranteed to retrieve some documents.

An error is detected when a query term does not exist in the lexicon. If no error is found then the system runs as normal but if an error is detected then we use one of two classical algorithms to search for possible corrections: the Levenshtein

Distance algorithm or the Longest Common Subsequence algorithm.

The Levenshtein Distance was a concept introduced by V. Levenshtein in 1965 to measure the distance between two strings [12]. This algorithm is more commonly known as the Edit Distance algorithm. This is because the algorithm measures the difference between two strings by finding the number of edits it takes to change one term to another. An edit is taken as a deletion of a character, substituting a character with another, or an addition of a character. So to change 'bread' to 'bean' would require two edits, a deletion of the character 'r' and a substitution of the character 'd' with the character 'n'. Therefore, the distance between these two strings is two.

Mathematician G. de B. Robinson first described an algorithm for finding the longest common subsequence (LCS) of two strings in 1938 [13]. The Longest Common Subsequence (LCS) algorithm finds the string sequence of largest length that exists in both of the two input strings. Say the two input strings were 'the sand over there' and 'cat sat on the mat', then the longest common subsequence would be 't sa o the' with length value ten, since the algorithm uses sequences regardless of the sequence's content. Whitespace is included in the computation.

Both these algorithms find possible corrections for misspelled query terms by looking at different properties of the input string. The Edit Distance measures the difference between two strings while the LCS algorithm measures what the two strings have in common. For the best possible correction to a misspelled term we desire the Edit Distance between the error term and the possible correction from the lexicon to be as small as possible. Whereas the larger the LCS between the terms in the lexicon and the misspelled term, the better the chance of the error being corrected with the intended term.

## 4. ENVIRONMENTAL SETTING

We use an existing search engine and choose what weighting model we use (see Section 4.1). In addition, we detail the two data collections (WT2G and .GOV), on which we performed our experiments (see Section 4.2). Finally, we discuss our evaluation methodology and the steps taken to ensure optimal processor performance and memory usage, an important consideration in developing IR systems (see Section 4.3).

### 4.1 PL2 Weighting Model

For the experiments we used the Terrier IR system<sup>1</sup>. Terrier uses various models for its matching functionalities. We used the PL2 matching function in the experiments which is a member of the Divergence From Randomness models [2]. These models were developed by Amati and van Rijsbergen based on their work on a probabilistic framework for IR. Using PL2, a document's relevance score  $d$  for a query  $Q$  is computed by the following formula:

<sup>1</sup>Terrier is developed by the University of Glasgow Information Retrieval Group. Further information can be found in <http://ir.dcs.gla.ac.uk/terrier>

$$\begin{aligned}
score(d, Q) &= \sum_{t \in Q} w(t, d) \\
&= \sum_{t \in Q} qtf \cdot \frac{1}{tfn + 1} (tfn \cdot \log_2 \frac{tfn}{\lambda} + (\lambda + \\
&\quad \frac{1}{12 \cdot tfn} - tfn) \cdot \log_2 e + 0.5 \cdot \log_2(2\pi \cdot tfn))
\end{aligned} \tag{1}$$

where  $\lambda$  is the mean and variance of a Poisson distribution,  $w(t, d)$  is the weight of document  $d$  for query term  $t$  and  $qtf$  is the number of occurrences of term  $t$  in the query.

The normalised term frequency  $tfn$  in equation (1) is computed using *normalisation 2*[2]:

$$tfn = tf \cdot \log_2(1 + c \cdot \frac{avgJ}{l}) \quad (c > 0) \tag{2}$$

where  $tf$  is the within document frequency of a given term,  $l$  is the document length and  $avgJ$  is the average document length in the whole collection. The value  $c$  is a free parameter and can be different for different collections. By measuring the normalisation effect,  $c$  is automatically estimated [9]. We use short queries and so the value of  $c$  is 10.57 and 1.25 for WT2G and .GOV, respectively.

## 4.2 TREC Data Collections

In order to have an idea on how the algorithms perform with any given corpora regardless of size, we experiment using two collections. One relatively small collection (WT2G) and one that is much larger (.GOV).

The WT2G collection is 2 gigabytes in size with 50 queries, i.e. TREC<sup>2</sup> topics 401 - 450 [7]. This collection contains documents of a miscellaneous nature. The larger collection is called .GOV and is 18 gigabytes [4, 5]. The .GOV collection is a sample of the ‘.gov’ domain. We used the collection along with TREC11-12 which has 99 queries in total. All TREC queries were checked beforehand and found to be correctly spelled.

We ran our algorithms on both collections so that we could observe the effects of spelling error detection and correction on both small and large collections, as well as on a range of different data.

