Journal of King Saud University – Computer and Information Sciences xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Amazigh spell checker using Damerau-Levenshtein algorithm and N-gram

Youness Chaabi, Fadoua Ataa Allah *

CEISIC, The Royal Institute of Amazigh Culture, Morocco

ARTICLE INFO

Article history:
Received 23 March 2021
Revised 2 July 2021
Accepted 22 July 2021
Available online xxxx

Keywords:
Spellchecking
Amazigh Language
Lexical errors
N-gram
Levenshtein algorithm

ABSTRACT

Natural language processing (NLP) is a rapidly growing research field in computer science and cognitive science. Automatic correction of lexical errors is one of the applications of NLP, which aims to detect errors in the text and suggest possible corrections based on computer and linguistic models. After a state of the art and a comparison of spelling correction approaches, we developed a spelling error correction system for the Amazigh language combining the Damerau-Levenshtein algorithm and N-gram. This tool will propose possible corrections for each misspelled word in the text. Successful tests have been carried out using an Amazigh corpus.

1. Introduction

In the last few years, the automatic correction of lexical errors has become more and more useful for research in the field of Automatic Language Processing (ALP). Strong reasons for this include: the emergence of a range of practical applications, the integration of components for spelling and lexical checking in many applications such as word processing software, e-mail and web browsers. In this way, these components facilitate the writing of texts without errors, the incorporation of more sophisticated solutions in systems for complete correction and Computer-assisted language learning (CALL) (Zourou, 2009). In addition, fully automatic text correction is used as part of pre-processing or post-processing in several ALP tasks, such as Information Retrieval (IR), Optical Character Recognition (OCR), Automatic Speech Recognition (ASR), and Machine Translation (MT). Despite its popularity, automatic correction of lexical errors is far from being completely solved.

Spell checkers have been an area of research since the 1960s (Kukich, 1992). They are now commonly used by the general public, since common text editors often include one, and they provide a significant level of comfort when writing texts. These correctors operate in an interactive mode in which the user intervenes, unlike fully automatic spell checkers such as in the field of optical character recognition (which we do not address here).

In this paper, we are interested in the Amazigh language spelling correction, based on the combination of Damerau-Levenshtein algorithm and N-gram. This combination aims to reduce the error detection time and to ensure a good performance of our spelling correction system.

2. Spelling correction

Spelling correction consists in suggesting words closer to a wrong word. The search for solutions to this problem has long remained a challenge. Several researchers have studied this problem and thanks to their efforts, various techniques and many algorithms have been developed. They have classified errors into two types (Kukich, 1992):

- **Lexical errors:** also called usage errors. They are due to errors applied to the words taken in themselves, regardless of their role in the sentence (Hacken and Tschichold, 2001).

* Corresponding author.
E-mail addresses: chaabi@ircam.ma (Y. Chaabi), ataaallah@ircam.ma (F. Ataa Allah).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2021.07.015
1319-1578/

Please cite this article as: Y. Chaabi and F. Ataa Allah, Amazigh spell checker using Damerau-Levenshtein algorithm and N-gram, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2021.07.015

- **Grammatical errors:** connoted morphosyntactic errors. They include both grammatical spelling errors and syntactic errors. They relate to the combination of words or their grammatical modification, associated to conjugation or declension marks.

The causes of errors are numerous, they vary between cognitive and typographical errors (Peterson, 1980). Cognitive errors result from not knowing the correct spelling of the word. However, typographical errors occur when keyboard keys are pressed in the wrong order.

A spell checker is, essentially, proceeds in two stages: the detection and the correction of spelling errors. The detection and correction methods work according to errors' types (lexical vs grammatical), which are often treated separately in the literature (Kukich, 1992). In this paper, we focus on the approaches used for the detection and the correction of lexical spelling.

2.1. Detection

To suggest a set of possible corrections, a spellchecker has to identify first misspelled words. This involves the development of an error model. Research in this area has been carried out by many authors mentioned in the survey of Hládek et al. (2020). The main techniques used to identify erroneous words in a text are either based on dictionary research or on n-gram analysis.

2.1.1. Consulting the dictionary

The most common method used in spelling correction is based on consulting the dictionary (the corrector lexicon). The latter contains all the words of the language or at least the basic forms. The technique consists of identifying words that are absent from the corrector lexicon and marking them as incorrect. However, keeping all the words of the language in the proofreader's lexicon can make the search tedious. Therefore, only the basic shapes are stored in the lexicon and other tools such as morphological analyzers are used to generate the derived shapes. A morphological analyzer allows to derive the morphological construction of a known word or to make hypotheses about an unknown word. Morphology is traditionally divided into word formation derivational or compositional morphology and inflectional morphology. Inflectional morphology consists in conjugating a verb and declining a noun or adjective from a lexeme, while derivational morphology describes the composition of a word from roots and affixes, or the creation of a word from other words.

The most used tools for morphological analysis in spelling correction are lemmatisers, conjugators, decliners which are generally performed with automata.

In this step, several studies have been carried out to reduce response time when searching in a dictionary, among which we quote:

- **Hash table** is one of the most widely used data structures to reduce response time when searching a dictionary (Kukich, 1992). The hash table idea was first introduced in 1953 (Salifou and Naroua, 2014). It has the advantage of allowing, thanks to the hash code, selective access to the searched word. This considerably reduces the response time. However, the major disadvantage is that the hash function admits very few collisions and gives clues that are evenly distributed in the interval considered.
- **Binary search trees** are especially useful for checking whether a given word is part of a larger set of dictionary words (Bent et al., 1985). There are several variants of binary search trees that have been used to speed up dictionary searches as part of spell checking.
- **Finite automata** have also been used in some dictionary or text search algorithms. One of the famous algorithms in this field is that of Aho and Corasick (1975). The algorithm consists of advancing in an abstract data structure called a dictionary that contains the word or words searched for by reading the letters of the text one by one. The data structure is implemented efficiently, ensuring that each letter of the text is read only once. Usually, the dictionary is implemented using a digital sort or tree to which suffix links are added. A sort can be seen as a representation of the transition function of a deterministic finite automaton. Once the dictionary is implemented, the algorithm has a linear complexity in the size of the text and the searched strings.

