For an Independent Spell-Checking System from the Arabic Language Vocabulary

Bakkali Hamza
Telecom and Embedded Systems Team,
SIME Lab ENSIAS, University of Mohammed V Souissi
Rabat- Morocco

Gueddah Hicham
Telecom and Embedded Systems Team, SIME Lab
ENSIAS,
University of Mohammed V Souissi
Rabat- Morocco

Yousfi Abdellah
Eradiass Team
Faculty of Juridical, Economic
and Social Sciences University of
Mohammed V-Souissi,
Rabat- Morocco

Belkasmi Mostafa
Telecom and Embedded Systems Team, SIME Lab
ENSIAS,
University of Mohammed V Souissi
Rabat- Morocco

Abstract—**In this paper, we propose a new approach for spell-checking errors committed in Arabic language.**

This approach is almost independent of the used dictionary, of the fact that we introduced the concept of morphological analysis in the process of spell-checking. Hence, our new system uses a stems dictionary of reduced size rather than exploiting a large dictionary not covering the all Arabic words.

The obtained results are highly positive and satisfactory; this has allowed us to appreciate the validity of our concept and shows the importance of our new approach.

Keywords—Arabic language; Lexicon; Misspelled word; Error model; Spell-checking; Edit distance; Morphological analysis; Prefix; Stem, Suffix

I. Introduction

Automatic correction of spelling errors is one of the most important areas in the field of Natural Language Processing (NLP) and it has been a subject of many researches since the 60's [1]. Spell-checking consists in suggesting the closest corrections for a misspelled word, this implies the development of error models and methods allowing the scheduling of plausible corrections and the disposal of a representative lexicon for a given language in order to compare.

Early researches consist in founding a kind of error modeling. Since, appears Damerau's [2] definition which consider a spelling error as a simple combination of elementary edition operations of insertion, deletion, transposition and substitution. Based on Damerau's definition, Levenshtein [3] will define his distance (Levenshtein distance) which is characterized by three of elementary edition operations: insertion, deletion and permutation. Another modeling proposed by Pollock and Zamora [4] consists in associating for each word in the dictionary its alpha-code (consonants of the word), hence the need of having two dictionaries: one for the words and the other for their alpha-codes, and therefore the correction will be done by comparing alpha-codes with the misspelled word. This method is efficient for permutation errors cases.

We also find among these studies: the decomposition method based on the concept of N-gram language model which is based on decomposing a misspelled word to di-trigrams and compare them to the dictionary di-trigrams in order to produce a similarity index to designate the nearest words to the misspelled word [5]. In 1996, Oflaser [6] defined a new approach called tolerant recognition of spelling errors by using the concept of finite state automaton and a distance called cut-off edit distance. Using this approach, the correction of a misspelled word is done by browsing the dictionary automaton and by calculating the cut-off edit distance for each transition, without exceeding a threshold previously defined in the algorithm. Gueddah, Yousfi and Belkasmi [7] proposed a typical and efficient variant of edit distance by integrating frequency editing errors matrices [8] in the Levenshtein algorithm in order to improve the scheduling of the solutions of an erroneous word in Arabic documents.

Generally in natural languages, and especially in Arabic, existing spelling automatic correction systems do not cover all the misspelled words. Spelling correction was always related to the disposition of a given lexicon covering the totality of misspelled words.

Several studies have been made towards the development of dictionaries adapted for spell-checking systems. Among these studies we cite in particular: the Ayaspell[1] project that aims to generate dictionaries, for example the Arabic lexicon Hunspell-ar Version 3.2 that contains more than 300000 Arabic words designed for free office suite applications of Open Office (writer) and Mozilla Firefox 3, Thunderbird and Google Chrome incorporating the spell-checker Hunspell[2] (originally designed for the Hungarian language).

[1] http://ayaspell.sourceforge.net
[2] http://hunspell.sourceforge.net

Despite linguistic resources available to the Arabic language, we note that we do not have yet a robust spell-checker capable of covering all the spelling errors committed. And that raised a major challenge for spell-checking [9].

In order to overcome this limitation raised on spell-checking for the Arabic language, we propose in this paper a new approach which aims to introduce the concept of morphological analysis in the process of spell-checking.