## 4.3 Experimental Methodology

Our purpose in these experiments is to explore and survey the effects on querying performance with respect to two classical algorithms over different corpora. We evaluate the two algorithms by measuring their successful error correction rate and by using precision/recall measures.

As mentioned in Section 2, there are four basic types of spelling errors. As the TREC queries are correctly spelled, we can simulate a spelling error by randomly changing one of

<sup>2</sup>More information about the Text REtrieval Conference can be found on <http://trec.nist.gov>

the existing terms of each TREC query. Then we incrementally increase the number of errors to quantify the amount of errors each algorithm can handle before a visible degradation in performance. To illustrate, we change the TREC queries to have one extra letter (Type 1), then change the queries to have two extra letters and so on. This is repeated for each error type. This allows us to measure the effect each different error type has and whether one error type is more difficult to correct than another, as well as each algorithm’s cut off point in performance degradation regarding the number of errors.

We also measure the effects on querying performance using the traditional precision/recall values for each type of error. We use the original provided TREC queries and record the precision/recall values. This is our baseline, which we hope to match. On the other hand, we also record the precision/recall values for when there is no error correction for the modified query terms. By comparing these values with the precision/recall values obtained when the error correction algorithms are used, we can decide on the impact of using error correction algorithms on querying performance.

As with any system, we wish the running time of the system to be as fast as possible due to the large size of the corpus. It has been found that spelling errors have a frequency of at least a factor of 100 lower than correctly spelled terms [14]. Hence, we do not need to process terms in the lexicon with a lower term frequency than the misspelled query term since any term that has a lower frequency than a misspelled term is not likely to be correctly spelled itself. This cuts down the number of terms in the lexicon that need to be processed which in turn saves processing time.

To further reduce processing time, it is safe to assume that a user will enter correctly the first initial of a query term. This is the same assumption that the popular search engine Google uses. With this assumption, the system needs only to look at terms in the lexicon that have the same initial character. We created an index of the places in the lexicon where each initial character starts. This requires thirty six restricted binary searches but is run once on loading of the lexicon.

The amount of memory used by the system must also be kept to a minimum for practical purposes. We can not store all the terms in memory. Using the idea of dynamic programming by Hirschberg, we can use just two 1-dimensional arrays to store previous and current calculations instead of a  $m \times n$  matrix for the character comparisons [10]. This reduces the memory needed for the Edit Distance and LCS algorithms from  $O(mn)$  to  $O(n)$ . This is possible for both algorithms as we only need the last value of the matrix, which contains the distance or LCS length, for the purposes of this experiment.

## 5. EVALUATION

We evaluate the two algorithms by graphing each algorithm’s performance on the various types of errors (Types 1-4), to see the rate of successful error corrections and behaviour of the algorithms as the number of errors increase. In addition, by calculating the precision/recall values for each error type for both algorithms, we can compare and contrast the

two algorithms and promote one over the other in terms of querying performance improvement. We evaluate using precision/recall measures for the two used web collections to find out whether the effects of error correction are independent of the size or content of a collection.

## 5.1 Aims

In our experiments we measure how well each error correction algorithm performs. We compare them against each other and as stand alone algorithms. Based on how these algorithms work, we aim to explore and provide support for the following hypotheses:

1. Precision/Recall measures are higher using error detection and correction algorithms compared to systems without this functionality.
2. The two classical algorithms look at different properties of the query term and will not necessarily give the same results/performance. One algorithm may be better for specific types of errors.
3. Overall, the Edit Distance algorithm should perform better than the LCS algorithm, since the Edit Distance algorithm utilises 3 out of the 4 errors tested in our experiments.
4. The less errors there are in the query term, the better the chances of an accurate correction.
5. If an algorithm performs well on certain types of errors then the algorithm should perform well on a combination of these error types.
6. Error correction is independent of the collection's size and content.

## 5.2 WT2G Collection Results

Using the WT2G collection, we analyse the experimental results by measuring the rate of successful error corrections and using traditional precision/recall measures. We graph the percentage of accurate corrections while incrementally increasing the number of certain types of errors. We also use the precision/recall measures. With this range of evaluation methods, we can survey the effectiveness of using the Edit Distance and Longest Common Subsequence algorithms in errors correction in IR.

### 5.2.1 Errors Correction Graphs

For each error type, we graph the percentage of accurate corrections for each increment in the number of errors in the query term (Figure 1).

There is an overall downward trend to the percentage of accurate corrections as the number of mistakes increase, which marginally supports Hypothesis 4. In addition, it is easy to see from the graphs that the two classical algorithms do not give the same corrections all the time since they concentrate on different aspects of a term (Hypothesis 2) - if they did then surely the graphs would be the same for both Edit Distance and Longest Common Subsequence.

