PROTOTYPING NLP NON-WORD DETECTION SYSTEM FOR DINKA USING DICTIONARY LOOKUP APPROACH

Manyok Chol David[1] and Robert Balagadde Ssali[2]

[1]Department of Information Technology, University of Juba, South Sudan
[2]Department of Computer Science, School of Mathematics and Computing at Kampala International University

ABSTRACT

Spellcheckers are computer software used for non-word or real word error detection. The Dinka text editors have been developed, however, no one has developed their spellcheckers. The research entitled Prototyping NLP Non-Word Detection System for Dinka Using Dictionary Lookup Approach was a solution to Dinka spellchecking. The study objectives were: requirements gathering and analysis. The computer keyboard was customized to accept the Dinka characters. Dinka lexicon was created with 6,976 words. The prototype was implemented using java programming language and dictionary lookup approach was used for non-word detection. The accuracy of detection (detecting real words and non-words) gave 98.10%, and the accuracy of non-word detection (detection of non-words only) was 91.36%. The True Positive Rate (TPR) was 99.10% and the True Negative Rate (TNR) was 91.36 %. The speed of non-word detection which was found as1, 044 Hz was slow.

KEYWORDS

Dinka, lexicon, non-word, detection, correction, prototype.

1. INTRODUCTION

Non-words detection system is a tool used for spellchecking that can be intended for non-word checking, real word error checking or both. It is one of the most widely used tools in Natural Language Processing (NLP) that is incorporated for spelling, grammar, and style-checking for English or other languages [1].

Non-words are words that a text editor flagged as errors, because it cannot find them in the referenced lexicon, however, they may or may not exist in a given language and these are referred to as non-word errors when they arise from misspelling [2].

The problem of spelling and grammar checking has been a research topic for decades in computational linguistics [3]. The first attempts were based on the idea that a combination of lexicon and hand-crafted grammar rules were used to suffice and solve the problem for both spelling and grammar [4].

The development of tools and methods for language processing concentrated on languages mainly from the industrial part of the world. There is, however, a potentially larger need for investigating the application of computational linguistic methods to the languages of the developing countries, since the number of computer and internet users is growing, while most of them do not speak the European and East-Asian languages [5]. As the demand for computer

usage in local languages increases, there is a need to build robust and effective approaches to detect errors [4].

Africa is a continent with a very high linguistic diversity, about 1500-2000 languages [6]. It is clear that African languages are not yet widely used in computing [7].

As Africa languages increase their presence in the use of computers, the need for spellcheckers arise and the construction of spellcheckers become a forward step in their computerization [8].

The Dinka are a group of related people of South Sudan and Sudan. They are grouped into: Northeastern, Northwestern, Southeastern, Southwestern and South Central. This group has a lexical similarity of 84-92% [9] and a total of 3.5 million [10]. Dinka is Nilo-Saharan language and its orthography was adapted from Latin, Greek, Coptic, and International Phonetic Alphabet [11].

The Juba Rejaf Language conference of 1928 came up with the development of Dinka orthography which was continuously modified [9]. Between 1928 -1950, the missionaries worked on Dinka language to translate the Bible and to developed educational materials such as PadaangDinka-English dictionary. Despite all the efforts, the orthography was not comprehensive and had to be continuously modified because of limited vocabularies. Consequently, some words were borrowed from Arabic, for example '*bermil'* for '*barrel'* and '*tharabeth'* for '*table'* [9].

The Dinka speakers face challenges of spelling, because of the dialect differences that caused a loss of the distinction between initial velar and palatal stop, for example '*kiir'* means '*river'* and '*ciin'* means '*parts of animal hides'*. These words *begin* with the same sound in Dinka Agar and DinkaBor South respectively [12].

The adoption of unique letters such as *ä, ë, ï, ö, ŋ, ɤ, ɔ, ɛ* and *ɛ̈* from Latin, Greek, Coptic and IPA created a problem in adapting existing spellcheckers to be used in Dinka. Though commercial programs are available for Dinka text processing, they lack tools for non-word detection.