In reality, there are few works that deal with morphological analysis in spell-checking process. Among these works are cited, specially:

- Emirkanian [10] have developed an expert system in the fields of spell-checking, morphological analysis and syntactic analysis for the French language by developing a morpho-syntactic analyzer capable of detecting and correcting spelling, morphological and syntactic errors frequently committed in French documents entries. This system is based on the integration of knowledge-based rules of French for various levels: orthographic (radical's dictionary), morphological (analyzer, suffixes dictionary) and syntactic (syntactic tree, substitution rules, completion rules). This system uses a metrical distance [11] to limit the search space and define the substitution rules.
- In another approach proposed by Bowden and Kiraz [12], they have presented a morpho-graphemic model for spelling and morphological errors correction based on McCarthy morphological analyzer [13]. The advantage of this model is the way it's combines lexical analysis with morphological analysis to determine the correction possibilities.
- Another recent study is presented by Shaalan and his team [14]. This project present a spell-checking system for Arabic language that aims to explore in a first step a huge dictionary of few millions words (13 millions) generated by the AraComLex3 finite state transducer with only 9 millions of valid lexical forms filtered by the AraMorph of Buckwalter morphological analyzer [15]. These ones have explored this dictionary to propose a generic spell-checking model for Arabic by using finite state automaton technology [6] and a specific metrical distance [3] combined with a noisy channel model and also with knowledge-based rules to assign weights to the suggested corrections in order to refine the best solutions.

The major inconvenient of all the spell-checking systems resides in the limitation of the used dictionaries, because they do not cover the totality of the words of a given language. Our idea presented in this paper aims to develop a spell-checking system for Arabic language independently from the vocabulary[4] by introducing morphological analysis in the spell-checking process.

[3] http://aracomlex.sourceforge.net
[4] in the broadest sense of vocabulary

II. THE MORPHOLOGICAL ANALYZER: ARAMORPH

The Buckwalter morphological analyzer [15] developed by LDC (Linguistic Data Consortium), named AraMorph, allows segmenting each word into a sequence of triplet "prefix-stem-suffix". The AraMorph analyzer is formed mainly on three lexicons: prefixes (548 entries), suffixes (906 entries) and stem (78 839 entries). Lexicons are complemented by three compatibility tables used to cover all the possible combinations of prefix-stem (2435 entries), suffix-stem (1612 entries) and prefix-suffix (1138 entries). Thus, the parser will output the stems, prefixes and suffixes associated to the word to be analyzed, and then it checks the validity of these solutions in the lexicon of the system and in the correspondence tables prefix-stem, stem-suffix and prefix-suffix. The stems used in AraMorph are constructed as follows: the stems of root "**فعل**" are: "**فاعل**", "**فعول**", "**فعيل**" , "**فوعل**" and "**فعال**".

III. THE LEVENSHTEIN DISTANCE

Among the most known metrical methods in the field of spell-checking, we have the unavoidable Levenshtein distance [3], also known as the Edit Distance. The edit distance calculates the minimal number of elementary editing operations required to transform a misspelled word to a dictionary word. Editing operations considered by Levenshtein are: insertion, deletion, and permutation. The procedure of calculating the edit distance between two strings

$P = p_1 p_2 \dots p_m$ and $Q = \mathrm{q}_1 \mathrm{q}_2 \dots \mathrm{q}_\mathrm{n}$ where the length is respectively m and n, consists in calculating recursively step by step in a matrix $O(m, n)$ the edit distance between different substrings of P and Q.

The calculation of the cell (i, j) corresponding to the edit distance between the two substrings

$P_{1}^{i} = p_1 p_2 .. p_i$ and $\mathrm{Q}_1^i = q_1 q_2 .. q_i$, is given by the following recurrent relation:

$$\mathrm{D}(\mathrm{i},\mathrm{j}) = \mathrm{Min}\begin{cases} \mathrm{D}(\mathrm{i}-1,\mathrm{j}) + 1, \\ \mathrm{D}(\mathrm{i},\mathrm{j}-1) + 1, \\ \mathrm{D}(\mathrm{i}-1,\mathrm{j}-1) + \mathrm{cost} \end{cases} \qquad (1)$$

with

$$cost = \begin{cases} 0 & if\ \ p_{\mathrm{i}-1} = q_{\mathrm{j}-1} \\ 1 & otherwise \end{cases} \qquad (2)$$

Admitting these following initializations: $\mathrm{D}(\mathrm{i}, \emptyset) = i$ and $\mathrm{D}(\emptyset, \mathrm{j}) = j$, where $\emptyset$ is the empty string.

IV. INTRODUCING MORPHOLOGICAL ANALYSIS INTO LEVENSHTEIN DISTANCE

Our new idea in this work is to use a dictionary of small size that represents Arabic language stems[5] to correct spelling errors instead of using a large dictionary. In other words, our vision is to invest in a relevant metric method instead of building a dictionary that covers all the words in a given language, which is usually difficult to build.

