On the Computerization of African Languages

Harouna Naroua and Lawaly Salifou

Département de Mathématiques et Informatique, Université Abdou Moumouni, Niamey, Niger

Abstract: In this article, a computer tool for processing African languages has been designed. It is intended to be a contribution to the automatic processing of African languages. The current study is focused on West African languages where five main languages from Niger, two from Mali and one from Burkina Faso are considered. After a brief review of African languages processing, we designed a tool which uses minimum resources and operates essentially on a dictionary and the characteristics of the language alphabet. The dictionary is represented using a trie data structure. For the sake of application, the designed tool operates as a spell checker. To detect and correct spelling errors, the edit distance and the specificities of the language are used. Although they do not have processing tools, it was shown that existing tools for computerized languages can be adapted to African languages efficiently. To extend the designed tool to any African language, we only need to provide an appropriate dictionary and alphabet.

Keywords: Computerization, Natural Language Processing, Spell Checker, African Language, Computer Resource

Introduction

The importance of Automatic natural language processing cannot be overemphasized. It has many industrial applications such as spell checking, parsing, text indexing and retrieval of information from the Internet, voice recognition and synthesis, vocal control of domestic robots, automated response systems and machine translation (Kukich, 1992; Pierre, 2006). Applications such as text editing are used by millions of people every day. They are present in all computer systems, internet search engines and electronic gadgets. To make their use more effective, processing tools like spell checking and grammar correction are integrated to these softwares and the main objective is to assist the user. While spell checking is concerned about detecting and correcting single word errors, parsing is concerned about grammatical errors detection and correction through a rigorous syntax analysis. Though syntax analyzers are more important, the first step towards the development of such programs is that of spell checkers where three different techniques are used which are non-word error detection, isolated-word error correction and context error detection and correction (Kukich, 1992). Such programs are generally designed for a given language. Though different processing techniques exist for other languages like English and French, African languages do not yet have processing tools.

Review of African Languages Processing

Africa is a continent with a very high linguistic diversity. Estimated at about 1500-2000 languages, four main groupings can be distinguished which are Afro-Asiatic, Nilo-Saharan, Niger-Saharan and Khoisan (see the AG Bell Association web site). All African languages are considered official languages of the African Union. Representing one third of the world's languages (Van Der and Gilles-Maurice, 2003), African languages are an important and irreplaceable component of the linguistic heritage of humanity and its ecolinguistic diversity. According to Osborn (2006), it is clear that African languages are not yet widely used in the content of computing applications or on the internet. He observed that African languages are represented on the web but not prominently as media of communication. A significant number of sites that treat African languages were given by Diki-Kidiri and Edema (2003) but with minimal content in the languages themselves.

The existence of resources is the first step in the computerization of a language (Chanard and Popescu-Belis, 2001). The majority of well-resourced languages have well-formed corpuses but this is not the case for African
languages. Despite the existence of different projects aimed at the computerization of African languages such as Pal and DILAF, the resources of these languages are still very scarce. Nevertheless, important resources like bilingual and editorial dictionaries exist and can be used in favour of African languages computerization. Although they have a lot of differences, we believe in the possibility of using the little resources they have in common to develop processing tools. From a vocal point of view, different tones can be found in the words of African languages. For example, Hausa words have high tones and low tones and one can observe a flexion of gender and number (Mijinguini and Naroua, 2012). Many African languages are currently used by major radio stations of the world such as BBC (UK), CRI (China), Deutsche Welle (Germany), IRIB (Iran), Radio Moscow (Russia), RFI (France) and VOA (USA). Unfortunately, the presence of African languages on Internet is very precarious even though they represent 30% of the languages of the world (Van Der and Gilles-Maurice, 2003). The current researches on African languages choose oral and written corpuses as a transitional alternative or build a corpus from the Web (Gilles-Maurice, 2002).

Another difficulty to overcome in the computerization of African languages is text entry. In fact, computer keyboards are designed for well-resourced languages and are not compatible with African languages. To enter texts in languages like Hausa, Fulfulde and Bambara requires special softwares. Unfortunately, this is the case for all African languages due to the presence of special characters. An evaluation of keyboard layouts for five languages from Niger (Fulfulde, Hausa, Kanuri, Songhai-Zarma, Tamasheq) recommended LLACAN which covers all the symbols of the alphabets of those languages, produces valid Unicode code and requires less buttons to press (Enguehard and Naroua, 2008).

