Fast and Accurate Misspelling Correction in Large Corpora

Octavian Popescu
Fondazione Bruno Kessler
Trento, Italy
popescu@fbk.eu

Ngoc Phuoc An Vo
University of Trento
Fondazione Bruno Kessler
Trento, Italy
ngoc@fbk.eu

Abstract

There are several NLP systems whose accuracy depends crucially on finding misspellings fast. However, the classical approach is based on a quadratic time algorithm with 80% coverage. We present a novel algorithm for misspelling detection, which runs in constant time and improves the coverage to more than 96%. We use this algorithm together with a cross document coreference system in order to find proper name misspellings. The experiments confirmed significant improvement over the state of the art.

1 Introduction

The problem of finding the misspelled words in a corpus is an important issue for many NLP systems which have to process large collections of text documents, like news or tweets corpora, digitalized libraries etc. Any accurate systems, such as the ones developed for cross document coreference, text similarity, semantic search or digital humanities, should be able to handle the misspellings in corpora. However, the issue is not easy and the required processing time, memory or the dependence on external resources grow fast with the size of the analyzed corpus; consequently, most of the existing algorithms are inefficient. In this paper, we present a novel algorithm for misspelling detection which overcomes the drawbacks of the previous approaches and we show that this algorithm is instrumental in improving the state of the art of a cross document coreference system.

Many spelling errors in a corpus are accidental and usually just one or two letters in a word are affected, like *existnece* vs. the dictionary form *existence*. Such misspellings are rather a unique phenomenon occurring randomly in a text. For an automatic speller which has access to a dictionary, finding and compiling a list of correct candidates for the misspelled words like the one above is not very difficult. However, not all misspellings are in this category. To begin with, proper nouns, especially foreign proper names, are not present in the dictionary and their misspelling may affect more than one or two characters. Moreover, the misspelling of proper names may not be random, for example there might be different spellings of the same Chinese or Russian name in English, the incorrect ones occurring with some frequency. Also, especially if the corpus contains documents written by non native speakers, the number of characters varying between the correct and the actual written form may be more than two. In this case, finding and compiling the list of correct candidates is computationally challenging for traditional algorithms, as the distance between the source string and the words in the candidates list is high.

The Levenshtein distance has been used to compile a list of correct form candidates for a misspelled word. The Levenshtein distance between two strings counts the number of changes needed to transform one string into the other, where a change is one of the basic edit operations: deletion, insertion, substitution of a character and the transposition of two characters. The Edit Distance algorithm, (ED) computes the similarity between two strings according to the Levenshtein distance. Most of the random misspellings which are produced by a native speaker are within one or maximum two basic edit operations (Damerau, 1964). For this reason the ED algorithm is the most common way to detect and correct the misspellings. However, there is a major inconvenience associated with the use of ED, namely, ED

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1634–1642,
October 25-29, 2014, Doha, Qatar.

runs in quadratic time considering the length of the strings, O(n^2). The computation time for more than a few thousands pairs is up to several tens of seconds, which is impracticably large for most of large scale applications. By comparison, the number of proper names occurring in a medium sized English news corpus is around 200, 000, which means that there are some 200, 000, 000 pairs.

In order to cope with the need for a lower computation time, on the basis of ED, a series of algorithms have been developed that run in linear time (Navaro 2001). Unfortunately, this improvement is not enough for practical applications which involve a large amount of data coming from large corpora. The reason is two-fold: firstly, the linear time is still too slow (Mihov and Schulz, 2004) and secondly, the required memory depends both on the strings' length and on the number of different characters between the source string and the correct word, and may well exceed several GBs. Another solution is to index the corpus using structures like trie trees, or large finite state automata. However, this solution may require large amounts of memory and is inefficient when the number of characters that differ between the source string and the candidate words is more than two characters (Boytsov, 2011).

We focus specifically on misspellings for which there is no dictionary containing the correct form and/or for which the Levenshtein distance to the correct word may be higher than two characters. For this purpose, we developed a novel approach to misspelling correction based on a non indexing algorithm, which we call the prime mapping algorithm, PM. PM runs in constant time, O(1), with insignificant memory consumption. The running time of the PM algorithm does not depend either on the strings' length or on the number of different characters between the source string and the candidate word. It requires a static amount of memory, ranging from a few KBs to a maximum of a few MBs, irrespective of the size of the corpus or the number of pairs for which the misspelling relationship is tested. We run a series of experiments using PM on various corpora in English and Italian. The results confirm that PM is practical for large corpora. It successfully finds the candidate words for misspellings even for large Levenshtein distances, being more than 30 times faster than a linear algorithm, and several hundred times faster than ED. The running time difference is due to the fact that PM maps the strings into numbers and performs only one arithmetic operation in order to decide whether the two strings may be in a misspelling relationship. Instead of a quadratic number of characters comparisons, PM executes only one arithmetic operation with integers.