[5] Stems dictionary is the one used by Buckwalter in AraMorph Parser

We note by:

- $T = \{T_1, T_2, \dots T_i\}$ The set of Arabic stems.
- $\mathrm{P} = \{\mathrm{P}_1, \mathrm{P}_2, \dots \mathrm{P}_j\}$ All Arabic prefixes.
- $S = \{S_1, S_2, \dots S_k\}$ All Arabic suffixes.
- L means an Arabic lexicon.

Let $\mathrm{W}_{\mathrm{err}}$ a misspelled word, Levenshtein distance consists in finding the words W_{s} satisfying the following relation:

$$W_s = \underset{W_i \in L}{ArgMin}(D_{lev}(W_{err}, W_i)) \qquad (3)$$

with D_{lev} presents Levenshtein distance.

According to the morphological analysis approach, there exist (P_v, T_v, S_v) in $P \times T \times S$ such as $W_i = P_v T_v S_v$ respectively for the misspelled word $W_{err} = P_{err} T_{err} S_{err}$, where P_{err} means an erroneous prefix and T_{err} means an erroneous stem and S_{err} means an erroneous suffix.

In order to introduce the morphological analysis concept (used by Buckwalter) in the Levenshtein algorithm, we have defined a new measure noted M, as well the measurement between W_{err} and vector (P_v, T_v, S_v) is given by the following formula:

$$\boldsymbol{M}(W_{err}, (P_v, T_v, S_v)) = ArgMin_{(P_v,T_v,S_v) \in (P \times T \times S)} [\underset{W_{err}=P_{err}T_{err}S_{err}}{Min}((D_{lev}(P_{err}, P_v) + D_{lev}(T_{err}, T_v) + D_{lev}(S_{err}, S_v))] \quad (4)$$

The corrections of erroneous word W_{err} are given by:

$$W_s = P_s T_s S_s = \underset{(\mathrm{P_v}, \mathrm{T_v}, \mathrm{S_v}) \in P \times T \times S}{ArgMin} \boldsymbol{M}(W_{err}, (\mathrm{P_v}, \mathrm{T_v}, \mathrm{S_v})) \qquad (5)$$

For all prefixes, stems and suffix respectively belonging to P, T and S, we calculate the minimum only on prefixes, stems and suffixes that are compatibles with each other, and that by introducing the three tables of correspondence between prefix-stem, stem-suffix and prefix-suffix already used by Buckwalter.

<u>Example</u>**:**

Let "قغخل" a misspelled word to correct. By applying the formula (4), we get the following solutions in these first orders:

Min Prefix	Min Stem	Min Suffix
D_{lev} (∅, ف)= 1	D_{lev} (قغخل, دخل)=2	D_{lev} (∅, ∅)=0
D_{lev} (ق, ف)= 1	D_{lev} (غخل, غول)=1	D_{lev} (∅, ـت)=1
D_{lev} (قغ, ف)=2	D_{lev} (غخل, دخل)=1	
D_{lev} (قغخ, ف)=3	…..	…..
D_{lev} (قغخل, ف)=4	......	......

– $\boldsymbol{M}$("قغخل", (ف, "دخل", ∅)) **=2** (wich presents the minimal distance in all stems, prefixes and suffixes) → the system suggest the solution "**فدخل**" as a correction of the misspelled word "**قغخل**", with distance 2.

– Thus our method returns the solution $\boldsymbol{M}$("قغخل", (ف, "غول", ∅)) **=2** → which represents the word "**فغول**", with distance 2.

V. Tests and Results

To highlight our approach, we have developed a spell-checking program[6] that allows comparing our method to the classical approach of Levenshtein.

The list of words used in this study as reference lexicon for Levenshtein approach contains more than 170000 words extracted from MySpell[7] program of Open Office Writer.

For our approach, we relied on a list of prefixes, suffixes and a list of stems built on Buckwalter approach basis, for example, besides the root "**فعل**" we find also the stems list of the five forms generated from this root: **"فاعل"**, **"فعول"**,**"فعيل"**,**"فوعل"** and **"فعال"**.

The rectifications suggestions proposed by Levenshtein distance are the word of minimal distance relative to the misspelled word. For our approach, we used the formula (4), explained in the previous paragraph. For our tests, we have used a corpus of 2784 misspelled words. There were three types of errors: addition, deletion and permutation. The table below shows the rate of correction by editing operations:

TABLE I. COMPARATIVE TABLE BETWEEN THE TWO METHODS

		Our approach	Levenshtein distance
Editing operators	Insertion	**85%**	**44%**
	Deletion	**81%**	**61%**
	Permutation	**86%**	**46%**
Average time / Erroneous word		**0.10 ms**	**0.19 ms**

To compare our new approach with Levenshtein's, we have used the following three indicators:

– The correction average time.
– The rate of rectified words.
– The size of each system lexicon.