1.1. Research motivation

The number of Dinka people using computer and the internet is growing due to the demand of written Dinka langauge. To computerize the language, commercial word processors have been developed, however, these text input programs lack spellcheckers. The adoption of characters from Latin, Coptic, Greek and International Phonetic Alphabet (IPA) that are used in Dinka alphabets makes it hard for existing algorithms and non-word error detection systems to be adopted for Dinka spellchecking, because they do not exist in the Standard English alphabets [12]. It is based on the identified gaps that the researcher Prototyped NLP Non-Word Detection System for Dinka Using Dictionary Lookup Approach.

1.2. Objectives of the study

1.2.1. General objective

The general objective of the research was to Prototype NLP Non-Word Detection System for Dinka using Dictionary Lookup Approach.

1.2.2. Specific objectives

1. To collect and analyze system requirements.
2. To create a Dinka lexicon from existing Dinka corpus.
3. To design and develop a module to customize computer keyboard for Dinka text input.
4. To design and develop Input Module, Detection Module, Suggestion and Correction Module, Add Module, Ignore Module and Display Module for Dinka non-word detection prototype.
5. To test and evaluate the Dinka non-word detection prototype using a Dinka lexicon.

2. METHODOLOGY

2.1. Research design

The research was an experiment to test the accuracy of detection and speed of non-word detection.

A computation of the accuracy of non-word detection of the spellchecker was done using the variables namely; **N, TP, TN, FN** and **FP**.

Where; **True Positives (TP)** are defined as words that exist in the referenced lexicon, **True Negatives (TN)** as words that the spellchecker cannot find in the referenced lexicon, **False Positives (FP)** as non-words that the spellchecker fails to detect and **False Negatives (FN)** as words that the spellchecker fails to identify as real words, yet they exist in the referenced lexicon.

The accuracy of detection of the spellchecker was computed as **(TP+TN)/N**. **The True Positive Rate (TPR)** which is the rate that shows how accurate the spellchecker detects the real words was computed as **TPR=TP/ (TP+FN)** and **True Negative Rate (TNR)** which is the rate at which the spellchecker detect the non-words, was calculated as **TNR= TN/ (FP+TN)** [13] and [14].

2.2. Research approach

The prototype took a mixed approach of quantitative and qualitative approaches. The designs and development of the prototype required description of procedures, hence the qualitative approach, while testing data required numerical analysis, hence quantitative approach.

2.3. Designing the Dinka lexicon

The collection of 6,976 unique Dinka words were collected from the Dinka Bible and written to the lexicon named the **dictionary.txt**.

2.4. Modules designs.

The modules were designed using UML in order to diagrammatically explain their functions and it covered the following;

Input Module: The diagram of Input Module was designed using flowchart to explain how a user would input Dinka words using the keyboard or by browsing to a file on the computer local disk.

Display Module: The flowchart for the Display Module was used to elaborate how the prototype would display Dinka non-words and suggests the correct words to the user.

Detection Module: The flowchart and sequence diagrams were used to design the Detection Module to explain diagrammatically how detection would be carried out on Dinka non-words.

Suggestion and Correction Module: These model were elaborated using a parse tree diagram. The Dameru Edit Distance of one was used to generate the Dinka word suggestions based on deletion, transposition, insertion and substitution techniques.

Add Module: The Add Module design describes how Dinka non-words from the display would be added to the Dinka lexicon.

Ignore Module: This module was designed to describe diagrammatically how the Dinka non-words would be added to the error list.