2.1.2. N-gram analysis

The use of n-gram model in language processing was discussed first by Shannon (1951). After this contribution, the n-gram model has been applied to many problems including spelling correction. One of the main advantages of n-gram is that this model is language independent.

N-gram analysis is an error detection technique that allows to detect incorrect words in a text. It does not compare the text words to words in the dictionary, but establishes a comparison process in a square matrix of size n, which stores the most frequent n-grams accepted (Pollock and Zamora, 1984). Each text string is split into a set of adjacent n-grams. The development of a system based on the n-grams implies first of all to calculate the probability of error of each n-gram individually.

This probability is simply the number of times it appears on wrong words out of the total number of times it appears on words in the text. It is also possible that a sequence of n letters (n-gram) is considered valid in one context and invalid in another. This method therefore requires a learning corpus. Only the n-grams constructed from the incorrect words represent the invalid n-grams. An n-gram, which is both present on a word and on its incorrect form, is considered as not contributing to the error.

The probability of error (P) of an n-gram is given by the following formula $p = \frac{E}{E+V}$

where:

- E is the number of times it has been classified as an invalid n-gram;
- V is the number of times it has been classified as a valid n-gram.

2.2. Correction

The next stage after error detection in a spelling correction process is the generation of words that may be the correct form of the misspelled word. The correction techniques discussed below focus on the word but not on the context in which the word is to be used. The correction is performed in two steps:

- Generation of words that are likely to be the correct spelling.
- Ranking generated words, proposed in the first step.

2.2.1. Alpha-code

To find the closest forms of the unknown word, most spell checkers use a word representation coded. This string encoding technique is known as alpha-coding. The alpha-code of a word is the string of characters made up of all its letters in alphabetical order. It has different methods of computation:

- Mar Ndiaye and Anne V. Faltin proposed a method that consists in arranging the consonants followed by the vowels in alphabetical order (Ndiaye and Faltin, 2003). The set of words relative

to a given alpha-code is called the class of this alpha-code. The correction system has the set of alpha-codes corresponding to the lexicon and, for each one, the class of words.

- Joseph J. Pollock and Antonio Zamora proposed a model consisting in associating for each word in the dictionary its alpha-code (the consonants constituting the word) (Pollock and Zamora, 1984). Hence the need to have two dictionaries: one for the words and the other for their alpha-codes. Therefore, the correction is made by comparing the alpha-codes with the alpha-code of the erroneous word. This method is effective in the case of permutation errors.

The corrector first establishes the alpha-code of the wrong word. If this alpha-code is listed, the words of the corresponding class are retained as candidates. If the alpha-code is unknown, the system adds one then two characters to it and checks if the new alpha-code obtained exists.

In this case, the class is retained. It proceeds in the same way by deleting then substituting one or two characters. The candidate words obtained are then the subject of a proximity calculation with the incorrect string, which allows a ranking of this list of corrective words. A word has only one alpha-code but several words can share the same alpha-code. Thus, a spell checker will be able to look for words that contain the same alpha-code as the unknown word.

2.3. Lexicographical distance

The lexicographical distance calculates the minimum number of inserting, deleting, transposition and substituting letters necessary to transform one word into another (Vienney, 2004). The greater the number of differences between strings, the greater the distance. The most popular lexicographical distance is the Levenshtein distance, also known as "edit distance", although that term may also denote a larger family of distance metrics referred collectively as edit distance. This metric calculates the edit distance by considering three string operations: substitution, insertion and deletion (Levenshtein, 1966).

Let $lev_{i,j}$ be the Levenshtein distance between the subsequence formed with the i first characters of a sequence W_1 and the subsequence formed with the j first characters of a sequence W_2. The Levenshtein distance between the two subsequences W_1 and W_2 (of length $|W_1|$ and$|W_2|$ respectively) can be recursively calculated as follows:

$$lev_{i,j} = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} lev_{i-1,\,j} + 1 \\ lev_{i,\,j-1} + 1 \\ lev_{i-1,\,j-1} + I(w_{1i} \neq w_{2j}) \end{cases} & \text{otherwise} \end{cases}$$

Calculating the Levenshtein distance from this recursion directly takes exponential time (Moisan et al., 2014). However, a dynamic programming algorithm can compute the matrix $[lev]$ by computing its values row by row in $O(|W_1||W_2|)$ time, considering that $lev_{i,j}$ represents the Levenshtein distance between $W_1(w_{11}, \ldots, w_{1i})$ and $W_2(w_{21}, \ldots, w_{2i})$. Hence, to obtain the complete Levenshtein distance between these two sequences, one needs to compute $lev_{|W_1|,|W_2|}$. This matrix allows to use the above recursion without having to compute the values $lev_{i,j}$ multiple times.

The other types of edit distance allow different sets of string operations. For instance, the Damerau-Levenshtein distance brings an important contribution by adding the transposition operation of two adjacent characters (Damerau, 1964).

As for the Levenshtein distance, the Damerau-Levenshtein distance can be computed recursively as follows:

Considering $dl_{i,j}$ the Damerau-Levenshtein distance between the subsequence formed with the i first characters of a sequence W_1 and the subsequence formed with the j first characters of a sequence W_2,

$$dl_{i,j} = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} dl_{i-1,\,j} + 1 \\ d_{i,\,j-1} + 1 \\ dl_{i-1,\,j-1} + I(w_{1i} \neq w_{2j}) \\ dl_{i-2,\,j-2} + 1 \end{cases} & \text{if } i,j > 1^{\wedge} \\ & w_{1i} = w_{2j-1}{}^{\wedge} \\ & w_{1i-1} = w_{2j} \\ \min \begin{cases} dl_{i-1,\,j} + 1 \\ dl_{i,\,j-1} + 1 \\ dl_{i-1,\,j-1} + I(w_{1i} \neq w_{2j}) \end{cases} & \text{otherwise} \end{cases}$$

The Damerau-Levenshtein distance can also be calculated in O $(|W_1||W_2|)$ time, using dynamic programming algorithm by filing in the matrix $[dl]$ row by row.