- We have taken 170000 words as lexicon size for Levenshtein method. For our system, the theorical

[6] Developed in Java language under Eclipse platform
[7] http://myspell.sourceforge.net

lexicon size is N words, with: N= Nbre Prefixes x Nbre Suffixes x Nbre Stems ≈ 197x106 words. The real size of our system (lexicon) is much less than this number because the tables of correspondence between suffixes, prefixes and stems reduce this number. Generally, it is a ten of millions of words order.

- Despite the fact that the number of words covered by our method is about 1000 times higher than the size of the lexicon used by Levenshtein method in this study, the average time to correct a word is faster in our method 0.10 ms versus 0.19 ms in Levenshtein's.
- For numeral results regarding correction rate, it is obvious that our system rectify misspelled words 84% more correctly versus an average of 50.3% within Levenshtein distance. This difference is mainly due to the fact that our lexicon contains enough words compared to Levenshtein's. Spelling errors in our system are mainly due to the stems base insufficiency that stays incomplete and do not contains all the stems.

We clearly notice that our system is better than Levenshtein's, either at lexicon level or at runtime level or compared to the correction rate.

VI. Conclusion

We can see clearly, that our system is much better that the one using classical comparison between two lexical forms via Levenshtein distance. The result we have gotten in the previous paragraph shows clearly the interest of our new approach and the facility of integration it has in an automatic spell-checking system.

References

[1] Kukich K.," Techniques for Automatically Correcting Words in Text ", ACM Computing Surveys, Volume 24, No.4, pp, 377-439, December 1992.

[2] Damerau F.J.," A technique for computer detection and correction of spelling errors ", Communications of the Association for Computing Machinery, 1964.

[3] Levenshtein V.," Binary codes capable of correcting deletions, insertions and reversals ", SOL Phys Dokl,pp, 707-710, 1966.

[4] Pollock J. and Zamora A., " Automatic Spelling Correction in Scientific and Scholarly Text ", Communications of the ACM, 27(4), pp, 358-368, 1984.

[5] Ukkonen E., " Approximate string matching with q-grams and maximal matches ", Theoretical Computer Science, 92, pp, 191–211, 1992.

[6] Oflazer K.," Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction ", Computational Linguistics Archive Volume 22 Issue 1,pp,73-89, March 1996.

[7] Gueddah H., Yousfi A. and Belkasmi M.," Introduction of the Weight Edition Errors in the Levenshtein Distance ", International Journal of Advanced Research in Artificial Intelligence, Volume 1 Issue 5,pp, 30-32, 2012.

[8] Gueddah H. and Yousfi A., " Etude Statistique sur les erreurs d'édition dans la langue Arabe", La 5éme conférence internationale sur les Technologies d'Information et de Communication pour l'Amazighe, IRCAM, Septembre 2012.

[9] Mitton R., " Ordering the suggestions of a spellchecker without using context ", Natural Language Engineering 15 (2), pp, 173-192, 2009.

[10] Emirkanian L. and Bouchard L.H.," La correction des erreurs d'orthographe d'usage dans un analyseur morphosyntaxique du français " dans langue Française N 83, Paris, Larousse, pp, 106-122, 1989.

[11] Romanycia M.H. and Pelletier J.F., "What is an heuristic? " Computational Intelligence, volume 1, pp, 47-58, 1985.

[12] Bowden T. and Kiraz G.A.," A morphographemic model for error correction in nonconcatenative strings ", Proceedings of the 33rd annual meeting on Association for Computational Linguistics, pp, 24-30, 1995.

[13] McCarthy J., "A prosodic theory of non-concatenative morphology ", Linguistic Inquiry 12(3), pp, 373-418, 1981.

[14] Shaalan K., Samih Y., Attia M., Pecina P., Genabith J.V.," Arabic Word Generation and Modelling for Spell Checking ", In the Proceedings of The 8th international conference on Language Resources and Evaluation (LREC'12), pp,719-725, May 2012.

[15] Buckwalter T., " Buckwalter Arabic Morphological Analyzer version 1.0 ", Philadelphia: Linguistic Data Consortium, Catalog No.LDC2002L49, ISBN 1-58563625760, 2002.