The ability of processing a language with the famous word processing softwares like MS Word and OpenOffice Writer is another important issue. Various techniques have been developed for well-resourced languages for various purposes. However, all existing techniques are limited and inadequate in the case of African languages. Despite the scarcity of linguistic resources, it is possible to develop computer tools for African languages and improve them over the time with the possibility to create extensions for some popular softwares.

**Spell Checking Techniques**

The main objective of spell checkers is to detect and correct errors. Their task is composed of three sub-tasks: Detecting errors, generating possible corrections and ranking suggested corrections. To achieve this, various techniques were invented. Each technique is related either to non-word error correction, real-word error correction, or both. Spelling errors may be typographical, cognitive or phonetic. Typographical errors occur when the keys are pressed in the wrong order. Cognitive errors arise from ignorance of the correct spelling of the word. Phonetic errors are special cases of cognitive errors. A phonetic error refers to a wrong word that is pronounced the same way as the correct word. It was shown that in typed texts, 1 to 3% of the errors are spelling errors (Daniel and James, 2000). Damerau (1964) stated that 80% of these errors are related to insertion, deletion, substitution, or transposition.

Error detection is to find incorrect words in a text. A wrong word is then marked by the application in charge of spell checking. If the word is really wrong, an error is said to be detected. Many authors have made important contributions in this area like (Damerau 1964; Cyril, 1967; Peterson, 1980; Zamora et al., 1981; Laurent, 2001; Suzan, 2002; Pierre, 2006). The main techniques used for non-word error detection in a text are either based on analysis of n-grams, or dictionary lookup (Kukich, 1992). The techniques based on n-grams are to analyze each n-gram of a given input word and check its validity in a precompiled table. These techniques usually require a dictionary or corpus that’s large enough to determine the statistics table of n-grams (Kukich, 1992). A dictionary is a collection of correct or acceptable words. The techniques based on the use of a dictionary or lexicon involve taking a word as input and verifying its existence in the dictionary. Any word that is not in the dictionary is then considered wrong (Kukich, 1992). A detection algorithm based on dictionary lookup is given by Peterson (1980). Some of the data structures used in spell checking are hash tables, binary search trees, tries and finite automata. One of the famous algorithms in this area is that of Aho and Corasick (1975). The algorithm is to move through an abstract data structure called dictionary that contains the words to search by reading the text characters one by one. The data structure is implemented efficiently, which ensures that each character of the text is read only once. Generally, the dictionary is represented using a trie. A trie may be seen as a representation of the transition function of a deterministic finite automaton. The algorithm has a linear complexity in the size of the text and search strings. Comparatively, techniques using n-grams derived from a dictionary provide less accuracy than those using all the information in the dictionary. But, the latter ones are time consuming depending on the data structure used to represent the dictionary. A comparative study showed that the hash table provides better performance than the AVL tree, the Red-Black tree and Skip list (Mark, 2009). A comparison of five data structures was performed for the Punjabi dictionary (Lehal and Singh, 2000). It concerned binary search tree,
Error correction refers to the fact of equipping spell checkers with the ability to correct detected errors. This is to find words in the dictionary that are similar in some ways to the misspelled word. The minimum edit distance or simply edit distance is until now the most widely used technique in the spelling errors correction. It has been applied in almost all spell checking functions in text editors and command language interfaces.

The first spelling correction algorithm based on this technique was proposed by Damerau (1964). Almost at the same time, Levenshtein also developed a similar algorithm. Several other algorithms on edit distance were born thereafter. The edit distance is defined as the minimum number of edit operations required to transform a word to another (Kukich, 1992). These operations are insertion, deletion, substitution and transposition. In most cases, correcting a spelling error requires the insertion, deletion or substitution of a single character, or the transposition of two characters. When a wrong word can be transformed into a dictionary word by inverting one of these operations, the dictionary word is considered a plausible correction.

Damerau’s algorithm (Damerau, 1964) for edit distance detects spelling errors by comparing words of four to six characters with a list of most frequently used words. When there are multiple candidate words for a given edit distance on a detected word, the first word in the dictionary appearing in alphabetical order is chosen.