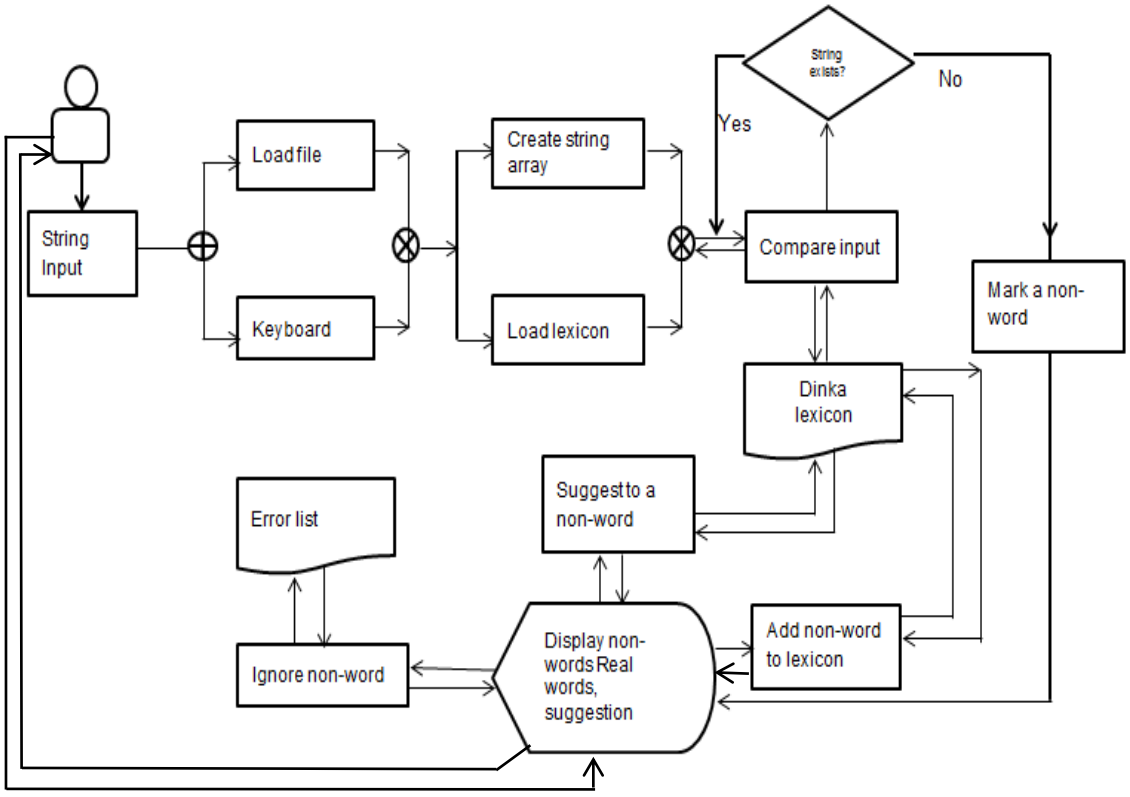

Figure 1. The model diagram of Dinka non-word detection prototype.

3. SYSTEM IMPLEMENTATION

The system was developed using java programing language. Netbean 7.1 was the development enviroment and notepad was used to create the Dinka lexicon file.

4. TESTING

Testing of the Dinka non-word detection protoype covered unit testing, intergration testing and system testing.

4.1. Unit testing

The unit testing covered Detection Module, Suggestion and Correction Module and Add Module.

4.1.1. Testing the Detection Module

The Dinka sentences with intentionally induced non-words were typed and the input were check by the algorithm to test if the prototype would detect the Dinka non-words as shown in figure 2.

Figure 2. The Detection Module testing

4.1.2. Testing the Suggestion and Correction Module

The edit distant of one was used to generate the suggestions for the Dinka non-words. The user would select and corrected the Dinka non-word as shown in Figure 3.