From this distance, also called Full Damerau-Levenshtein distance, derives the Restricted Damerau-Levenshtein distance.

The difference between these three distances is as follows:

- **Levenshtein distance:** it provides the minimal number of insertions, deletions and replacements needed for transforming string W_1 into string W_2.
- **(Full) Damerau-Levenshtein distance:** it is like the Levenshtein distance. Moreover, it allows transposition of adjacent symbols.
- **Restricted Damerau-Levenshtein distance:** it is like (full) Damerau-Levenshtein distance, but each substring may only be edited once.

These different forms of the Levenshtein's edit distance are used to develop the following algorithms:

2.3.1. Norvig

The Norvig's algorithm is using the full Damerau-Levenshtein edit distance. It could be modified to use the Levenshtein distance. The idea is if we artificially generate all terms within maximum edit distance from the misspelled term, then the correct term must be among them. We have to look all of them up in the dictionary until we have a match. So all possible combinations of the 4 spelling error types (insert, delete, replace and adjacent switch) are generated. This is quite expensive with e.g. 114.324 candidate term generated for a word of length = 9 and edit distance = 2 (Ahamed et al., 2020).

2.3.2. BK-Tree

Burkhard Keller Tree, known as the BK-Tree, is a Tree-Based Data Structure that is used to find the near-matches to a String Query. The BK-Trees were first proposed in the Walter A. Burkhard and Robert M. Keller's paper, entitled "Some approaches to best match file searching" (Burkhard and Keller, 1973). Since it has been used as an algorithm for performing Spell Check.

The BK-Tree algorithm uses two basic concepts to reduce the time complexity of the string-matching operations. These concepts are the Levenshtein distance and the Triangular Inequality. BK-Trees are also utilized in implementing the auto-complete feature in many software and web applications and is regarded as more efficient.

In a Burkhard Keller Tree, any arbitrary node can be chosen as the Root Node, which can have zero or any number of sub-trees, and each node consists of the individual words in the dictionary.

Furthermore, the total number of nodes is equal to the number of the words that are being inserted into the dictionary. Each edge that connects the nodes possess an integer value that is corresponding to the Levenshtein distance, which is based upon the following three criteria:

- If the distance between X and Y is zero then X is equal to Y.
- The distance from X to Y is equal to the distance between Y to X.
- Triangle Inequality: Path from one point to another must be no longer than any path that goes through any other intermediate path.

2.3.3. LinSpell

Linear search spelling correction algorithm, designated LinSpell, is using the Restricted Damerau-Levenshtein edit distance. It could be modified to use the Levenshtein distance or the full Damerau-Levenshtein distance.

The LinSpell spelling correction algorithm does not require edit candidate generation or specialized data structures like BK-tree or Norvig's algorithm. In most cases LinSpell is faster and requires less memory compared to BK-tree or Norvig's algorithm. LinSpell is language and character set independent. The word frequency list was created by intersecting the two lists mentioned below. By reciprocally filtering, only those words which appear in both lists are used. Additional filters were applied and the resulting list truncated to ≈ 80,000 most frequent words.

This is basically a linear scan through the wordlist and calculating the edit distance for every single word (with a few tweaks). It was intended as the baseline measure for the benchmark. Surprisingly enough and despite its O(n) characteristics it turned out to be faster than both BK-tree and Norvig's algorithm.

2.3.4. SymSpell

The Symmetric Delete spelling correction algorithm, known as SymSpell, improves the complexity of edit candidate generation and dictionary lookup for a given Damerau-Levenshtein distance. It is language independent and six orders of magnitude faster than the standard approach with deletes, transposes, replaces and inserts (Murugan et al., 2020).

Replaces and inserts are expensive and language dependent: e.g. Chinese has 70,000 Unicode Han characters! The speed comes from the inexpensive delete-only edit candidate generation and the pre-calculation.

An average 5 letter word has about 3 million possible spelling errors within a maximum edit distance of 3, but SymSpell needs to generate only 25 deletes to cover them all, both at pre-calculation and at lookup time.

2.3.5. N-gram

Error detection by n-gram allows to know the position of the error in the word. This information facilitates the correction of the error. The correction method proposed by Sundby (2009) relies on this position to generate a suggestion list for the erroneous word.

His algorithm is based on the four editing operations (Damerau, 1964), namely insertion, deletion, substitution and transposition, to search for correct forms that may correspond to the misspelled word. The sub-algorithms of these operations are described below:

- **Insertion:** It is an algorithm that corrects erroneous words with a missing character. The principle of this algorithm consists in inserting a letter of the alphabet at the position where the error occurred and checking if the resulting word is correct before adding it to the list of candidate words. The process is repeated for all the letters of the alphabet.
- **Deletion:** It is an algorithm that removes characters from a wrong word. The algorithm removes a character from the word and checks if the formed word is correct before adding it to the suggestion list. The process is repeated for all the letters of the word.
- **Substitution:** The substitution algorithm replaces each letter of the word, one after the other, with a letter of the alphabet and checks if the resulting word is correct before adding it to the suggestion list. The same process is repeated for all the letters of the alphabet.
- **Transposition:** The algorithm changes the position of one character of the word by putting it in all the other positions and each time it checks if the formed word is correct before adding it to the suggestion list. The process is repeated for all the letters of the word.

3. Amazigh language characteristics

Amazigh is one of the oldest languages in Africa with a history dating back more than 40 centuries (Hachid, 2000). Amazigh populations are geographically dispersed over several nation-states in the territory commonly known as Tamazgha or the Amazigh Homeland. Tamazgha covers an area of nearly 5,000,000 km^2 extending from east to west from the Egyptian-Libyan border to the Atlantic Ocean and from north to south of the southern shore of the Mediterranean to the Stromboli massif in Burkina Faso.

In Morocco, Amazigh is divided into three major dialectal regions that cover all mountainous areas: in the northeast, the Rif with the Tarifit dialect; in the center, the Middle Atlas and part of the High Atlas with the Tamazight dialect; in the south and southwest, the High Atlas, the Anti-Atlas and Souss, the Chleuh domain with the Tashelhit dialect.