Accuracy and efficiency of edit distance-based spell checkers depend on the selection of dictionary words. To achieve this, a number of classes are necessary. The class diagram of the entire process is shown in Fig. 1.

The words are represented in form of nodes. Each node has as many links as there are characters in the alphabet. Each valid character string is assigned a value. This may be of any type. It can be used to store information on every word in the dictionary such as definition, grammatical class, translation into another language, etc.
Table 1. Special characters used in the alphabets of the studied languages

| Name                        | Sign | Languages | Unicode |
|-----------------------------|------|-----------|---------|
| Latin letter e with tilde   | ê    | son       | U+0065  |
| Latin letter i with tilde   | i    | son       | U+0069  |
| Latin letter o with tilde   | ô    | son       | U+006F  |
| Latin letter r short stroke overlay | r | kau | U+0072  |
| Latin letter u with tilde   | ü    | son       | U+0075  |
| Latin letter a with tilde   | å    | son       | U+00E3  |
| Latin letter a with breve   | ä    | son       | U+0103  |
| Latin letter ENG            | η    | bam, ful, son | U+014B  |
| Latin letter s with caron   | š    | tmh       | U+0161  |
| Latin letter k with hook    | ƙ    | hau       | U+0199  |
| Latin letter y with hook    | γ    | ful, hau  | U+01B4  |
| Latin letter y with hook    | y'   | ful, hau  | U+01B4  |
| Latin letter turned e       | a    | kau, tmh  | U+01DD  |
| Latin letter g with caron   | g    | tmh       | U+01E7  |
| Latin letter j with caron   | j    | tmh       | U+01F0  |
| Latin letter b with hook    | b    | ful, hau  | U+0253  |
| Latin letter open o         | o    | bam       | U+0254  |
| Latin letter d with hook    | d'   | ful, hau  | U+0257  |
| Latin letter gamma          | y    | tmh       | U+0263  |
| Latin letter epsilon        | r    | bam       | U+025B  |
| Latin letter n with retroflex hook | n | bam, son | U+0272  |
| Latin letter d with dot below | d   | tmh       | U+1E0D  |
| Latin letter l with dot below | l   | tmh       | U+1E37  |
| Latin letter s with dot below | s   | tmh       | U+1E65  |
| Latin letter t with dot below | l   | tmh       | U+1E6D |
| Latin letter z with dot below | z   | tmh       | U+1E93  |

The R attribute of the class Trie is the number of symbols or letters of the alphabet. The characters are represented by indices of next array (Node [ ]). Unfortunately, representing characters by indices of next array will set a large value for R. This will inevitably lead to a waste of memory space and additional checks to prevent foreign words from being added to the trie. To avoid this problem, a trick is to find a mapping function
between indices of the next array and letters of the alphabet (Robert and Kevin, 2011). That is why the alphabet attribute is present in the class Trie. It is of type String but it may also be an array of characters. Two additional methods, toChar and toIndex assure the conversion from indices to characters and vice versa. The charAt and the indexOF methods of the String class can be effectively used and to make the trick more flexible, we can totally delegate this task to an interface Alphabet that defines toChar and toIndex. The KeysThatMatch is another interesting method. Indeed, it allows to search the trie for words that match a given pattern. The patterns used are those with a wildcard, for example a dot ('\'). It is this possibility that we use to implement the reverse editing distance. The KeysThatMatch method uses a data structure List (a linked list of Strings) to keep the search results. The List class has methods to add an item, to verify the existence of an item and to delete an item. Tags may be used to take care of the contextual meanings of a word. It may be very useful for applications like syntax analyzers.

To abstract the implantation of the real dictionary, add flexibility, simplify maintenance and facilitate scalability of the spell checker, an abstract dictionary is represented by a class (TrieBasedDico) that implements Dico interface (or abstract class). It defines the methods (add, remove, contains) needed to operate on a dictionary. TrieBasedDico class is designed by composition from Trie class.