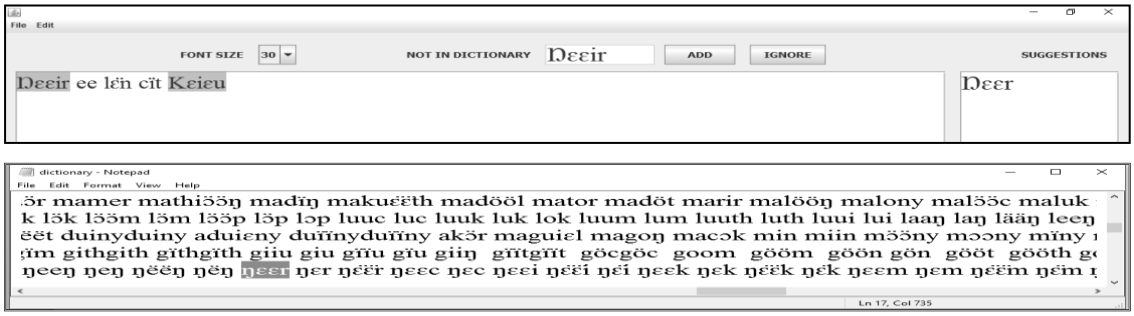

Figure 3. Testing the Suggestion and Correction Module

4.1.3. Testing the Add Module

The Add Module was tested by selecting a given Dinka non-word and added it to the lexicon. If the same word is re-typed, the system would not labelled as non-word again as shown in figure 4.

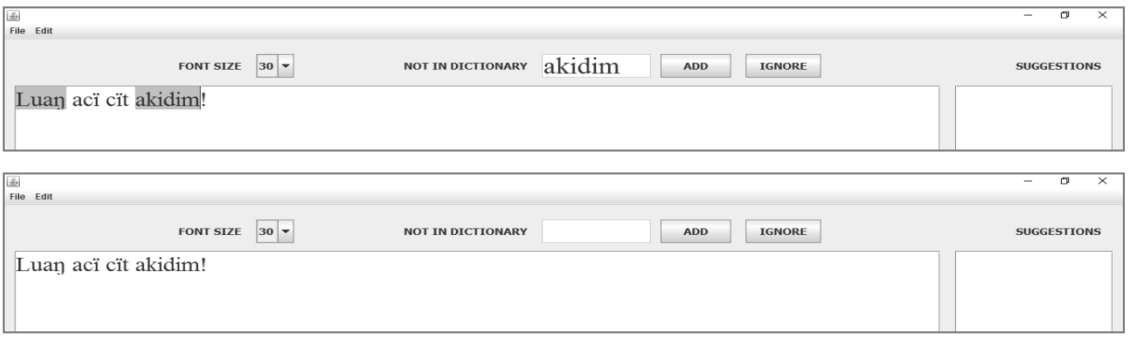

Figure 4. Testing the Add Module

4.2. System testing

The overall system testing covered three tests; testing the speed of non-word detection, the accuracy of detection and the accuracy of non-word detection.

4.2.1. The Speed of Detection

The speed of detection is defined as the number of tokens that the spellchecker processed per second and it is measured in Hertz, because it is considered as frequency. The speed of detection was calculated using the formula of [1] as shown.

$$\text{Frequency of Detection} = \frac{\text{Total number of tokens in a file}}{(\text{time taken to process the tokens})/1000}$$

Table 1. The number of tokens against detection time in milliseconds

Experiment	Number of Tokens	Time in millisecond	Frequency
1.	166	1,785	93
2.	225	1,785	126
3.	541	1,339	404
4.	763	1,785	427
5.	1,161	1,339	867
6.	1,473	1,339	1,100
7.	1,977	1,339	1,476
8.	3,927	1,785	2,200
9.	4,823	1,785	2,702
		Average frequency	**1,044**

In determining the speed of detection, a total of nine experiments were conducted. For each experiment, the frequency of detection was calculated by dividing the total number of tokens by execution time in seconds as shown in table 1. The average frequency of detection (FD) was calculated by adding up the total frequencies of the nine experiments divided by 9 and it was found to be 1,044 Hertz. Figure 5 showed the speed of Dinka non-word detection.