The Amazigh language was introduced to Moroccan schools in September 2003 after Ajdir's Royal Speech of October 17, 2001, and the creation of the Royal Institute of Amazigh Culture (IRCAM) that is in charge of the Amazigh language and culture promotion. In July 2011, it changed its status from "national language" to "official language" recognized beside the Arabic language in the Moroccan Constitution as the official languages of the country.

3.1. Amazigh alphabet

Tifinagh is the writing system of the Amazigh language. It originates from the old Libyan and Saharan alphabet. It was used by the populations of North Africa, the Sahel and the Canary Islands. This alphabet has undergone changes in order to provide this language with an alphabetic system that is adequate and usable for all current Amazigh languages. Thus in 2003, starting from an ancient as well as modern and contemporary heritage, IRCAM developed an alphabet system under the name of Tifinagh-IRCAM (Ameur et al., 2004). It is oriented horizontally from left to right and is composed of thirty-three letters:

- 27 consonants including: labials (ⵀ, ⵖ, ⴼ), dentals (ⵜ, ⵟ, ⴷ, ⴹ, ⵏ, ⵔ, ⵕ, ⵍ), alveolars (ⵣ, ⵙ, ⵥ, ⵚ), palatales (ⵛ, ⵊ), velars (ⴽ, ⵅ), labiovelars (ⴽⵯ, ⵅⵯ), the uvulars (ⵇ, ⵃ, ⵄ), pharyngeals (ⵄ, ⵃ) and laryngeal (ⴰ);
- 2 semi-consonants: ⵢ and ⵡ;
- 4 vowels: three full vowels ⴰ, ⵉ, ⵓ and the neutral vowel (called schwa) ⴻ which has a rather special status in Amazigh phonology.

3.2. Amazigh morphology

In contrast to English, Amazigh is a highly inflected language, which makes the task of automatic correction more complex. It

has three main syntactic categories: noun, verb, and particle (Ameur et al., 2004).

3.2.1. Noun

Nouns distinguish two genders: masculine and feminine; two numbers: singular and plural; and two cases: expressed in the nominal prefix.

- The feminine is used for female persons and animals as well as for small(er) objects. The productive derivation masculine feminine is quite regular morphologically, using noun prefixes and suffixes.
- The plural has three forms: the external plural consisting in changing the initial vowel, and adding suffixes; the broken plural involving changes in the internal noun vowels; and the mixed plural that combines the rules of the two former plurals.
- The annexed (relative) case is used after most prepositions and after numerals, as well as when the lexical subject follows the verb; while, the free (absolute) case is used in all other contexts.

3.2.2. Verb

The verb has two forms: basic and derived forms. The basic form is composed of a root and a radical, while the derived one is based on a basic form in addition to some prefix morphemes. Whether basic or derived, the verb is conjugated in four aspects: aorist, imperfective, perfect, and negative perfect. Person, gender, and number of the subject are expressed by affixes to the verb. Depending on the mood, these affixes are classed into three sets: indicative, imperative, and participial.

3.2.3. Particles

Particles contain pronouns; conjunctions; prepositions; aspectual, orientation and negative particles; adverbs; and subordinates. Generally, particles are uninflected word. However, in Amazigh language, some of these particles are flectional, such as the possessive and demonstrative pronouns.

4. The Amazigh spell checker

With the aim to construct an Amazigh spell checker, we focused, in a first stage, to correct isolated word error spelling. Thus, we proposed a N-gram based system (Chaabi et al., 2019). In this paper, we try to improve the system performance by adding the Levenshtein algorithm.

4.1. The proposed system

The proposed spell checker works in several steps: error detection, selection of possible corrections, filtering possible corrections and presenting a list of candidate corrections to the user (Fig. 1).

Our system is a sequential combination of two approaches including the Damerau-Levenshtein distance and N-gram.

By applying the Damerau-Levenshtein algorithm, we obtain a set of solutions $\{w_1, w_2, \ldots, w_3\}$, where the distance (D) between w_{error} and w_i is given by:

$$D(w_{error}, w_i) = d_i$$

The Damerau-levenshtein algorithm uses a matrix (N + 1) * (P + 1), with N and P are respectively the lengths of the words W_1 and W_2. The distance between the words W_1 and W_2 is defined by:

$$D_{w_1,w_2}(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ min\begin{cases} dl_{i-1,\,j} + 1 \\ d_{i,\,j-1} + 1 \\ dl_{i-1,\,j-1} + I(w_{1i} \neq w_{2j}) \\ dl_{i-2,\,j-2} + 1 \end{cases} & \text{if } i,j > 1^{\wedge} \; w_{1i} = w_{2j-1}{}^{\wedge} \; w_{1i-1} = w_{2j} \\ min\begin{cases} dl_{i-1,\,j} + 1 \\ dl_{i,\,j-1} + 1 \\ dl_{i-1,\,j-1} + I(w_{1i} \neq w_{2j}) \end{cases} & \text{otherwise} \end{cases}$$

where $1_{(w_i \neq w_j)}$ is the indicator function equal to 0 when $w_i \neq w_j$ and equal to 1 otherwise, and $D_{w_1,w_2}(i,j)$ is the distance between the first i characters of w_1 and the first j characters of w_2.