We also report here the results obtained when using PM inside a cross document coreference system for proper nouns. Correcting a proper name misspelling is actually a more complex task than correcting a misspelled common word. Some misspellings may not be random and in order to cope with repetitive misspellings, as the ones resulting from the transliteration of foreign names, the PM is combined with a statistical learning algorithm which estimates the probability of a certain type of misspelling considering the surrounding characters in the source string. Unlike with common words, where a misspelling is obvious, in the case of proper names, *John* vs. *Jon* for example, it is unclear whether we are looking at two different names or a misspelling. The string similarity evidence is combined with contextual evidence provided by a CDC system to disambiguate.

To evaluate the PM algorithm we use publicly available misspelling annotated corpora containing documents created by both native and non-native speakers. The PM within a CDC system for proper names is evaluated using CRIPCO (Bentivogli et al., 2008). The experiments confirm that PM is a competitive algorithm and that the CDC system gains in accuracy by using a module of misspelling correction.

The rest of the paper is organized as follows. In Section 2 we review the relevant literature. In Section 3 we introduce the PM algorithm and compare it against other algorithms. In Section 4 we present the CDC system with misspelling correction for proper names. In Section 5 we present the results obtained on English and Italian corpora.

2 Related Work

In a seminal paper (Damerau, 1964) introduced the ED algorithm. The rationale for this algorithm was the empirical observation that about 80% of the misspelled words produced by native speakers have distance 1 to the correct word. ED cannot be extended to increase the accuracy, because for k = 2, k being the maximal admissible distance to the correct word, the running time is too high. Most of the techniques developed further use ED together with indexing methods and/or parallel processing.

In (San Segundo et al., 2001) an M-best can-

didate HMM recognizer for 10,000 Spanish city names is built for speech documents. An N-gram language model is incorporated to minimize the search spaces. A 90% recognition rate is reported. The model is not easily generalizable to the situation in which the names are unknown - as it is the case with the personal proper names in a large corpus. The N-gram model is memory demanding and for 200,000 different names the dimension of the requested memory is impracticably big.

The problem related to personal proper names was discussed in (Allan and Raghavan, 2002). However, the paper addresses only the problem of clustering together the names which "sound alike" and no cross document coreference check was carried out. The technique to find similar names is based on a noisy channel model. The conditional probabilities for each two names to be similarly spelled are computed. The time complexity is quadratic, which renders this technique unfeasible for big data. In fact, the results are reported for a 100 word set. A different approach comes from considering search queries databases (Bassil and Alwani, 2012). These techniques are similar to the model based on the noisy channel, as they compute the conditional probabilities of misspellings based on their frequencies in similar queries. Unfortunately, large numbers of queries for proper names are not available. A similar technique, but using morphological features, was presented in (Veronis, 1988). The method can manage complex combinations of typographical and phonographic errors.

It has been noted in many works dedicated to error correction, see among others (Mihov and Schulz, 2004), that the ED algorithm is impracticably slow when the number of pairs is large. A solution is to build a large tries tree. While this solution improves the searching time drastically, the memory consumption may be large. Automata indexing was used in (Oflazer, 1996). While the memory consumption is much less than for the tries tree approaches, it is still high. For Turkish, the author reported 28,825 states and 118,352 transitions labeled with surface symbols. The recovery error rate is 80%. In (Boytsov, 2011) a review of indexing methods is given. Testing on 5,000 strings for k=1,2,3 is reported and the paper shows the problem the systems run into for bigger values of k. In (Huldén, 2009) a solution employing approximations via an A* strategy with finite automata is presented. The method is much faster for k bigger than the one presented in (Chodorow and Leacock, 2000). However, the usage of A* for proper names may be less accurate than the one reported in the paper, because unlike the common words in a given language, the names may have unpredictable forms, especially the foreign names. The results reported show how the time and memory vary for indexing methods according to the length of the words for k=1,2,3.

A method that uses mapping from strings to numbers is presented in (Reynaert, 2004). This method uses sum of exponentials. The value of the exponential was empirically found. However, the mapping is only approximative. Our mapping is precise and does not use exponential operations which are time consuming.