While overall, our graphs support Hypothesis 4, error correction rates for a term containing one error are at times

less than those for two errors. This is because our dictionary with which we check our query terms is the lexicon of the system. Since the lexicon has not been spellchecked beforehand, we may not detect the same misspelled query terms with one error (which is more common) if they already exist in the lexicon. So while a query term with more errors is harder to correct accurately, it also has a better chance of being detected as an error.

The Edit Distance performs well with error types 1, 2 and 3 (Figures 1(a), 1(b) and 1(c) respectively), with an error correction rate of over 60% for terms with one spelling error. Only the case when characters are swapped around (Type 4 with Figures 1(c)) is the correction rate below 50% for one error. This is expected since as mentioned in Section 3, it is exactly these three edits that are included in the algorithm's calculations. Indeed it performed better than the Longest Common Subsequence algorithm for queries with a single error, supporting Hypothesis 3.

However, in the case of deletion errors (Figure 1(b)) as the number of errors increased, the LCS algorithm managed to outperform the Edit Distance algorithm. Although the percentage did decrease for both algorithms, it was not as sharp a decrease as for the Edit Distance algorithm. In fact, in all four graphs the LCS algorithm appears to have a more stable behaviour even as the number of errors increases.

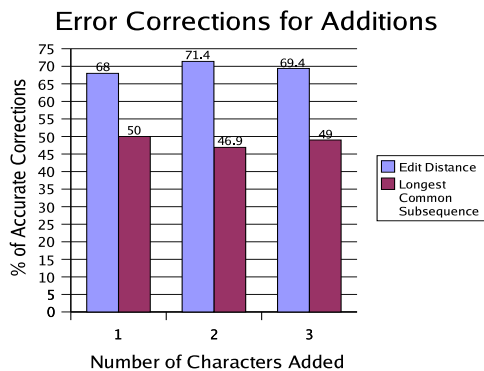
### 5.2.2 Precision/Recall Results

We calculate the precision/recall measures for the TREC queries which are correctly spelled. We compare the precision/recall values obtained using each spelling algorithm with the precision/recall values obtained from the original unaltered TREC queries and the precision/recall values when there is no error correction. Tables 1, 2, 3 and 4 detail these precision/recall values for each error type.

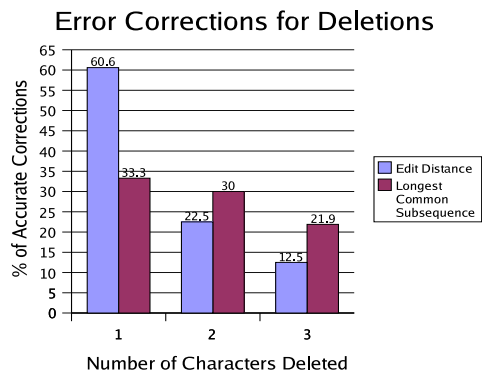
The average precision for queries that had used a spelling correction algorithm never fell below that of those queries that contained errors but were not corrected. Therefore, precision will on average increase with a spelling correction function. The two algorithms are able to cope with all the four common types of errors mentioned for this collection. This supports the first of our hypotheses made in this paper; that precision/recall measures will improve with error correction.

The Edit Distance algorithm performed particularly well for all error types apart from deletion errors (Type 2), outperforming the LCS algorithm. But it should be taken into consideration that while the algorithm performed poorly for increasing amounts of deletion errors, query terms are often short and so as more characters are missing, there are less remaining characters to work with.

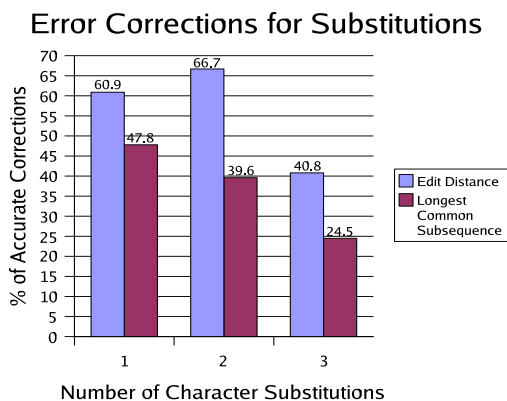
In the case for queries with errors of three extra characters (Table 1) using Edit Distance correction, the number of relevant documents retrieved is greater than those for the original queries with no errors by two documents. However there was a 0.0192 decrease in average precision compared to the baseline average precision. As the query terms now have three extra miscellaneous terms and since the query terms are short the mistake is more likely to be detected.