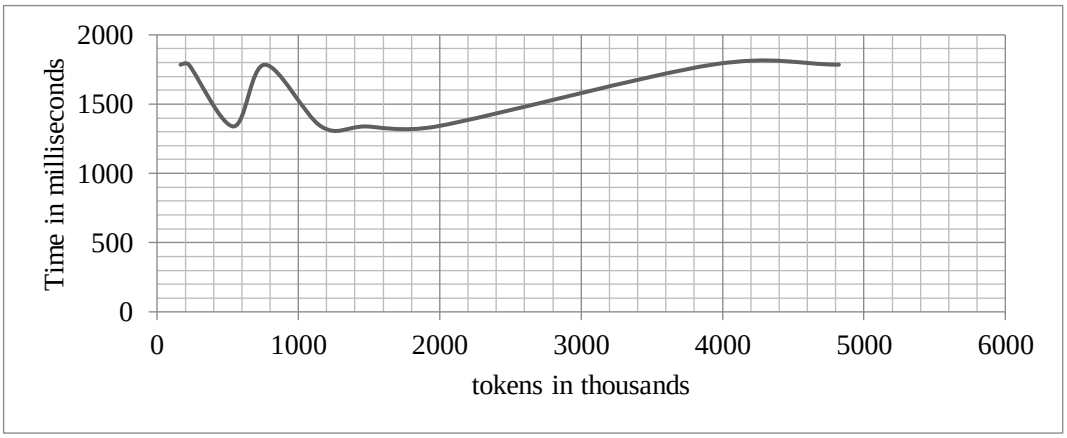

Figure 5. Testing the speed of detection

4.2.2. Accuracy of Detection

The Accuracy of Detection (detection of real words and non-words) is defined as the total number of True Positive (TP) detected, plus the total number of True Negative (TN) detected, divided by the total number of tokens (N) under investigation. The True Positives (TP) are the real words detected by the spellchecker. True Negative (TN) are the non-words detected by the spellchecker and (N) is the total number of tokens under evaluation.

The Accuracy of Detection for the Dinka spellchecker was computed using Confusion Matrix as described by [14]): N is the total number of tokens in the test file. In the experiment, all the True Positives (TP) were added to True Negatives, then divided by total number of tokens (N) of the tested file. The formula was written as;

Accuracy = (TP+TN)/N.

The experiments were conducted using three Dinka files to evaluate the accuracy of detection. A Dinka story entitled '*Meth Anyueth'* that literally means '*weaning a toddler'* was typed using the prototype. The prototype detected 247 Dinka words, where 47 words were detected as True Negative (TN), 9 words were detected as False Negative (FN) and True positive (TP) were detected as 191 words. The Accuracy of Detection was found to be 96.35%.

In the second experiment, the book of Mark of the Dinka New Testaments, chapter 4:1-40 was typed using the prototype. A total of 140 words were detected as True Negative (TN), 10 words were detected as False Negative (FN) and 596 words were detected as True Positive (TP). The accuracy of detection was found as 98.65%.

The third experiment was a Dinka story; "*Muɔ̈k meth*" where the prototype detected 126 words as True Negative, zero (0) words as False Negative and 351 words as True Positive. The accuracy of detection was found to be 99.16%.

Table 2. Accuracy of Detection

File no.	TN	FN	TP	number oftokens	Accuracy of detection
1	47	9	191	247	96.36%
2	140	10	596	746	98.67%
3	122	4	351	477	99.16%

4.2.3. Average Accuracy Detection

The average accuracy of detection was found to be 98.10%.

4.2.4. True Positive Rate (TPR)

The True Positive Rate is the rate at which the spellchecker can correctly detect the real words. In determining the TPR of the prototype, the data on table 2 was used in the formula TPR= TP/ (TP+FN);

By plugging in the values of file no. 1, TPR = (191)/ (191+9) the result was 100 %. Using the tokens of file no. 2, TPR = 596/ (596+10), the result was 98.35%. For test of the file no.3, TPR= (351+126)/ (477), the result was 98.87%. The average True Positive Rate of the three experiments was found as 99.10%.