The study in (Navarro, 2001) is focused on non indexing approximate string search methods, in particular on the simple ED distance. The non-indexing methods may reach linear running time, but it is not always the case that they are scalable to big data. In (Nagata et al., 2006) a study on the type of errors produced by non-native speakers of English is carried out, but the long distance misspellings are not considered.

3 Prime Mapping Misspeling Algorithm

The algorithms based on the Levenshtein distance use the dynamic programming technique to build a table of character to character comparisons. We present here a novel approach to misspelling which does not build this table, skipping the need to compare characters. In a nutshell, the prime mapping algorithm, PM, replaces the characters compare operations to a unique arithmetic operation.This can be done by associating to any letter of the alphabet a unique prime number. For example we can associate 2 to a, 3 to b, 5 to c ... 97 to z. Any string will be mapped into a unique number which is the product of the prime numbers corresponding to its letters. For example the name $abba$ is mapped to $2 \cdot 3 \cdot 3 \cdot 2 = 36$. By computing the ratio between any two words we can detect the different letters with just one operation. For example, the difference between $abba$ and aba is 36/12 = 3, which corresponds uniquely to b because the product/ratio of prime numbers is unique.

Unlike the ED algorithm, the prime mapping does not find the number of edit operations needed to transform one string into another. In fact, two words that have just one letter in the mutual difference set may be quite distinct: all the strings

aba, aab, baa differ by one letter when compared with $abba$. In order to be in a misspelling relationship, the two strings should also have a common part, like prefix or middle, or suffix. The complete Prime Mapping (PM) algorithm consists of two successive steps: (1) find all the candidate words that differ from the target word by at most *k* characters and (2) check weather the target word and the candidate word have a common part, suffix, prefix or middle part. Both steps above are executed in constant time, therefore they do not depend either on the length of the strings or on k, the maximal number of different characters. Normally, $k = 3$, because the probability of a misspelled word having more than three distinct letters is insignificant, but unlike in the case of ED, the choice of k has no influence on the running time. The first step takes an integer ratio and a hash table key check, both being O(1). The second step checks if the first k letters at the beginning or at the end of the word are the same, and it requires $2k$ character comparisons, which is also an O(1) process, as *k* is fixed. The pseudo code and detailed description of the PM algorithm are given below.

Algorithm 1 Prime Mapping

Require: $charList$ $wordsList$, $primeList$, k
Ensure: $misspList$
1: $misspList \leftarrow \emptyset$
2: **foreach** α **in** $charList$: p(α) $\leftarrow$ p_i, p_i **in** $primeList$
3: **foreach** w **in** $wordsList$: p(w) $\leftarrow \prod p(\alpha)$, α **in** w
4: $primeKTable \leftarrow \binom{n}{k}$ of prime arithmetics
5: **for** w **in** $wordsList$ **do**
6: **for** w' **in** $wordsList$, w $\neq$ w' **do**
7: r $\leftarrow \frac{p(w)}{p(w')}$
8: **if** r **in** $primeKTable$ **then**
9: **if** commonPart (w, w') $\neq \emptyset$ **then**
10: $misspList \leftarrow misspList$ + (w, w')
11: **end if**
12: **end if**
13: **end for**
14: **end for**

map letters to prime numbers. A helpful way to assign primes to letters is according to their frequency; on average, the numbers corresponding to names are smaller and the operation gets less time.

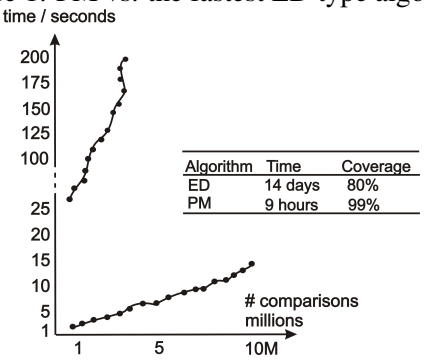
compute a hash table with prime arithmetics of K primes. In the hash table *primeKTable* we record all the combinations that can result from dividing two products which have less than *k* primes: $1/p_i$, p_i, p_i/p_j etc. If the ratio between two mappings is in the hash table, then the corresponding words have all the letters in common, except for at most *k*. The number of all the combination is $\binom{n}{k}$. The memory consumption for different values for *k* is given in Table 1. The figures compare extremely favorably with the ones of ED based approaches (gigs magnitude) . (line 7-8)

k letter difference	*#combination*	*Memory*
1	60	480B
2	435	8K
5	142,506	0.9MB
6	593, 775	3.8MB
10	30, 045, 015	180MB

Table 1: The PM algorithm memory needs

find misspelling candidates by ratio. By computing the ratio and by checking the hash table, we found the pairs which use the same letters, except for at most *k*. The procedure *commonpart* checks whether the two strings also have a common part by looking at the start and end *k*. If this is the case, the pair is in a misspelling relationship.