According to Bayes' formula, we get the equation that we try to maximize:

$$P(c|w) = \frac{P(w|c) * P(c)}{P(w)}$$

$$\text{argmax } P(c|w) = \frac{P(w|c) * P(c)}{P(w)}$$

with:

- w is the word typed by the user;
- c is a possible correction of this word;
- P(c|w) is the probability that the writing word w is an observation of the correct word c;
- P(w|c) is the probability of having made the mistake w by trying to type the word c, this is the error model;

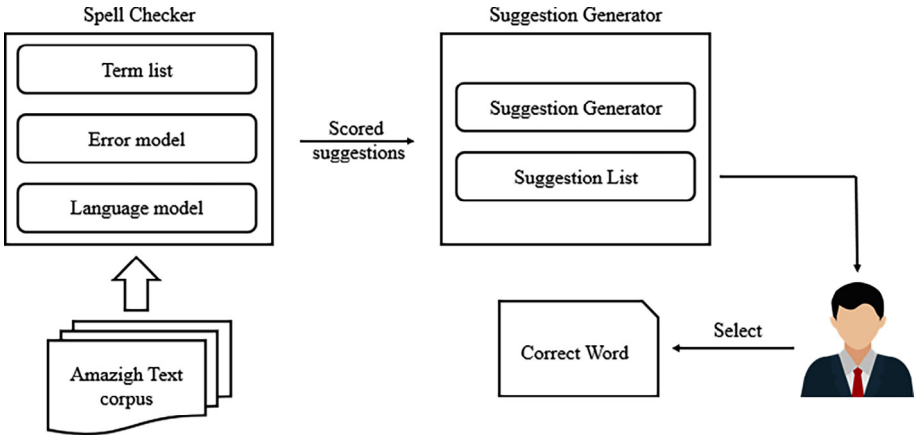


Fig. 1. Amazigh Spell Checker Framework.

- P(c) is the probability that the user wants to type the word c, this is the language model.

The denominator P(w) is only a normalizing factor that is constant over the set C of corrections. Therefore, we can choose the convention P(w) = 1.

Then, the equation becomes as follows:

$$\underset{c\in C}{\operatorname{argmax}} P(w|c) * P(c)$$

A typing error is interpreted as an elementary operation, to which a certain probability of error (Pe) is attributed. The editing distance d between two words is then calculated, i.e. the minimum number of elementary operations to pass from one word to another.

We note that a character or a space symbol is marked C; a word or a sequence of characters is a finite sequence of C, referenced as $Sc = U_{k=1}^{\infty} C^k$.

It is possible to define the editing distance d as follows:

$$d: Sc * Sc \rightarrow R^+$$

$$(w_1, w_2) \rightarrow \min_{O\ operation\ sequence} d(w_1, w_2, O)$$

with w_1, w_2 are words.

Finally, the following error model is applied:

$$P(w|c) = P_e = \begin{cases} \frac{Removing[w_{i-1},w_i]+1}{Numcharac(w_{i-1},w_i)+|K|} \\ \frac{Insertion[w_{i-1},x_i]+1}{Numcharac(w_{i-1})+|K|} \\ \frac{Substitution[x_i,w_i]+1}{Numcharac(w_i)+|K|} \\ \frac{Switching[w_i,w_{i+1}]+1}{Numcharac(w_i,w_{i+1})+|K|} \end{cases}$$

Where:

- K is the number of characters, or sequences of two successive characters present in the word correctly spelled in the dataset.
- Numcharac (w_i) corresponds to the number of characters w_i found in a dataset.

If there are several n errors, the probability of each of the error is calculated as follows:

$$P(w|c) = P(error_1) * P(error_2) * \ldots\ldots * P(error_n)$$

$$P(w|c) = P_e^{d(w,c)}$$

To improve the scheduling of these solutions, we introduce the n-grams model language in the Levenshtein distance that is characterized in this case by the probability $P(\frac{w_i}{c})$ with c is a possible correction of this word.

The weighted Levenshtein distance is defined by this probability by:

$$D(W_{error}, W_i) = \begin{cases} \frac{d_i}{P(w_i|c)} & if\ P(w_i|c) \neq 0 \\ \min_{w,u\in Y} \frac{d_i}{P(w|u)} & otherwise \end{cases}$$

The best corrections noted w_c are given by the following equation:

$$w_c = \min_{w_i \in S} D(W_{error}, W_i)$$

4.2. The system interface

The proposed system is a Word add-on it accepts as input a text written in Tifinagh graphics. It starts by the segmentation of a text into a list of tokens. Where a token is a sequence of characters delimited by a blank (space, tabulation, newline) or a punctuation separator (“.”, “?”, etc.). For each token, the misspelling detector is used to tell us whether the token is in the dictionary or not. The corrector returns a list of suggestions including candidate corrections related to the word assumed to be incorrect or misspelled.

This list, if it is not empty, is stored according to the edit distance between the misspelled word and the possible corrections, in addition to their frequency in the corpus. The list is presented to the user so that s/he can choose the right correction or to enter it by her/himself if it is not in the list, or to indicate that the word is correct. The last case means that the detector does not recognize the word due to its absence from the dictionary. The correct word obtained after the user intervention will be inserted into the result text and the program proceeds to parse the rest of the text until the end.

When a word is not recognized, the user can add it to her/his personal dictionary. Thus, the spell checker memorizes the word, and take it into consideration next time.

To illustrate this process, we take the first misspelled word (‘ⵜⴰⵎⴰⵣⵉⵖⵜ’ [tamazict]), shown in Fig. 2. Since the system does not find this word in its dictionary, it is considered as an error. So, the system, first, generates, for this word, a set of candidate corrections

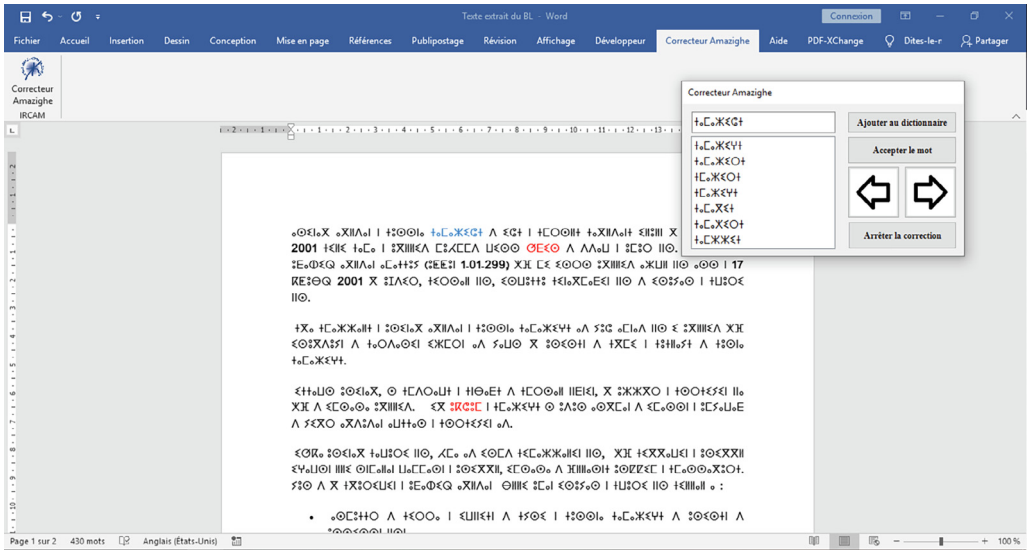


Fig. 2. The Amazigh language Spell checker.