4.2.5. True Negative Rate

The True Negative Rate (TNR) of a spellchecker indicates how accurate the tool can detect non-words. The same data on table 2 was used in the test. The formula used was; TNR=TN/ (FP+TN).

The tokens of the file no. 1 of table 2 produced the TNR of 83.93%, the tokens of the file no. 2 resulted to the TNR valued of 93.33% and the tokens of file no.3 gave TNR value of 96.83%. The average TNR of the three experiments was found to be 91.36%.

4.2.6. Testing the accuracy of non-word error detection

The accuracy (A) of non-word detection is the number of non-words (NWD) detected by the spellchecker divided by the total number of non- words (TNW) [1].The accuracy of non-word detection is calculated using the formula;

A= NWD / TNW;

Where NWD = Non words detected or True Negative (TN) and TNW=FP+TN. The results of the test is shown in table 3.

Table 3. Average non-word detection

File no.	NWD	FN	TNW	no. of tokens	Accuracy
1	47	9	56	247	83.93%
2	140	10	150	746	93.33%
3	122	4	126	477	96.83%
				Average non-word detection	91.36%

In determining the accuracy of non-word detection of the prototype, three Dinka files were tested (table 3). Firstly, the tokens of file no.1 test resulted into a non-word detection accuracy of 83.93 %. Secondly, the tokens of the file no. 2 test produced a non-word detection accuracy of 93.33% and thirdly, the tokens of the file no. 3 test resulted into a non-word detection accuracy of 96.8%. The average non-word detection accuracy of the three experiments was found to be 91.36%.

5. RESULTS

The research objectives of prototyping NLP Non-word detection prototype for Dinka using dictionary lookup approach were ; to collect and analyse system requirements, to create a Dinka lexicon from existing Dinka corpus, to design and develop a module to customize computer keyboard for Dinka text input, to design and develop Input Module, Detection Module, Suggestion and Correction Module, Add Module, Ignore Module , Display Module, to test and evaluate the prototype using a Dinka lexicon.

To fulfil system requirements, a total of 6,976 Dinka words were collected from the Dinka New Testament. The study found out that Dinka non-words arise majorly from transposition, substitution and deletion operations. User requirements were analysed using a use case diagram and system requirements were analysed using a sequence diagram. The method used to collect Dinka words from existing data agreed with the work of [15] which described word collection from articles.

The Dinka lexicon was created using a notepad with 6,976 Dinka words entries. The method used to create the Dinka lexicon agreed with the literature as described by [16] which explained lexicon design and creation.

The computer keyboard was customized in order to allow the Dinka characters input. This was achieved using append(), KeyTyped() and consume() java methods: 'ä' (a with diatrics), 'ë'(e with diatric) 'ŋ' (nga) , 'ɛ' (open e) , 'ɣ'(ram horn), 'ɔ' (open o) , 'ɛ' (open e) ,'ɛ̈ '(open e with diatric),'ö'(o with diatric), 'ï'(i with diatric), and 'ɔ̈'(open o with diatric) replaced the English characters: 's', 'q', 'f', 'v', 'z', 'x', '<', '>', '{' , '}' and ' ` ' respectively. The other letters from English remain the same on the QWERTY keyboard. This method agreed with the literature of [17] where Unicode and symbols were mapped onto the computer keyboard when customizing the computer keyboard.

The Input, Detection, Suggestion and Correction, Add, Ignore and Display Modules were successfully designed and developed. The method used in this research to design and develop Input Module, Detection Module, Ignore Module, Suggestion and Correction Module and Display Modules was supported by the literature of [2] which explains the design of spellcheckers.