The list of candidate words for the correction of an erroneous word is determined in several steps. Once a word is identified as being erroneous, the procedure for determining the type of the error follows. We defined three types of errors (inspired by our research on OpenOffice):

- **IS_NEGATIVE_WORD**: Error caused by the presence of a number or a character not belonging to the alphabet in the word. The word is called negative
- **CAPTION_ERROR**: Case Error. This is when a word that should be written with the first letter capitalized is written entirely in lowercase
- **SPELLING_ERROR**: represents all other types of spelling errors

The types of errors are short integers encapsulated as static fields in the LySpell class. The corrector has two methods for the determination of errors. First, the getSpellFailure method which analyzes a given word and returns -1 if the word is correct or one of the three types of errors mentioned above otherwise. Then isValid method that checks whether a given word is valid according to the result returned by getSpellFailure and spellchecking settings. If getSpellFailure returns a value:

- Equal to -1, the word is valid and isValid returns true
- Other than -1, the correction parameters are taken into account to determine the validity of the word. For example when you choose not to correct words with numbers and the erroneous word contains digits, isValid returns true. This method can be exploited to correct spelling as you type

**currentLanguage** represents the language being supported by the spell corrector. It is an instance of Language class. Searching suggestions is performed by **propose** which is an instance of a class that implements the interface Proposer.

The method getProposals provides correction suggestions for an invalidated word by isValid depending on the type of error detected by getSpellFailure.

The processed language is represented by the Language class. After several attempts, we decided that the dictionary is an attribute of the language and not the reverse. The local attribute of the Language class stores information about the processed language. It is of type Locale (representation of a language in Java) and provides among others: A two-letter ISO 639-1 code of the language, a two-letter ISO 3166 code of the country as well as the complete names of the language and the country. We use this data for naming resources and for user display. The properties attribute is of type Map (mapping key/value) and stores other properties of the language that we use to design the checker and which are not provided by Locale. They are currently the alphabet of the language, the special characters in the alphabet, the characters that look like special characters and the punctuation symbols that we divided into two parts: Word separators and end of sentence signs. All the characters of the alphabet are coded in Unicode. The class in charge of finding suggestions implements Proposer interface which defines two methods: isNegativeWord and propose. The TrieBasedDicoProposer class uses some features of the alphabet to find candidate words which are found using the reverse edit distance as follows:

- All words having an edit distance equal to 1 with the wrong word are generated by applying edit operations such as insertion, deletion, substitution and transposition. A total of 60n+28 words are generated for a wrong word of length n
- Each previously generated word is searched in the trie. If it is there, then it is retained as a possible correction of the erroneous word

The research is conducted by a private method called proposeByReverseEditDistance. This method is actually based on keysThatMatch. It takes an argument of type TrieBasedDico and a word or a pattern and returns the result as an array of Strings. Methods that perform
editing operations on a given word are provided by the *StringTools* class which consists of tools shared by different classes. The minimum edit distance is used to rank the suggested words. Those who are closest to the wrong word are placed at the top of the list. To implement that, a comparator was designed.

**Implementation**

In this article, a software tool has been proposed for processing eight West African languages from three different countries. It has been designed as a spell checker for which only a dictionary and the alphabet of a language are needed as linguistic resources. A prototype of the designed software tool has been implemented in Java under Linux environment. For the sake of application, a dictionary (Mijinguini, 2003) and the official alphabet of the Hausa language of Niger are used. The developed tool was tested as a standalone program through a text editor designed for that purpose and as an extension for the OpenOffice suite. Figure 2 and 3 respectively show the dialog box for spell checking and error correction in OpenOffice writer. However, in OpenOffice, the Hausa options are only Nigeria and Ghana and we are limited to select one of them. From the results, we observe that:

- It is possible, from minimum language resources and proven techniques to develop automatic processing tools for African languages
- Only a dictionary as language resource and the alphabet of an African language are needed to develop a spell checker
- The specificities of the African languages can be efficiently handled
- Although they contain special characters, the dictionaries of African languages can efficiently be represented by tries

**Conclusion**

This work is a contribution to the automatic processing of African languages. Although it may be necessary to improve the performance of the designed tool, we believe that the results we have obtained will add value to the computerization of African languages and contribute to their effective use in institutions of education and on media.

**Acknowledgement**

The authors would like to thank Université Abdou Moumouni of Niamey for the financial support.

**Funding Information**

The study was supported by the research grant offered by Université Abdou Moumouni of Niamey.

**Author’s Contributions**

The authors have equally contributed to the research work and writing of the article.