Figure 1: PM vs. the fastest ED type algorithm



The PM is much faster than ED. The fastest variant of ED, which does not compare strings having length difference bigger than 1, theoretically finds only 80% of the misspellings. In practice, only around 60% of the misspellings are found because of proper names and words misspelled by non-native speakers. The PM algorithm considers all possible pairs, finds more than 99% of misspellings and is 35 times faster. To obtain the same coverage, the ED algorithm must run for more than 100 days. The time comparison for millions of pairs is plotted in Figure 1. The experiments were performed on an i5, 2.8 GHz processor.

There is an immediate improvement we can bring to the basic variant of PM. The figures reported above are obtained by doing the whole set of possible pairs. By taking into account the fact that two words differing by $k + 1$ letters cannot be k similar, we can organize the number representing the names into an array which reduced drasti-

cally the number of comparisons. For example, all the words containing the letters *x, y, z* cannot be $k = 2$ similar with the words not containing any of these letters. By dividing the mapping of a word to the primes associated with the letters of an k-gram, we know if the words containing the k-gram can be misspelling candidates with at most k difference, and there is no more need to carry out all the ratios. We arrange the mappings of all words into an array such that on the first indexes we have the words containing the less frequent $k + 1$ gram, on the next indexes we have the words containing the second less frequent $k+1$ gram and do not contain the first $k + 1$ gram, on the next indexes the words containing the third less frequent $k + 1$ gram and do not contain the first two $k + 1$ gram, etc. In this way, even the most frequent $k + 1$ gram has only a few words assigned and consequently the number of direct comparisons is reduced to the minimum. The mapping corresponding to a $k+1$ gram are ordered in this array according to the length of the words. The number of trigrams is theoretically large, the $k + 1$ power of the size of the alphabet. However, the number of actually occurring k-trigrams is only a small fraction of it. For example, for $k = 2$, the number of trigrams is a few hundred, out of the 2, 700 possible ones. PM2gram runs in almost a quarter of the time needed by the basic PM. For the same set of names we obtained the results reported in Table 2. The last column indicates how many times the algorithm is slower than the PM in its basic form.

algorithm	*time*	*coverage*	*times slower*
basicED	132 days	99%	310
ED1	14 days	80%	35
PM	9 hours	99%	1
PM2gram	2 hours 42min	96%	0.26

Table 2: ED variants versus MP

4 Correcting Proper Names Misspellings

In this section we focus on a class of words which do not occur in a priorly given dictionary and for which the misspelled variants may not be random. Proper names are representative for this class. For example, the same Russian name occurs in corpus as *Berezovski*, *Berezovsky* or *Berezovschi* because of inaccurate transliteration. By convention, we consider the most frequent form as the canonical one, and all the other forms as misspelled variants.

Many times, the difference between a canonical form and a misspelled variant follows a pattern: a particular group of letters substitutes another one in the context created by the other characters in the name. A misspelling pattern specifies the context, as prefix or suffix of a string, where a particular group of characters is a misspelling of another. See Table 3 for examples of such patterns.

Pattern	*Context*	*Example*
dj→dji	ovic	djiukanovic djukanovic
k→kh	aler	kaler khaler, taler thaler
ki→ky	ovsk	berezovski berezovsky
n→ng	chan	chan-hee chang-hee
dl→del	abd	abdelkarim abdlkrim

Table 3: Name misspellings patterns

Finding and learning such patterns, along with their probability of indicating a true misspelling, bring an important gain to CDC systems both in running time and in alleviating the data-sparseness problem. The CDC system computes the probability of coreference for two mentions t and t' using a similarity metrics into a vectorial space, where vectors are made out of contextual features occurring with t and t' respectively (Grishman, 1994). However, the information extracted from documents is often too sparse to decide on coreference (Popescu, 2009). Coreference has a global effect, as the CDC systems generally improve the coverage creating new vectors by interpolating the information resulting from the documents which were coreferred (Hastie et al., 2005). This information is used to find further coreferences that no single pair of documents would allow. Thus, missing a coreference pair may result in losing the possibility of realizing further coreferences. However, for two mentions matching a misspelling pattern which is highly accurate, the threshold for contextual evidence is lowered. Thus, correcting a misspelling is not beneficial for a single mention only, but for the accuracy of the whole.