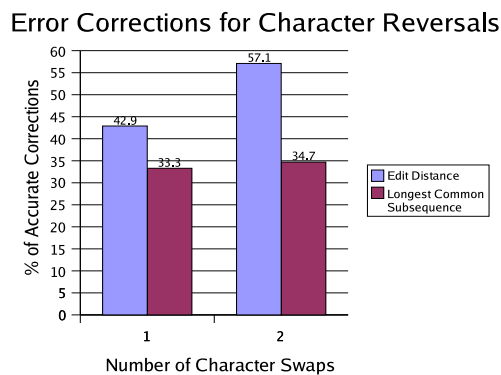
(a) Character additions (Type 1).



(b) Character deletions (Type 2).



(c) Character substitutions (Type 3).



(d) Character reversals (Type 4).

**Figure 1: Percentage of accurate corrections for various degrees of four error types (WT2G).**

Therefore, in this case it could simply be that more of the query terms are being picked up as errors.

The Edit Distance algorithm did consistently well considering that the average precision for correctly spelled queries was 0.2971. Using error correction average precision was in the region of 0.2 while without correction the average precision was no higher than 0.1548.

Like the Edit Distance algorithm, the LCS algorithm still managed to improve precision/recall values regardless of error type. The number of relevant documents also increased.

While the LCS algorithm did not perform as well as the Edit Distance, supporting Hypothesis 3, the LCS managed to outperform the latter with deletion errors as the number of deletion errors increased as mentioned earlier. Having two character deletions causes a sharper drop in average precision with the Edit Distance when compared with just one deletion, whereas the LCS algorithm has a more stable decrease (Table 2). Deletion errors do not change the sequence of correct characters as reversal errors, (Type 4), do and there are no noisy incorrect characters to take into account like those that addition and transposition errors create (Types 1 and 3 respectively).

### 5.3 .GOV Collection Results

We continue our experiments with a different collection, the .GOV collection, in order to achieve an overview of how these classical algorithms perform with a different and larger data set.

#### 5.3.1 Errors Correction Graphs

Similar to Section 5.2.1, we graph the percentage of accurate corrections. For the .GOV collection, the LCS algorithm slightly outperformed the Edit Distance algorithm for deletion errors Type (2), as can be seen in Figure 2(b). This could be due to the fact that deletion errors still have their sequence of characters intact in a given query term, which is key to the LCS algorithm.

Figure 2 shows again that, overall, as the number of errors increase, it becomes increasingly difficult for the system to decide what the correct term should be from the possible corrections (Hypothesis 4). It is quite possible that the system will choose a term that has the same property (same Edit Distance or LCS value) as another term yet the former is not the desired term.

Similar to the WT2G collection in Section 5.2.1, the Edit Distance algorithm outperformed the LCS algorithm for most

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>48181</b>	<b>2279</b>	<b>1682</b>	<b>0.2971</b>	<b>0.3263</b>	<b>0.4880</b>
Type 1 - 1 error (no correction)	43499	2279	1153	0.1452	0.1697	0.2320
Edit Distance Correction	48781	2279	1665	0.2609	0.2890	0.4140
LCS Correction	48616	2279	1334	0.2040	0.2335	0.3360
Type 1 - 2 errors (no correction)	46440	2279	1154	0.1330	0.1556	0.2340
Edit Distance Correction	48781	2279	1519	0.2210	0.2490	0.3760
LCS Correction	47902	2279	1330	0.2061	0.2318	0.3260
Type 1 - 3 errors (no correction)	41585	2279	1013	0.1548	0.1823	0.2220
Edit Distance Correction	48781	2279	1684	0.2779	0.3107	0.4540
LCS Correction	48181	2279	1255	0.1937	0.2280	0.3200

**Table 1: The precision/recall value for correction of character addition errors (WT2G).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>48181</b>	<b>2279</b>	<b>1682</b>	<b>0.2971</b>	<b>0.3263</b>	<b>0.4880</b>
Type 2 - 1 error (no correction)	46378	2279	1072	0.1244	0.1390	0.2220
Edit Distance Correction	47658	2279	1334	0.1938	0.2194	0.3180
LCS Correction	48107	2279	1088	0.1676	0.1937	0.2680
Type 2 - 2 errors (no correction)	46390	2279	1072	0.1229	0.1352	0.2240
Edit Distance Correction	47433	2279	1234	0.1405	0.1483	0.2360
LCS Correction	48569	2279	1137	0.1525	0.1723	0.2560
Type 2 - 3 errors (no correction)	46495	2279	1079	0.1222	0.1373	0.2200
Edit Distance Correction	47237	2279	1655	0.1263	0.1392	0.2300
LCS Correction	48674	2279	1096	0.1293	0.1417	0.2220