**Ethics**

This article is original and contains unpublished material. The corresponding author confirms that all authors have read and approved the manuscript and no ethical issues are expected to arise after its publication.
References

Aho, A.V. and M.J. Corasick, 1975. Efficient string matching: An aid to bibliographic search. Commun. ACM, 18: 333-340. DOI: 10.1145/360825.360855

Chanard, C. and A. Popescu-Belis, 2001. Encodage informatique multilingue: Application au contexte du Niger. Les Cahiers du Rifal, 22: 33-45.

Cyril, N.A., 1967. String similarity and misspellings. Commun. ACM, 10: 302-313. DOI: 10.1145/363282.363326

Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Comm. ACM, 7: 171-176. DOI: 10.1145/363958.363994

Daniel, J. and H.M. James, 2000. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. 1st Edn., Prentice Hall, Englewood Cliffs, Inc., ISBN-10: 013122798X, pp: 934.

Diki-Kidiri, M. and A.B. Edema, 2003. Les langues africaines sur la Toile. Cahiers du Rifal, 23: 5-32.

Enguehard, C. and H. Naroua, 2008. Evaluation of virtual keyboards for West-African languages. Proceedings of the Sixth International Conference on Language Resources and Evaluation, May 28-30, Marrakech, Morocco, pp: 1-5.

Gilles-Maurice, D.S., 2002. Web for/as Corpus: A perspective for the African languages. Nordic J. Afr. Stud., 11: 266-282.

Horst, B., 1993. A fast algorithm for finding the nearest neighbor of a word in a dictionary. Proceedings of the 2nd International Conference on Document Analysis and Recognition, Oct. 20-22, IEEE Xplore Press pp: 632-637. DOI: 10.1109/ICDAR.1993.395657

Hsuan, L.L., 2008. Spell checkers and correctors: A unified treatment. MSc. Thesis, University of Pretoria.

Kukich, K., 1992. Techniques for automatically correcting words in text. ACM Comput. Surveys, 24: 377-439. DOI: 10.1145/146370.146380

Laurent, B., 2001. Production de logiciels et d'utilitaires pour le traitement informatique de langues africaines dans un contexte de NTIC multilingues. Proceedings of the 2nd World Congress of Community Networks, (CCN’ 01), Buenos Aires, Argentine, pp: 1-13.

Lehal, G.S. and K. Singh, 2000. A comparative study of data structures for Punjabi dictionary. Proceedings of the 5th International Conference on Cognitive Systems, Reviews and Previews, (ICC’ 99), pp: 489-497.

Mark, P.N., 2009. A comparison of dictionary implementations.

Mijinguini, A., 2003. Dictionnaire Elémentaire Hausa-Français. 2nd Edn., Editions GG, Niamey, Niger, pp: 752.

Mijinguini, A. and H. Naroua, 2012. Règles de formation des noms en haoussa. Proceedings of the Conférence Conjointe Traitement Automatique des Langues Africanes, (ALA’ 12), JEP-TALN-RECITA, pp: 63-74.

Osborn, D.Z., 2006. African languages and information and communication technologies: Literacy, access and the future. Proceedings of the 35th Annual Conference on African Linguistics: African Languages and Linguistics in Broad Perspectives, (CAL’ 06), Cascadilla Proceedings Project, Somerville, MA, USA, pp: 86-93.

Peterson, J.L., 1980. Computer programs for detecting and correcting spelling errors. Comm. ACM, 23: 676-687. DOI: 10.1145/359038.359041

Pierre, M.N., 2006. An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation and Application with Special Consideration of English, French and German. 1st Edn., Springer Science and Business Media, Berlin, ISBN-10: 3540343369, pp: 515.

Robert, S. and W. Kevin, 2011. Algorithms. 4th Edn., Addison-Wesley Professional, ISBN-10: 0132762560, pp: 992.

Suzan, V., 2002. Context-sensitive spell checking based on word trigram probabilities. MSc. Thesis, University of Nijmegen.

Van Der, A.V. and D.S. Gilles-Maurice, 2003. The African languages on the internet: Case studies for hausa, somali, lingala and isiXhosa. Cahiers Du Rifal, 23: 33-45.

Zamora, E.M., J.J. Pollock and Z. Antonio, 1981. The use of trigram analysis for spelling error detection. Inform. Process. Manage., 17: 305-316. DOI: 10.1016/0306-4573(81)90044-3

AG Bell Association. http://www.nationsonline.org/oneworld/african_languages.htm