The strategy we adopt for finding patterns is to work in a bootstrapping manner, enlarging the valid patterns list while maintaining a high accuracy of the coreference, over 90%. Initially, we start with an empty base of patterns. Considering only the very high precision threshold for coreference, above 98% certainty, we obtain a set of misspelling pairs. This set is used to extract patterns of misspellings via a parameter estimation found using the EM-algorithm. The pattern is considered valid only if it also has more than a given number of occurrences. The recursion of the previous steps is carried out by lowering with an ε the threshold for accuracy of coreference for pat-

tern candidates. The details and the pseudo code are given below.

Algorithm 2 Misspelling Pattern Extraction

Require: $thCoref$, ε, $minO$, $thAcc$
Require: $thCDC$
Ensure: $pattList$
1: $pattList, candPattList \leftarrow \emptyset$
2: **while** there is a pair (t, t') to test for coreference **do**
3: **if** (t, t') *matches* p, p **in** $pattList$ **then**
4: prob ← corefProb(p)
5: **else**
6: use PM algorithm on pair (t, t')
7: prob ← $thCoref$
8: **end if**
9: **if** pair (t, t') coref with prob **then**
10: $candPattList \leftarrow candPattList$ + (t, t')
11: **end if**
12: extractPatterns from $candPattList$
13: **for** cp **in** $new\ extracted\ patterns$ **do**
14: **if** #cp>$minO$ **and** corefProb(cp)>$thAcc$ **then**
15: $pattList \leftarrow pattList$ + (t, t')
16: **end if**
17: **end for**
18: **if** prob>$thCDC$ **then**
19: corefer (t, t')
20: **end if**
21: **end while**
22: $thCoref \leftarrow thCoref$ - ε
23: **goto** line 2

1. Compile a list of misspelling candidates For each source string, t, try to match t against the list of patterns (initially empty). If there is a pattern matching (t, t') then their prior probability of coreference is the probability associated with that pattern (line 4).

2. CDC coreference evidence For each pair (t ,t') in the canonical candidates list use the CDC system to compute the probability of coreference between t and t'. If the probability of coreference of t and t' is higher than $thCoref$, the default value is 98%, then consider t as a misspelling of t' and put (t, t') in a candidate pattern list (line 10).

3. Extract misspelling patterns Find patterns in the candidate pattern list. Consider only patterns with more than $minO$ occurrences, whose default value is 10, and which have the probability of coreference higher than $thAcc$, whose default value is 90% (line 15).

4. CDC and pattern evidence For each (t,t') pair matching a pattern and the CDC probability of coreference more then $thCDC$, whose default value is 80%, then corefer t and t' (line 21). The fact that the pair (t,t') matches a pattern of misspelling is considered supporting evidence for coreference and in this way it plays a direct role in enhancing the system coverage. Decrease $thCoref$ by ε,whose default is value 0.5, and repeat the process of finding patterns (goto line 2).

To extract the pattern from a given list of pairs, procedure extractPatterns at line 12 above, we generate all the suffixes and prefixes of the strings. We compute the probability that a group of characters represents a spelling error, given a certain suffix and/or prefix. We use the EM algorithm to compute these probabilities. For a pair (P, S) of a prefix and a suffix, the tuples (p(P)=p, p(S)=s, π) are the quantities to be estimated via EM, with π being the coreference probability. A coreference event is directly observable, without knowing, however, which prefix or suffix contribute to the coreference. The EM equations are given below, where X is the observed data; Z are the hidden variable, p and s respectively; θ the parameters (p,s, π); Q(θ,$\theta^{(t)}$) the expected log likelihood at iteration t.

