Novel Approach for Arabic Spell-Checker: Based on Radix Search Tree

Rasha AL-Tarawneh
Al-Balqa' Applied University
Aqaba University College
Department of Applied Science
Aqaba-Jordan

Hatem S. A. Hamatta
Al-Balqa' Applied University
Aqaba University College
Department of Applied Science
Aqaba-Jordan

Hasan Muiadi
Al-Balqa' Applied University
Prince Abdullah Bin Ghazi
Faculty of IT
Dept. of Computer Science

ABSTRACT
The main aim of this study is to develop a spell-checker system for Arabic language. This is done by investigating the viability of applying the radix search tree approach. Through this scientific research several shrubs that represent Arabic characters will be built through serialized tracking of characters word where it can be added to the dictionary and with a special mark in the node that contains the last characters from each word; on other side during searching process, every word can be tracked character by character according suitable path inside its shrub, Accordingly, correct word can be recognized if and only if searching process locates some leaves during the traverse of the shrub. Otherwise, the word will be considered incorrect.

General Terms
Your general terms must be any term which can be used for general classification of the submitted material such as Pattern Recognition, Security, Algorithms et. al.

Keywords
Spell-Checker, Radix Search Tree, Computational Linguistic

1. INTRODUCTION
As it is known Arabic language is considered as one of the oldest language in the world, it has played an important role in the shared history of all Arabs and gained a high level of interest by more than one billion muslims to read and understand the message of Qur'an Islam's Holy book [8]. So the Arabs must bring their research and their interest in it to get computers performs useful tasks and operations such as processing of text.

The use of computer is spreading rapidly in the Arab world. Many computer software packages and applications have been available; but they are restricted to other languages. Arabic language suffers from the absent of its own software packages. For this reason, we aim in this study to develop a spell checker system for Arabic language which belongs to the area of computational linguistics as a part of artificial intelligence (AI)[2] which concerns about the construction of computer programs to process words and texts in natural language.

These days, applied computational linguistic systems are widely used in business and scientific domains for many purposes. Some of the most important ones among them is the spell-checker. The spell-checker system is an integral part of modern word processors, search engines, email client, and electronic dictionary [7]. Such system is useful for many people such as: students, business people, and professional writers [5]. The main objective of the spell-checker system is to flag words in a text that may not be spelled correctly [7]. This flagging process is done at the word level without considering the text context [1].

2. RELATED WORK
At the level of free software and to the borders of 2006, there was no free and functional Arab spelling checkers. Despite the many Arab attempts related directly or indirectly with the ArabEyes institution, the most important attempts are for the brothers Mohammad Zubair in the "Dua'alyi" Program, and Mohamed Samir in the "Baghdad" program. The delays in getting support for the "Dhad Language" in free softwares in general, and the lack of spelling checker, refer in mainly to distinguished software and linguistic characteristics, and it refers also to the rare competency and weak interest in free softwares at the levels of the region, the economics and at the university levels. At the end, the solution came through the gate of free softwares which are: the "Hunspell" spelling checkers adopted by the project Open Office program "Aspell". The two programs are developed for Latin languages, but with the addition of the support property "Unicode" & the Bidirectional property, they become suitable for support other languages than Latin [4].

2.1 Muaidi and Al-tarawneh Spell-Checker
Hasan Muaidi and Rasha Al-tarawneh attempted to develop a simple spell-checker for arabic language based on N-Gram scores. Several matrices are built to present the combination of the connected Arabic letters word. Each matrix deals with each word within the text separately and extracts the 2-gram set for it, that may have 1,0 or 2 according to the connection between the letters. Then it examines the value for each item in the 2-gram set. When the corresponding value for the item is zero then the spell-checker will consider the tested word as a wrong word otherwise the next corresponding value is checked until it reaches to the final value of last two letters[1].

2.2 Zerrouki-Balla Spell-Checker
Zerrouki and Balla concerned in Arabic language so they tried to add infixes and support circumcises with ignoring diacritics to open source spell-checkers Aspell and Hunspell [10].

2.3 Shaalan Spell-Checker
Due to the rich morphology and complex Arabic language, this makes a challenges for implementing an automatic spell-checker [9]. Shaalan et. al., attempt to developing an Arabic spelling checker program for solving this challenges and to recognize common spelling errors for standard Arabic and Egyptian dialects.

They have implemented the Arabic spelling checker tool using SICStus Prolog on IBM PC. The interface is built using Microsoft Visual Basic.

The first step in spelling correction is the detection of an error, there are two possibilities:

1. The misspelled word is an isolated word (Non-word).
2. The misspelled word is a valid word . (As writing < **نال** > instead of < **مال** >).