Table 1
List of candidate corrections for the misspelled word 'ⵜⴰⵎⴰⵣⵉⵛⵜ' [tamazict].

Rank	Candidate corrections	Edit distance	Frequency	Rank	Candidate corrections	Edit distance	Frequency
8	ⵜⴰⵎⵏⵣⵉ�well [tamnziwt]	2	1	7	ⵜⴰⵎⵣⵣⵉⵜ [tamzzit]	2	2
5	ⵜⴰⵎⴰⴳⵉⵜ [tamagit]	2	28	**1**	**ⵜⴰⵎⴰⵣⵉⵖⵜ** [tamaziɣt]	1	611
4	ⵜⵎⴰⵣⵉⵖⵜ [tmaziɣt]	2	379	2	ⵜⴰⵎⴰⵣⵔⵜ [tamazirt]	1	192
3	ⵜⵎⴰⵣⵔⵜ [tmazirt]	2	514	6	ⵜⴰⵎⴰⵅⵔⵜ [tamaxirt]	2	3

(see Table 1). Then, it returns, to the user, this set sorted according to the lowest edit distance and the highest occurrence frequency of the candidates. If the list of suggestions is large, the system shows only the first seven (ⵜⴰⵎⴰⵣⵉⵖⵜ [tamaziɣt], ⵜⴰⵎⴰⵣⵔⵜ [tamazirt], ⵜⵎⴰⵣⵔⵜ [tmazirt], ⵜⵎⴰⵣⵉⵖⵜ [tmaziɣt], ⵜⴰⵎⴰⴳⵉⵜ [tamagit], ⵜⴰⵎⴰⵅⵔⵜ [tamaxirt], ⵜⴰⵎⵣⵣⵉⵜ [tamzzit]). In this example, the system puts at the top of the list the word 'ⵜⴰⵎⴰⵣⵉⵖⵜ' [tamaziɣt], which means Amazigh, that coincides with the right correction.

5. Benchmark and results

To evaluate the performance of our system, based on the combination of the Norvig's algorithm and the N-gram approach, in the context of isolated word spelling correction, we start by comparing it with five other approaches (Norvig's, BK-Tree, SymSpell, LinSpell and N-gram). To this end, we have built a corpus including 520.000 words in total and 167.857 distinct words.

5.1. Collection of linguistic data

The dictionary is placed at the core of our approach because it is used by the spell checker as a resource that must be maintained and enriched. Thus, we have collected textual resources in order to initialize our spell checker dictionary.

In view of the linguistic resources at our disposal, this dictionary is composed of a list of 361.816 words extracted from a textual corpus and of 158.184 words extracted from a set of Amazigh digital and paper lexicons. The paper ones include the legal lexicon (Amouzay et al., 2018) and the scholar lexicon (Agnaou et al., 2008). While, the digital lexicons consist of the DGLAI (Ataa Allah et al., 2019) and the AmaMorph (Ataa Allah et al., 2018). The first one (DGLAI, Dictionnaire General de la Langue Amazighe Informatisé) is the electronic version of the General Dictionary of Amazigh Language (Ameur et al., 2017). It contains 14.931 entries in Tifinaghe characters. The second is the Amazigh Morpholigical lexicon, providing inflected forms. It is composed of 15.523 nominal and verbal entries, in addition to 127.730 inflected forms. The used corpus is a raw textual collection, representing the three Moroccan Amazigh varieties. It has been selected from published data covering various topics and dealing with different literary genres: novels, poems, stories and newspaper articles. It has been proofread and linguistically reviewed by specialists according to the spelling rules.

The set used in this work contains 520.000 words in total and 167.857 distinct words. 70% of this set has been used for training our models; while, 30% of the data has been used for the test. Besides that, we have prepared a test text to measures the system detection performance. This latter is a manually proofread, containing 685 words and 55 misspellings. Furthermore, we have collected a list of 1010 erroneous words accompanied by their correction, in order to evaluate the accuracy of our system.

5.2. Comparison and results

Our main objective is to develop a robust spell checker able to correct the majority of Amazigh misspellings. Thus, we have compared the proposed approach with the five most used approaches in the literature (Norvig's, BK-Tree, SymSpell, LinSpell and N-gram). With the aim to benchmark these approaches, we have performed three tests for each approach. The first test consists in the calculation of the search time required to detect errors and to present correct suggestions to the user. The second one measures the lexical and the error coverage. The third test provides the spell check approach exactness and completeness.

The benchmark has been limited to maximum edit distance up to 4, because for natural language search an edit distance of 4 is already on the border of what's reasonable (Yujian and Bo, 2007), as the number of false positive grows exponentially with the maximum edit distance, decreasing precision, while improving recall only slightly.

5.2.1. Time reduction

The reduction of the time needed to detect and correct spelling mistakes in a text is seen from the perspective of optimizing the number of accesses to the vocabulary, which has long remained a challenge.

The total search time is measured for 1000 words with random spelling errors from our dictionary (167.857 words). Three maximum edit distances (2, 3, 4) are used, on a computer with a 2.40 GHz dual-core Intel Core i5-4210 processor and 8 GB of RAM.

According to Fig. 3, we remarked that N-gram is faster than our approach, SymSpell, LinSpell, BK-Tree and Norvig's approaches. N-gram lookup time grows moderately with the dictionary size and the maximum edit distance. It outperforms all the other approaches in all cases, often by several orders of magnitude. That's why N-gram speed has its origins in the conception of our approach.