The Input, Add, Ignore and Suggestion Module were tested if they were satisfying user requirements or not. The test for accuracy of detecting Dinka non-word and the speed of detecting Dinka non-words was also done.

The test on the Input Module found out that the module was able to allow the Dinka text input from a customized keyboard. The module also allowed the browsing of files containing Dinka text. The literature of [4] agreed with the testing method used in the prototype. The Detection Module was able to detect the Dinka non-words. The Suggestion and Correction Module gave suggestions to any detected Dinka non-words using the edit distance of one. The Add Module allowed the user to add non-words to the Dinka lexicon, while the Ignore Module allowed the user to ignore the detected Dinka non-words. The Display Module was a part of the Input Module and it allowed the user to see the Dinka non-words with their suggestions. The use of edit distance of one as explained by [1] supported the generation of word suggestions for the prototype.

The testing of accuracy of non-word detection (detection of non-words only) was found to be 91.36% and the accuracy of detection (detection of real words and non-words) of the prototype was found to be 98.10%.

These findings indicated that the Dinka non-word detection prototype was very accurate to detect Dinka non-words with the accuracy of 98.10%. It was also able to detect both the Dinka real words and non-words with the accuracy of 91.36%. Therefore the tool is capable of spellchecking the Dinka language, because these results were similar to the finding of [2] with the accuracy of 83.10%

The True Positive Rate (TPR) of the Dinka non-word detection prototype was found to be 99.10% (TPR is the rate at which the spellchecker detects real words). This finding indicated that the prototype was able to detect the Dinka real words to the accuracy of 99.10%

The True Negative Rate (TNR) of the non-word detection prototype for Dinka was found as 91.36 % (TNR is the rate at which the spellchecker detects the non-words). This finding indicated that the tool was able to detect the Dinka non-word to the accuracy of 91.36%. The TPR and TNR values of the study were in line with the work of [4] on ABC spellchecker.

The speed of detection of this prototype was found to be 1,044 Hertz and the processing speed of the computer used in the experiment was 2.0 Giga Hertz. This findings indicated that the detection algorithm was slow to detect Dinka non-words as the number of tokens increase in comparison with the work of [1].

6. CONTRIBUTION TO THE BODY OF KNOWLEDGE

The development of Dinka non-word detection prototype contributed to the existing body of knowledge in the following ways; The Dinka lexicon created in this research will be used by other researchers to develop machine translation systems using the created Dinka lexicon. The development of Dinka non-word detection will help in spellchecking when writing Dinka text. Finally the methods used in this prototype can be adopted to design spellcheckers for languages.

7. RECOMMENDATIONS FOR FURTHER RESEARCH

The Dinka lexicon created required further Dinka words collection to increase the suggestion accuracy. The dictionary lookup algorithm used in this prototype was not efficient enough as the findings indicated; increasing the number of tokens also increases the time of non-word detection. Therefore, any future work on this prototype will require an improvement on the speed of the non-word detection. The word suggestions was generated using the edit distance of one and therefore, any future work should improve the suggestion algorithm by trying the edit distance of more than one to increase the word suggestion accuracy. Any future work on this research should also addresses issues like automatic ignoring of numbers, uppercases and title cases. In addition, any future replication of this study should cover the development of the real word error detection.

ACKNOWLEDGEMENT

The author of the research thanks **Phillip Manyok Ph.D.** of the **American Public University** for financing the research. Extended thanks go to **Bugema University Kampala, Uganda and University of Juba, South Sudan** for offering the research environments.