$\mathbf{E-step}$ $\mu_i^{(t)}$

$$\begin{aligned}\mu_i^{(t)} &= E[z_i|x_i,\theta^{(t)}] \\ &= \frac{p(x_i|z_i,\theta^{(t)})\ p(z_i=P|\theta^{(t)})}{p(x_i|\theta^{(t)})} \\ &= \frac{\pi^{(t)}[p^{(t)}]^{x_i}[(1-p^{(t)}]^{(1-x_i)}}{\pi^{(t)}[p^{(t)}]^{x_i}[(1-p^{(t)}]^{(1-x_i)}+(1-\pi^{(t)})[s^{(t)}]^{x_i}[(1-s^{(t)}]^{(1-x_i)}}\end{aligned} \quad (1)$$

$\mathbf{M-step}$ $\theta^{(t+1)}$

$$\begin{aligned}&\frac{\partial Q(\theta|\theta^t)}{\partial \pi}=0 \quad \pi^{(t+1)}=\frac{\sum_i \mu_i^{(t)}}{n} \\ &\frac{\partial Q(\theta|\theta^t)}{\partial p}=0 \quad p^{(t+1)}=\frac{\sum_i \mu_i^{(t)} x_i}{\sum_i \mu_i^{(t)}} \\ &\frac{\partial Q(\theta|\theta^t)}{\partial s}=0 \quad s^{(t+1)}=\frac{\sum_i (1-\mu_i^{(t)}) x_i}{\sum_i (1-\mu_i^{(t)})}\end{aligned} \quad (2)$$

5 Experiments

We performed a set of experiments on different corpora in order to evaluate: (1) the performances of the PM algorithm for misspelling detection, (2) the accuracy of proper name misspelling pattern acquisition from large corpora, and (3) the improvements of a CDC system, employing a correction module for proper name misspellings.

In Section 5.1 the accuracy of the PM algorithm is tested on various corpora containing annotated misspellings of English words. In particular, we were interested to see the results when the edit distance between the misspelled pair is bigger than 3, because handling bigger values for k is crucial for finding misspelling errors produced by non-native speakers. The evaluation is directly relevant for the correction of the spelling of foreign names.

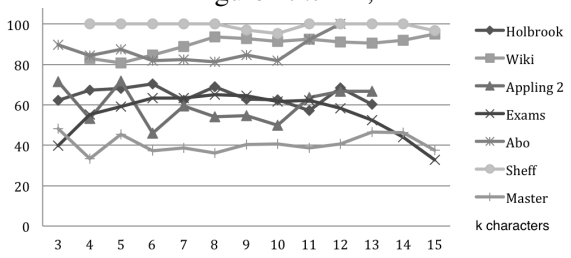
In Section 5.2 the proper name misspelling patterns were extracted from two large news corpora. One corpus is part of the English Gigawords, LDC2009T13 (Parker et al., 2009) and the second corpus is Adige500k in Italian (Magnini et al., 2006). We use a Named Entity Recognizer which has an accuracy above 90% for proper names. We evaluated the accuracy of the patterns by random sampling.

In Section 5.3 the accuracy of the CDC system with the correction module for proper name misspellings was tested against a gold standard.

5.1 PM Evaluation

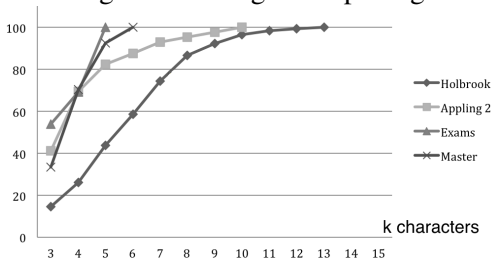
We consider the following publicly available English corpora containing the annotation of the misspelled words: Birkbeck, Aspell, Holbrook, Wikipidia. Birkbeck is a heterogeneous collection of documents, so in the experiments below we refer to each document separately. In particular we distinguish between misspellings of native speakers vs. misspelling of non-native speakers. Figure 2 shows that there are two types of corpora. For the first type, the misspellings found within two characters are between 80% and 100% of the whole number of misspellings. For the second type, less than 50% of the misspellings are within two characters.The second category is represented by the misspellings of non native speakers. The misspellings are far from the correct forms and they represent chunks of phonetically similar phonemes, like *boiz* vs. *boys*. The situation of the foreign name misspellings is likely to be similar to the misspellings found in the second type of corpora. For those cases, handling a k value bigger than 2 is crucial. Not only the

Figure 2: k = 1, 2



non-indexing methods, but also indexing ones are rather inefficient for k values bigger than 2 for large corpora. The PM algorithm does not have this drawback, and we tested the coverage of the errors we found for values of k ranging from 3 to 10. In Figure 3 we plot the distributions for the

Figure 3: Foreign Misspellings



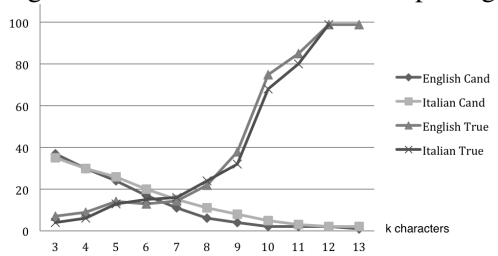
corpora which are problematic for k=2. Values of k are plotted on the OX axis, and the percentage of the misspellings within the respective k on the OY axis. The results showed PM is also able to find the phonemically similar misspellings. We can see that for k bigger than 9 the number of misspellings is not significant.