5.2.2. Detection error evaluation

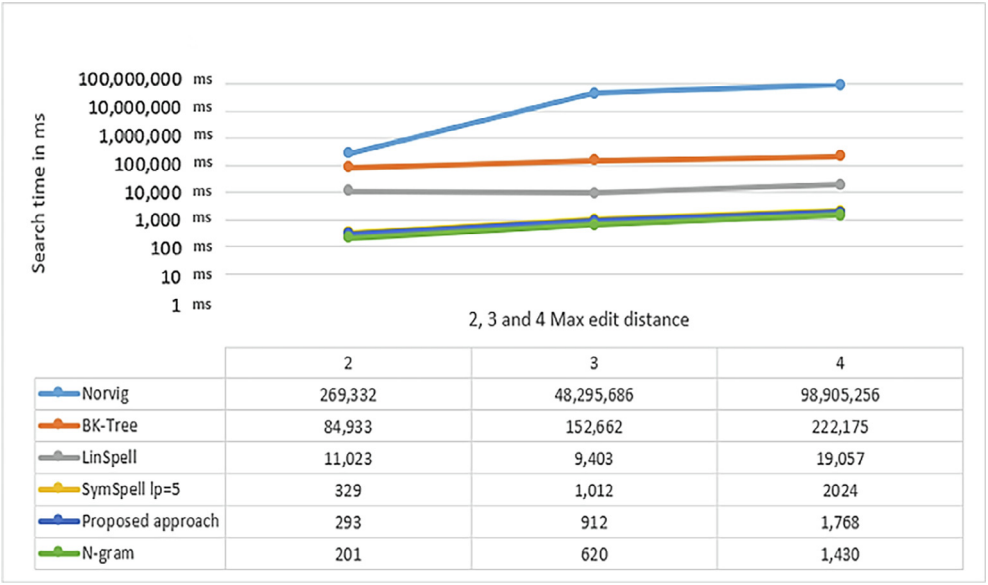
To measure the effect of the detection part of our approach, we used the test text and the following measures:

- True positive (TP): correct word which is recognized as correct word by the spell checker.
- False positive (FP): incorrect word which is recognized as correct word by the spell checker.
- False negative (FN): correct word which is recognized as incorrect word by the spell checker.
- True negative (TN): incorrect word which is recognized as incorrect word by the spell checker.

To know about the capacity to detect correct words, we used the *Lexical Recall* (R_c) and *Precision* (P_c). Whereas, to look for the ability to found misspelled ones, we calculated the *Error Recall* (R_i) and *Precision* (P_i). 'c' and 'i' are specifying respectively correct and incorrect.

$$R_c = \frac{T_P}{T_P + F_N}, P_c = \frac{T_P}{T_P + F_P}$$

$$R_i = \frac{T_N}{T_N + F_P}, P_i = \frac{T_N}{T_N + F_N}$$



	2	3	4
Norvig	269,332	48,295,686	98,905,256
BK-Tree	84,933	152,662	222,175
LinSpell	11,023	9,403	19,057
SymSpell lp=5	329	1,012	2024
Proposed approach	293	912	1,768
N-gram	201	620	1,430

Fig. 3. Search time for 1000 words in 167 857 dictionary words.

Table 2
Performance measures for the detection step.

	Norvig	BK-Tree	LinSpell	SymSpell	Proposed approach	N-gram
Lexical Recall	97.52%	97.52%	97.52%	97.52%	97.52%	97.52%
Lexical Precision	100%	100%	100%	100%	100%	100%
Lexical F-measure	98.74%	98.74%	98.74%	98.74%	98.74%	98.74%
Error Recall	100%	100%	100%	100%	100%	100%
Error Precision	76.38%	76.38%	76.38%	76.38%	76.38%	76.38%
Error F-measure	86.62%	86.62%	86.62%	86.62%	86.62%	86.62%

Table 3
Correction list for the Amazigh spell checker.

	Norvig	BK-Tree	LinSpell	SymSpell	Proposed approach	N-gram
Correct word as first suggestion	62	61	61	60	**66**	57
Correct word among first five suggestions	87	86	86	85	**87**	81
Correct word further down in list	9	10	10	11	**9**	13
Correct word not in list	4	4	4	4	4	5

F-measure gives the mean of recall and precision values. Hence, the F-measure equations to detect correct words and misspelled ones are as follows:

$$F_c = \frac{2 * R_c * P_c}{R_c + P_c},\ F_i = \frac{2 * R_i * P_i}{R_i + P_i}$$

From the results of Table 2, it is clear that all the approaches have the same performance to detect errors, since they use the same lexicon.

While analyzing the errors made by the spell checker during the detection step, we remarked that the test text includes proper nouns in addition to some morphological forms not included into the spell checker dictionary.

5.2.3. Correction accuracy

Generally, for an interactive spell checker, a misspelling is treated successfully if the intended target word appears in the suggestion's list. To measure the accuracy of the correction process, 100 words were selected randomly from the erroneous word's list. For each of the six approaches the found suggestions have been compared to ensure completeness of results. It is assumed that the higher the target word's position in the list, the more helpful (and successful) the correction process.

From the results illustrated in Table 3, it is clear that our approach performs better. The target word appears among the first five suggestions much more frequently (87%) that it occurs further down in the list (9%).

In summary, the Levenshtein Distance-based spell checkers give better and precise results in accordance with N-gram. However, N-gram is much faster, especially with large dictionaries. Thus, proposing an approach based on the combination of N-gram and the Levenshtein Distances, specially the Damerau-Levenshtein can handle large dictionaries much more efficiently.

6. Conclusion

The Amazigh language lacks spelling correction tools. Considering the importance of this kind of tools, mainly for not native speakers, this work has proposed a new system to overcome this lack. The proposed spell checker is based on the full Damerau-Levenshtein and N-gram approaches. The system performance has been evaluated on misspelling detection and correction

separately. On the task of misspelling detection, our system achieved F-measure ranged from 86.62% to 98.74%, similarly to the other approaches. While, the correction accuracy has proven satisfactory results compared to other approaches.