REFERENCES

[1] Robert Ssali Balagadde & Parvataneni Premchand (2016). *Non Word Detection For Luganda*. Department of Computer Science & Engineering University College of Engineering, Osmania University Hyderabad, 500007, TS, India

[2] Baljeet Kaur, Harsharndeep Singh, (2015). Design and Implementation of HINSPELL -Hindi Spell Checker using Hybrid approach. *International Journal of scientific research and management.* Volume 3 Issue 2 pp1-5. ISSN (e): 2321-3418

[3] Nivedita S. Bhirud1 R.P. Bhavsar2 B.V. Pawar3 (2017), Grammar Checkers for Natural Languages: A Review, International Journal on Natural Language Computing (IJNLC) Vol. 6, No.4, August 2017

[4] Rajashekara Murthy S, Vadiraj Madi , Sachin D, Ramakanth Kumar P (2012), *A Non-Word Kannada Spell Checker Using Morphological Analyzer And Dictionary Lookup Method* , International Journal of Engineering Sciences & Emerging Technologies, June 2012. ISSN: 2231 – 6604 Volume 2, Issue 2, pp: 43-52

[5] Lars Bungum & Björn Gambäck (2010), *Evolutionary Algorithm In Natural Language Processing,* Norwegian Artificial Intelligent Symposium , Gjovik Norway

[6] Van Der, A.V. & 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.

[7] Diki-Kidiri, M. and A.B. Edema .(2003). *Les Langues Africaines Sur La Toile*. Cahiers du Rifal, 23: 5-32.

[8] Harouna Naroua & Lawaly Salifou. (2016).Design Spell corrector for Hausa Language. *International Journal of Computational linguistics*. Département de Mathématiqueset Informatique, Université Abdou Moumouni, Niamey, Niger. pp1-13
[9] Helene Fatima Idris (2004),*Modern Development In The Dinka Language*. University of Goteborg Department of oriental and Africa language
[10] Orville Boyd Jenkins (2013). *The Dinka of South Sudan*, retrieved April 26 2017 from http://strategyleader.org/profiles/dinka.html
[11] B. Remijsen and C. A. Manyang. (2009). *"Luanyjang Dinka,"* Journal of the International Phonetic Association, vol. 39, no. 1, pp. 113–124 124.
[12] Robert Ladd, Remijsen, Bert (2008). *The Tone System Of The Luanyjang Dialect Of Dinka.* Journal of African Languages and Linguistics 29: 173-213.
[13] Balone Ndaba, Hussein Suleman, C. Maria Keet & Langa Khumalo, (2015). *The Effects of a Corpus on isiZulu Spell checkers based on N-grams*. Department of Computer Science, University of Cape Town, University Avenue 18, Cape Town, 7701, South Africa
[14] Patrick Chi Fu Chan (2012). Final Year Project: A Spell Checker for the ABC System, University of Manchester UK.
[15] Bassam Hammo, Faisal Al-Shargi, Sane Yagi & Nadim Obeid, (2005). *Developing Tools for Arabic Corpus for Researchers*, university of Jordan
[16] J. Sinclair & Wynne, M (2005) *Text: Basic Principles*.(Ed.). Developing Linguistic Corpora: A Guide to Good Practice: 1-16. Oxford
[17] MPSTME, NMIMS (2011) *Information Retrieval in Multilingual Environment:* Phonetic and Semantic Along with Ontology Approaches for Hindi and Marathi, Mumbai India

AUTHORS

Manyok Chol David obtained Bachelor of Information Technology from Ndejje University, Kampala Uganda (2013). He did a Master of Science in Software Engineering (2017) from Bugema University, Kampala Uganda.
His fields of research include; Natural Language Processing, Software and Web Applications Development, Databases Systems and Machine Translation. He is currently a senior lecturer at the Department of Information Technology, University of Juba, South Sudan.

Balaggade Robert Ssali holds a Ph.D. from Osmania University, Hyderabad, India; MCSE (Microsoft Certified Systems Engineer) Certification Future Technology Ltd, Microsoft Certified Technical Education Centre, Kampala, Uganda; and a Master of Science in Systems Engineering from Moscow State University of Mining, Moscow, Russia. His fields of research include; Artificial Intelligence, Internet of Things and Big Data Analytics. He is currently a senior lecturer at Kampala International University.