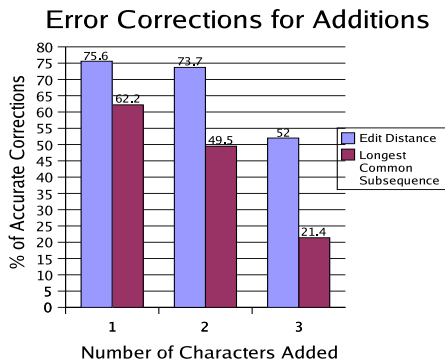
**Table 2: The precision/recall value for correction of character deletion errors (WT2G).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>48181</b>	<b>2279</b>	<b>1682</b>	<b>0.2971</b>	<b>0.3263</b>	<b>0.4880</b>
Type 3 - 1 error (no correction)	43751	2279	1107	0.1391	0.1616	0.2420
Edit Distance Correction	48336	2279	1294	0.2224	0.2371	0.3680
LCS Correction	47487	2279	1213	0.1958	0.2210	0.3200
Type 3 - 2 errors (no correction)	43749	2279	1106	0.1390	0.1616	0.2420
Edit Distance Correction	48449	2279	1465	0.2369	0.2513	0.3740
LCS Correction	49443	2279	1163	0.1799	0.2027	0.3080
Type 3 - 3 errors (no correction)	43749	2279	1106	0.1390	0.1616	0.2420
Edit Distance Correction	46137	2279	1243	0.1739	0.1919	0.3120
LCS Correction	48289	2279	1024	0.1632	0.1816	0.2720

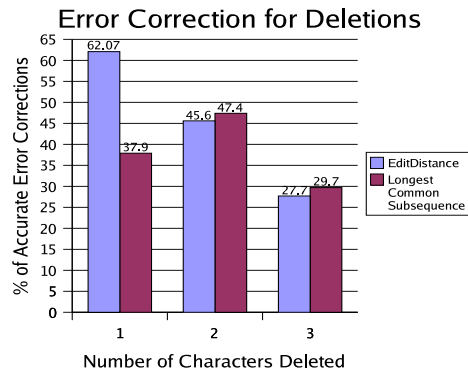
**Table 3: The precision/recall value for correction of character substitution errors (WT2G).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>48181</b>	<b>2279</b>	<b>1682</b>	<b>0.2971</b>	<b>0.3263</b>	<b>0.4880</b>
Type 4 - 1 error (no correction)	46367	2279	1174	0.1489	0.1706	0.2600
Edit Distance Correction	48546	2279	1284	0.1993	0.2115	0.3160
LCS Correction	48053	2279	1225	0.1785	0.1952	0.3160
Type 4 - 2 errors (no correction)	45719	2279	1095	0.1356	0.1569	0.2380
Edit Distance Correction	48781	2279	1389	0.2248	0.2518	0.3680
LCS Correction	49787	2279	1108	0.1748	0.1944	0.3020

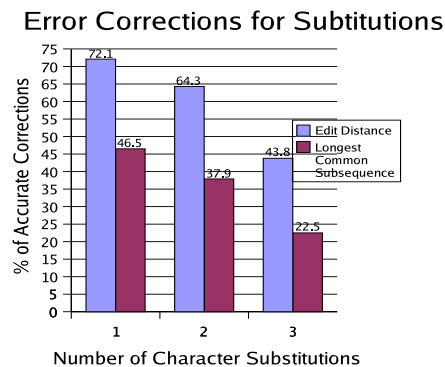
**Table 4: The precision/recall value for correction of character reversal errors (WT2G).**



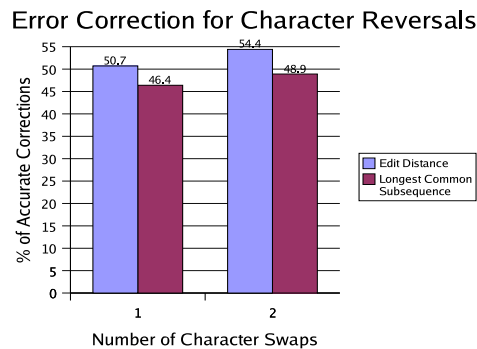
(a) Character additions (Type 1).



(b) Character deletions (Type 2).



(c) Character substitutions (Type 3).



(d) Character reversals (Type 4).

**Figure 2: Percentage of accurate distance corrections for various degrees of four error types (.GOV).**

experiments. In particular, there is a 25.6% difference between the two algorithms for the correction of a one character substitution (Type 3 with Figure 2(c)). A possible reason for this behaviour is that the Edit Distance actually takes into account three out of the four types of error tested, as was mentioned in Hypothesis 3.

### 5.3.2 Precision/Recall Results

As we did for the WT2G collection, we calculate the precision/recall measures for the .GOV collection for each error type (Tables 5, 6, 7 and 8).

In general, average precision is always better *with* error correction than without correction. In almost all cases, the number of relevant documents retrieved also increases when using error correction. There was a slight reduction in relevant documents retrieved when there were increasingly more deletion of characters (Table 6).