In our future work, we look for adding some morpholexical Amazigh language rules in the correction step, and enriching the spell checker dictionary. Furthermore, since at the present stage, the spell checker corrects only spelling errors, we want to focus our attention on other kinds of errors. Thus, we would like to take into account the morpho-syntactic and syntaxico-semantic knowledge, in order to move from the detection and correction of isolated words to global sentences.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Fatima Agnaou, Abdellah Bouzandag, M'hamed El Baghdadi, L'houssaine El Gholb, Abdeslam Khalafi, Kamal Ouqua, and Mustapha Sghir. 2008. Le lexique scholaire. The Royal Institute of Amazigh Culture.

Ahamed, I., Jaan, M., Tasnim, Z., Karim, T., Reza, S.SA., Hossain, D.A., 2020. Spell corrector for Bangla language using Norvig's Algorithm and Jaro-Winkler distance. Bull. Electr. Eng. Inf. 9 (6).

Aho, A.V., Corasick, M.J., 1975. Efficient string matching: an aid to bibliographic search. Commun. ACM 18 (6), 333–340. https://doi.org/10.1145/360825.360855.

Fadoua Ataa Allah, Jamal Frain, and Siham Boulaknadel. 2019. Dictionnaire Général de la Langue Amazighe Informatisé. Proceeding of the international conference on la Diversité Linguistique et TAL. December 16-17, 2019, Oujda, Morocco.

Ameur, M., Ansar, K., Boumalk, A., El Azrak, N., Laabdelaoui, R., 2017. Dictionnaire général de la langue amazighe. The Royal Institute of Amazigh Culture.

Meftaha Ameur, Aicha Bouhjar, Fatima Boukhris, Ahmed Boukous, Abdallah Boumalk, Mohamed Elmedlaoui, El Mehdi Iazzi, and Hamid Souifi. 2004. Initiation à la langue amazighe. The Royal Institute of Amazigh Culture.

Amouzay, L., Bouhjar, A., Boumisser, A., Ansar, K., 2018. Lexique Juridique. The Royal Institute of Amazigh Culture.

Fadoua Ataa Allah, and Siham Boulaknadel. 2018. Morpho-Lexicon for Standard Moroccan Amazigh. in MATEC Web of Conferences. Vol. 210.

Bent, S.W., Sleator, D.D., Tarjan, R.E., 1985. Biased search trees. SIAM J. Comput. 14 (3), 545–568. https://doi.org/10.1137/0214041.

Burkhard, W.A., Keller, R.M., 1973. Some approaches to best-match file searching. Commun. ACM 16 (4), 230–236. https://doi.org/10.1145/362003.362025.

Youness Chaabi, Mohamed Outahajala, and Fadoua Ataa Allah. 2019. Amazigh Spell Checker based on Naïve Bayes and N-Gram. Proceeding of Human Language Processing of Resource Scarce Languages Workshop, International Conference on Advanced Technologies and Humanitarian Sciences, July 25-26, 2019. Rabat, Morocco.

Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Commun. ACM 7 (3), 171–176. https://doi.org/10.1145/363958.363994.

Hachid, M., 2000. Les premiers berbères, Entre Méditérranée, Tassili et Nili. Edisud-Ina-Yas, Aix-en-Provence-Alger.

Hládek, D., Staš, J., Pleva, M., 2020. Survey of Automatic spelling correction. Electronics 9 (10), 1670. https://doi.org/10.3390/electronics9101670.

Kukich, K., 1992. Techniques for automatically correcting words in text. ACM Computing Surveys (CSUR) 24 (4), 377–439. https://doi.org/10.1145/146370.146380.

Levenshtein, V., 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (8) https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf.

Thierry Moisan, Claude-Guy Quimper, Jonathan Gaudreault, and Sbastien Michaud. 2014. Re-planning with minimal perturbation. 10th Conférence Francophone de Modélisation, Optimisation et Simulation, November 2014, Nancy, France.

Selvakumar Murugan, Tamil A. Bakthavatchalam, Malaikannan Sankarasubbu. 2020. SymSpell and LSTM based Spell-Checkers for Tamil.

Ndiaye, M., Faltin, A.V., 2003. A spell checker tailored to language learners. Int. J. Phytorem. 21 (1), 117–134. https://doi.org/10.1076/call.16.2.213.15881.

Peterson, J.L., 1980. Computer programs for detecting and correcting spelling errors. Commun. ACM 23 (12), 676–687. https://doi.org/10.1145/359038.359041.

Pollock, J.J., Zamora, A., 1984. Automatic spelling correction in scientific and scholarly text. Commun. ACM 27 (4), 358–368. https://doi.org/10.1145/358027.358048.

Salifou, L., Naroua, H., 2014. Design of a Spell Corrector for Hausa Language. Int. J. Comput Linguistics (IJCL) 5 (2), 14–26 https://www.cscjournals.org/manuscript/Journals/IJCL/Volume5/Issue2/IJCL-56.pdf.

Shannon, C.E., 1951. Prediction and entropy of printed english. Bell Syst. Tech. J. 30 (1), 50–64. https://doi.org/10.1002/j.1538-7305.1951.tb01366.x.

Sundby, D., 2009. Spelling correction using N-grams. Technical notes 3 http://fileadmin.cs.lth.se/cs/Education/EDA171/Reports/2009/david.pdf.

Ten Hacken, P., Tschichold, C., 2001. Word manager and CALL: structured access to the lexicon as a tool for enriching learners vocabulary. ReCALL 13 (1), 121–131. https://doi.org/10.1017/S0958344001001112.

Séverine Vienney, 2004. Correction automatique bilan et perspectives, Besançon.

Yujian, L.i., Bo, L., 2007. A normalized levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell. 29 (6), 1091–1095. https://doi.org/10.1109/TPAMI.2007.1078.

Zourou, K., 2009. Computer Supported Collaborative Learning (CSCL) et Apprentissage Des Langues Assisté Par Ordinateur (Alao): Un Dialogue à Ne Pas Manquer - Réflexions Autour Du Colloque Mondial CSCL 09. Alsic 12. https://doi.org/10.4000/alsic.1273.