The PM algorithm performed very well, being able to find the misspellings even for large k values. There were 47, 837 words in Aspell, Holbrrok and Wikipedia, and 30, 671 in Birkbeck, and PM found all the misspelling pairs in a running time of 25 minutes. This is a very competitive time, even for indexing methods. For k above 8 the access to the hash table containing the prime combinations was slower, but not significantly so.

5.2 Pattern Extraction Evaluation

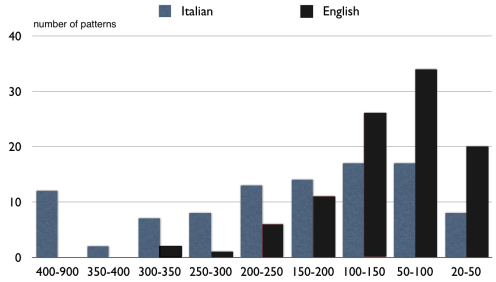
We extracted the set of names using a NER from the two corpora, LDC2009T13 and Adige500k. The set of proper names is rather large in both corpora - 160, 869 names from the English corpus and 185, 508 from the Italian corpus. Apparently, the quasi-similar names, which are considered as misspelled name candidates, is very high. In Figure 4 we plot this data. The *English Cand* and *Italian Cand* are absolute values, while the *English True* and *Italian True* represent percentages. For example, a name of length 5 is likely to have around 23 misspelling candidates, but only 17% of them are likely to be true misspellings, the rest being different names.

Figure 4: Candidates vs. True Misspellings



The numbers are estimated considering samples having the size between 30 and 50, for each name length. The percentages change rapidly with the length of the string. For names with the length bigger than 11, the probability that a misspelling candidate is a true misspelling is more than 98%. This fact suggests a strategy for pattern extraction: start from the higher name length towards the lower length names. The patterns found by the algorithm described in Section 4 have between 900 and 20 occurrences. There are 12 patterns having more than 400 occurrences, 20 having between 20 and 50 occurrences, see Fig. 5.

Figure 5: Distribution of the patterns:

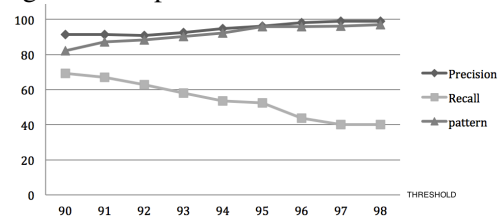


5.3 CDC and Misspelling correction

The CRIPCO corpus (Bentivogli et al., 2008) is a gold standard for CDC in Italian, containing pieces of news extracted from Adige500k. There are 107 names, the majority being Italian names. We scrambled the names to create misspelling candidates. For example the name $leonardo$ was scrambled like $teonardo$, $lionaldo$, $loenarod$ etc. We considered the top 15 frequency letters and maximum 4 letters for each scrambling. We randomly selected 70% of the original CRIPCO making no modifications, and called this corpus CRwCR. 30% of the original documents were assigned to the invented pseudo-names, and we called this corpus CRwSC (correct documents with scrambled names). From Adige500k we randomly chose 20, 000 documents and assigned them to the scrambled names as well, calling this corpus NCRwSC. From these pieces we created a new corpus: 70% of the initial CRIPCO documents with the original names, 30% of the CRIPCO documents with scrambled names and 20, 000 documents with the same scrambled names. For the names occurring in CRwCR, the scrambled names are valid name misspellings in the CRwSC corpus, and invalid in NCRwSC.

As expected, the PM algorithm found all the misspelling candidates and some others as well. We let the threshold confidence of coreference to vary from 90% to 98%. The number in Figure 6 refers to the precision and recall for the name misspellings in the CRIPCO corpus created via random scrambling. We were also interested to see how the pattern finding procedure works, but scrambling randomly produced too many contexts. Therefore, we chose to modify the names in a non random way, by replacing the final o to ino, ex. $paolo$ goes to $paolino$, and modifying one letter in the word for half of the occurrences, ex. $paorino$. The idea is that ino is a very common suffix for names in Italian. The system was able to learn the pseudo alternatives created in the context ino. The noise introduced was relatively low, see Fig. 6.