It can be seen that the average precision is consistent with the results for percentage of successful error corrections found in Subsection 5.3.1. When the percentage of accurate corrections is high, average precision is high and vice versa.

Much like the precision/recall values found in Section 5.2.2, the Edit Distance performs well on all types of error sup-

porting Hypothesis 1.

Not only did precision/recall improve with the algorithms but the number of relevant documents retrieved also increased. In particular, for terms with three extra characters, the Edit Distance managed to increase the number of relevant documents retrieved from 952 to 1371 (Table 5). This is an extra 419 relevant documents. This value is just 11 documents less than when the original queries without spelling mistakes are used (our baseline).

The results for the Longest Common Subsequence algorithm, while not as high as the Edit Distance algorithm, still managed to improve precision/recall values. Hence, Hypotheses 1 and 3 still hold.

As pointed out in the previous subsection, the Edit Distance did not outperform the LCS in all the experiments. The LCS algorithm did manage to perform better in two of the experiments. Without any error correction, average precision dropped by 0.0854 when two characters were transposed (Table 8). Using Edit Distance correction, average precision was improved slightly but more significantly, LCS correction had an average precision more than double that when no error correction was used.

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>98477</b>	<b>2090</b>	<b>1382</b>	<b>0.1334</b>	<b>0.1596</b>	<b>0.1545</b>
Type 1 - 1 error (no correction)	97115	2090	949	0.0624	0.0725	0.0838
Edit Distance Correction	98780	2090	1338	0.1187	0.1374	0.1394
LCS Correction	98780	2090	1208	0.0968	0.1087	0.1202
Type 1 - 2 errors (no correction)	97111	2090	952	0.0626	0.0725	0.0848
Edit Distance Correction	98780	2090	1349	0.1279	0.1502	0.1424
LCS Correction	99068	2090	1208	0.1001	0.1198	0.1202
Type 1 - 3 errors (no correction)	97111	2090	952	0.0626	0.725	0.0848
Edit Distance Correction	99068	2090	1371	0.1150	0.1343	0.1434
LCS Correction	99428	2090	1160	0.0853	0.1012	0.1162

**Table 5: The precision/recall value for correction of character addition errors (.GOV).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>98477</b>	<b>2090</b>	<b>1382</b>	<b>0.1334</b>	<b>0.1596</b>	<b>0.1545</b>
Type 2 - 1 error (no correction)	97948	2090	893	0.0554	0.0634	0.0737
Edit Distance Correction	95049	2090	971	0.0571	0.0704	0.0838
LCS Correction	95049	2090	900	0.0506	0.0569	0.0707
Type 2 - 2 errors (no correction)	97403	2090	875	0.0573	0.0668	0.0778
Edit Distance Correction	97602	2090	928	0.0671	0.0755	0.0919
LCS Correction	99585	2090	871	0.0542	0.0626	0.0798
Type 2 - 3 errors (no correction)	96950	2090	870	0.0532	0.598	0.0727
Edit Distance Correction	96496	2090	862	0.0492	0.0518	0.0697
LCS Correction	98458	2090	817	0.0515	0.0512	0.0768

**Table 6: The precision/recall value for correction of character deletion errors (.GOV).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>98477</b>	<b>2090</b>	<b>1382</b>	<b>0.1334</b>	<b>0.1596</b>	<b>0.1545</b>
Type 3 - 1 error (no correction)	97196	2090	856	0.0524	0.0623	0.0687
Edit Distance Correction	99463	2090	1266	0.0913	0.1153	0.1071
LCS Correction	98374	2090	960	0.0647	0.0793	0.0808
Type 3 - 2 errors (no correction)	97188	2090	861	0.0529	0.0633	0.0687
Edit Distance Correction	97524	2090	1221	0.0820	0.0975	0.0980
LCS Correction	98883	2090	969	0.0736	0.0916	0.0909
Type 3 - 3 errors (no correction)	97188	2090	834	0.0514	0.0606	0.0677
Edit Distance Correction	96538	2090	980	0.0716	0.0741	0.0879
LCS Correction	97334	2090	863	0.0514	0.0565	0.0657

**Table 7: The precision/recall value for correction of character substitution errors (.GOV).**

Error Type	Retrieved	Relevant	Relevant Retrieved	Average Precision	R Precision	Precision at 10
<b>Original Queries</b>	<b>98477</b>	<b>2090</b>	<b>1382</b>	<b>0.1334</b>	<b>0.1596</b>	<b>0.1545</b>
Type 4 - 1 error (no correction)	97932	2090	876	0.0480	0.0516	0.0636
Edit Distance Correction	99003	2090	1111	0.0675	0.0751	0.0879
LCS Correction	98780	2090	1208	0.0968	0.1087	0.1202
Type 4 - 2 errors (no correction)	97932	2090	879	0.0483	0.0528	0.0636
Edit Distance Correction	100000	2090	1209	0.0813	0.0958	0.1091
LCS Correction	98071	2090	1019	0.0688	0.0775	0.0909

**Table 8: The precision/recall value for correction of character reversal errors (.GOV).**

### 5.3.3 Combinations of errors

We now look at what happens when a term contains a combination of two of the types of errors tested. We tested for each possible combination of the four error types mentioned in Section 2. For example, we tested the case when a term had an addition error (Type 1), and character substitution error (Type 3), like misspelling the word ‘*housing*’ with ‘*huosinnng*’. Again, we compare the results of the Edit Distance algorithm with the Longest Common Subsequence algorithm, found in Table 9.

We saw that the Edit Distance algorithm performs well particularly for Types 1 and 3, (Figures 2(a) and 2(c) respectively). The algorithm achieved over 60% accurate corrections for up to two errors for both error types. Hence it is reasonable to expect that a combination of both an addition and a substitution will perform just as well (Hypothesis 5). When the results from the experiments were analysed, this was indeed the case. The Edit Distance managed to correct 79.7% of the errors that had both a character addition error and a character substitution error supporting our hypothesis.

Furthermore, the Edit Distance algorithm does not dominate the results as much as when there was only one type of error. The LCS algorithm outperformed the Edit Distance in two cases: a combination of an addition and a character reversal (Types 1 and 4), and a combination of a character deletion and a character reversal (Types 2 and 4). It is clear that the latter combination is the hardest to correct since both algorithms failed to correct even half of the errors, suggesting that the type and combination of errors do affect a spelling error correction algorithm.

## 5.4 Discussion

We have analysed our results with respect to two collections. We now look at all the results as a whole and draw our conclusions to give a more general overview.

We have seen that, overall, the Edit Distance performed better than the Longest Common Subsequence algorithm. In both collections the Edit Distance algorithm performs well. Addition, deletion and substitution errors have been corrected to a much higher success rate than reversal errors as the algorithm takes this into account in its calculations (Hypothesis 3).

The size of the corpora and content does not seem to be an important factor (Hypothesis 6). Indeed, our experimental results from the WT2G and the .GOV collections are consistent (Figures 5.2.1 and 5.3.1, and Tables 1 - 8).

While we have said that the Edit Distance generally performs better than the LCS (Hypothesis 3), the latter algorithm is not without merits. The LCS is better for the case when there are three deletion errors. Three deletion errors appear to be the breakdown point in performance for the Edit Distance. While the LCS’s degradation in accurate corrections is still apparent, the degradation is less severe as the number of errors in a query term increase (Figures 1(b) and 2(b)).

There is a decrease in performance for query terms with

three errors regardless of the error type. However, the Edit Distance is still able to accurately correct around half of the mistakes made (apart from deletion errors). Hence, this may or may not be the cut-off point in which the algorithm’s costs outweigh its benefits. Our experimental evidence suggests that the LCS’s limit is at two errors. Its performance, while not as good as the Edit Distance, manages to cope with one error well enough but after that only character reversal and addition errors are corrected at a rate of around 50%.

Transposition errors appear to be the most difficult to correct. The two algorithms can only manage to accurately correct about half of errors for just one transposition error. In this case, the percentage of accurate corrections for the two algorithms is similar to each other. The Edit Distance algorithm uses two edits rather than one to correct this type of error and so is bound not to perform as well as on the other error types. This is still in line with the LCS’s performance rate.

For the case of query terms consisting of a combination of error types, the performance of the two algorithms was more suggestive in terms of which algorithm performed better. For terms that have both an addition error and a character deletion, the Edit Distance error is best suited bettering the LCS algorithm by 12.5%. Yet for terms containing a character deletion as well as a character reversal the Edit Distance only managed to correct under a third of the errors accurately while the LCS algorithm corrected nearly a half of the same errors. Based on our experiments on one collection (.GOV), the results highlight again the differences between the two algorithms and that the type of spelling error is a factor in how well an algorithm performs (Hypothesis 2).

Clearly, the best way to correct spelling errors is to first be able to detect what type of spelling it is. From there an algorithm could be picked which would guarantee the best chance of correcting the error accurately. To illustrate, if an error was detected as several deletion errors, then the system should use LCS as it performed better than the Edit Distance (Figure 2(b)). However, if no classification of error is made and a general error correcting algorithm was needed, then the Edit Distance algorithm is recommended as three of the different types of errors (Types 1, 2 and 3), are taken into consideration when calculating its distance. Finally, according to the obtained results, the rate of successful corrections correlates with the average precision found using precision/recall measures.