Figure 6: Proper Names CRIPCO Evaluation



6 Conclusion and Further Research

In this paper we described a system able to correct misspellings, including proper name misspellings, fast and accurately. The algorithm introduced, PM, overcomes the time/memory limitations of the approaches based on the edit distance.

The system is built on a novel string compare algorithm which runs in constant time independently of the length of the names or the number of different letters allowed, with no auxiliary memory request. As such, the algorithm is much faster than any other non-indexing algorithms. Because it is independent of k, it can be used even for large k, where even the indexing methods have limitations. We also used an EM based technique to find misspelling patterns. The results obtained are very accurate.

The system makes a first selection of the documents, drastically reducing the human work load. Another line of future research is to use the PM algorithm in other NLP tasks, where finding the pairs having some particular elements in common is necessary: for example, comparing parsing trees or dependency trees. We think that PM can be used in other NLP tasks as well and we hope the community can take advantage of it.

References

James Allan and Hema Raghavan. 2002. Using Part-of-Speech Patterns to Reduce Query Ambiguity. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 307–314. ACM.

Youssef Bassil and Mohammad Alwani. 2012. OCR Post-Processing Error Correction Algorithm Using Google's Online Spelling Suggestion. *Journal of Emerging Trends in Computing and Information Sciences, ISSN 2079-8407, Vol. 3, No. 1*.

Luisa Bentivogli, Christian Girardi, and Emanuele Pianta. 2008. Creating a Gold Standard for Person Cross-Document Coreference Resolution in Italian News. In *The Workshop Programme*, page 19.

Leonid Boytsov. 2011. Indexing Methods for Approximate Dictionary Searching: Comparative Analysis. *Journal of Experimental Algorithmics (JEA)*, 16:1–1.

Martin Chodorow and Claudia Leacock. 2000. An Unsupervised Method for Detecting Grammatical Errors. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 140–147. Association for Computational Linguistics.

Fred J Damerau. 1964. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176.

Ralph Grishman. 1994. Whither Written Language Evaluation? In *Proceedings of the workshop on Human Language Technology*, pages 120–125. Association for Computational Linguistics.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. 2005. The Elements of Statistical Learning: Data Mining, Inference and Prediction. *The Mathematical Intelligencer*, 27(2):83–85.

Måns Huldén. 2009. Fast Approximate String Matching with Finite Automata. *Procesamiento del lenguaje natural*, 43:57–64.

Bernardo Magnini, Emanuele Pianta, Christian Girardi, Matteo Negri, Lorenza Romano, Manuela Speranza, Valentina Bartalesi Lenzi, and Rachele Sprugnoli. 2006. I-CAB: The Italian Content Annotation Bank. In *Proceedings of LREC*, pages 963–968.

Stoyan Mihov and Klaus U Schulz. 2004. Fast Approximate Search in Large Dictionaries. *Computational Linguistics*, 30(4):451–477.

Ryo Nagata, Koichiro Morihiro, Atsuo Kawai, and Naoki Isu. 2006. A Feedback-Augmented Method for Detecting Errors in The Writing of Learners of English. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 241–248. Association for Computational Linguistics.

Gonzalo Navarro. 2001. A Guided Tour to Approximate String Matching. *ACM computing surveys (CSUR)*, 33(1):31–88.

Kemal Oflazer. 1996. Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction. *Computational Linguistics*, 22(1):73–89.

Robert Parker, Linguistic Data Consortium, et al. 2009. *English Gigaword Fourth Edition*. Linguistic Data Consortium.

Octavian Popescu. 2009. Person Cross Document Coreference with Name Perplexity Estimates. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 997–1006. Association for Computational Linguistics.

Martin Reynaert. 2004. Text Induced Spelling Correction. In *Proceedings of the 20th international conference on Computational Linguistics*, page 834. Association for Computational Linguistics.

Rubén San Segundo, Javier Macías Guarasa, Javier Ferreiros, P Martin, and José Manuel Pardo. 2001. Detection of Recognition Errors and Out of the Spelling Dictionary Names in a Spelled Name Recognizer for Spanish. In *INTERSPEECH*, pages 2553–2556.

Jean Veronis. 1988. Morphosyntactic Correction in Natural Language Interfaces. In *Proceedings of the 12th conference on Computational linguistics-Volume 2*, pages 708–713. Association for Computational Linguistics.