## 6. CONCLUSION AND FUTURE WORK

This paper has looked at two classical algorithms that can be used in IR systems to detect and correct spelling errors. These algorithms are the Levenshtein Distance algorithm (more commonly known as the Edit Distance algorithm) and the Longest Common Subsequence algorithm. The Edit Distance measures the difference between two strings while the LCS measures how much two strings have in common with one another.

Experiments were performed to evaluate how well the algorithms performed for four types of errors. We also experimented with these errors on two different sized data collections of varying content. This enabled us to observe how

Error Combination	Edit Distance	Longest Common Subsequence
Character Addition & Deletion	63.0%	50.5%
Character Addition & Reversal	63.0%	68.0%
Character Addition & Substitution	79.7%	66.6%
Character Deletion & Reversal	30.9%	48.4%
Character Deletion & Substitution	57.1%	45.9%
Character Substitution & Reversal	54.0%	51.0%

**Table 9: Percentage of accurate corrections of errors containing a combination of two error types.**

the used string analysis algorithms perform on a range of data with a variety of spelling errors. In our evaluation, we identified and analysed the properties that an error correction algorithm might have difficulty with and those that are relatively simple to correct accurately. It was found that the Edit Distance performed generally well and bettered the LCS three out of four times for both the WT2G and .GOV collection. This seems to suggest that error correction is independent of the collection used.

It has also been pointed out that the algorithms can be used as an approach to tackle combinations of these errors and that their performance is related to how the algorithm performed for each error singularly. If the types of error could somehow be detected beforehand then we could choose one error correction algorithm over another based on the results of our experiments and its success in correcting that type of error. This is certainly an area of the work done in this paper that could be further developed in the future and reinforces the notion that spelling error correction impacts querying performance on varying degrees depending on the algorithm used.

We have only looked at two string matching algorithms but there are others such as those used in correcting optical character recognition (OCR) errors [3, 11]. It would be worthwhile to explore whether these techniques could be applied to spelling errors in queries and if they have better success at correcting errors than the two classical algorithms. We have also briefly mentioned that the size of the query term can affect the performance of the algorithm as seen when the number of deletion errors was significant. Further work can be done on the effects of average word length which is dependant on what language is in use.

The experiments have also shown that without an error correction functionality for query terms, average precision can fall by more than half of its original value. With just a small increase in time processing (< 850ms per query) error detection and correction can prevent such a severe degradation in performance.

## 7. ACKNOWLEDGMENTS

This work is funded by a UK Engineering and Physical Sciences Research Council (EPSRC) project grant, number GR/R90543/01. The project funds the development of the Terrier IR framework (<http://ir.dcs.gla.ac.uk/terrier>).

## 8. REFERENCES

- [1] R. C. Angell, G. W. Freund, and P. Willett. Automatic spelling correction using a trigram

similarity measure. *Information Processing & Management*, 19(4): 255–261, 1983.

- [2] G. Amati and C. J. van Rijsbergen. Probabilistic Models of Information Retrieval based on Measuring the Divergence from Randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, October 2002.
- [3] S. M. Beitzel, E. C. Jenson, and D. A. Grossman. A Survey of Retrieval Strategies for OCR Text Collections. *Invited to the 2003 Symposium on Document Image Understanding Technologies*, Greenbelt, Maryland, April 2003.
- [4] N. Craswell and D. Hawking. Overview of the TREC-2002 Web Track. In *Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)*, Gaithersburg, MD, 2002.
- [5] N. Craswell and D. Hawking. Overview of the TREC-2003 Web Track. In *Proceedings of the Twelfth Text Retrieval Conference (TREC-2003)*, Gaithersburg, MD, 2003.
- [6] I. Durham, D. A. Lamb, and J. B. Saxe. Spelling correction in user interfaces. *Comm. ACM*, 26(10):764–773, 1983.
- [7] D. Hawking, E. Voorhees, N. Craswell and P. Bailey. Overview of the TREC-8 Web Track. In *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD, 131–150, 1999.
- [8] D. Harman. How effective is suffixing? In *Journal of the American Society for Information Science*, 42(1):7-15, 1991.
- [9] B. He and I. Ounis. A Study of Parameter Tuning for Term Frequency Normalization. In *Proceedings of the Twelfth ACM CIKM International Conference on Information and Knowledge Management*, New Orleans, LA, 10–16, November 2003.
- [10] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, 18(6):341–343, 1975.
- [11] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.
- [12] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[13] G. de B. Robinson. On the representations of the symmetric group. *American Journal of Math*, 60:745–760, 1938.

[14] U. Quasthoff. Tools for automatic lexicon maintenance: Acquisition, error correction, and the generation of missing values. In *Proceedings of the first International Conference on Language Resources & Evaluation*, ELRA 1998, S. 853-856.