2.4 Haddad-Yaseen Spell Checker

Bassem Haddad and Mustafa Yaseen [3] proposed a hybrid model for spell-checking and correcting of Arabic words, based on semi-isolated word recognition. In their work, the error of Arabic word is classified into three types; typographic, cognitive and phonetic errors. Each one of these error types is categorized into a single error or multi error.

3. METHODOLOGY

Arabic Corpus

One of the main challenges for the researchers and the developers tools for the Arabic language is the absence of public free corpus\footnote{This is for the best of our Knowledge.}. The corpus which is used in this research study is adapted from Muaidi PhD thesis[1]. This corpus (hereafter refer to as Muaidi Corpus) is implemented and compiled in 2005-2008 during the Muaidi's PhD study at De Montfort University in UK [6].

The following items highlight the features of Muaidi Corpus:

1. The Muaidi Corpus consists of 848,779 Arabic words (including redundant words) written in Modern Standard Arabic (MSA).
2. The words cover a wide range of knowledge subjects.
3. The minimum word's length in this corpus is 1; while the maximum word's length is 25.
4. The total number of words which have a length greater than or equal 4 and less than or equal 12 is 596,916, about 101,987 of them are word-types.
5. The Muaidi Corpus is an annotated corpus in which the words in this corpus have morphological features such as roots, stems, affixes and part-of-speech tags.

In summary, the corpus which is used in the current study consists of 101,987 word-types. These words are used to train and test the Radix tree technique.

3.1 Developed Radix Tree

By radix tree approach 28 trees are built, each tree presents one Arabic letter. The trees are built by keeping a track of word's letters, and a special mark in the node which contains the last letter of the word. The search operator is executed by tracking the word letters using appropriate path in the right tree of the 28 trees; if the word finished and the tracking reached to the node with a special mark then this word is considered as a correct word otherwise it is considered as incorrect word.

The spell-checker based on the radix tree is composed of two phases:

First, Building the radix trees phase.

As it is mentioned the final tokens should be prepared to be stored in the trees. The process is applied by tracing the following steps:

1. After being connected with the corpus, each token is taken individually.
2. Each token is split into separate letters.
3. Each letter is taken individually to be stored in a tree according of it's order in the token, so the first letter is stored as a root, while the second letter is stored as a child of the same root at level one in the tree, also the third letter is stored as a child of the previous letter node... and so on.

Consequently, through storing process the root may be created by some of tokens which are shared with the same first letter of current token, in this case it will not be stored another time, but it is going to the second letter in the token and search for it in the children of the root, if does not exist it will be stored; else it will be going to the third letter, and continue until reach to the final letter in the token where a node must be store a final letter with a special mark.

The following example clearly shows how to build the radix tree from the tiny corpus in Figure 1.

أقلام، أحمد ، أحمر ،بسملة ،بائع ، باب ،تبدع ،تاج ، يسار ، يزيد

Fig 1: Tiny Corpus Token Inputs

The first token is < **أقلام** > which is split into five letters and is stored in the initial tree as it is shown in Figure 2-a. The next token is < **أحمد** >, the first letter has been stored already from the previous token, so the next letter is checked in the children of root node, as it is shown in Figure 2-b. Since it is not stored in the tree it will be added as a child of the root node and continue stores the rest letters. The third token is < **أحمر** >, the first three letters have been stored in the previous two tokens so they will not be stored another time, only the last letter of it will be added to the tree, as it is shown in Figure 2-c. The letter (ر) is assigned as an extra attribute which is (*) to indicate that this letter is the last one in the token.

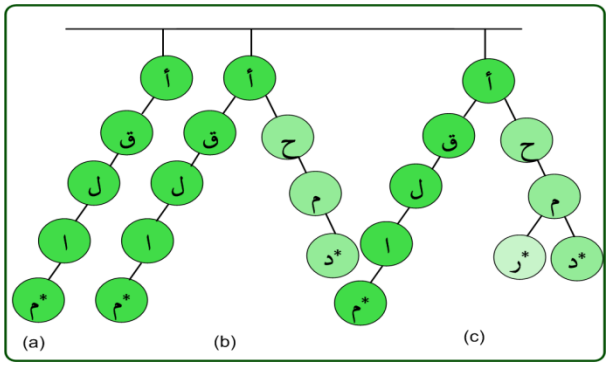

Fig 2: Initial Tree for First Three Tokens

When the process reaches to the Arabic word < **بسملة** > , then a new tree should be created and value of the root node is the letter (**ب**) as shown in Figure 3-a, then stores < **باب/بائع** >, in the same tree as it is shown in Figure 3-b and 3-c. A new two trees are created with roots (**ي/ت**) respectively to store the rest of tokens.

[1] Dr. Hasan Muaidi, AL-Balqa' Applied University.

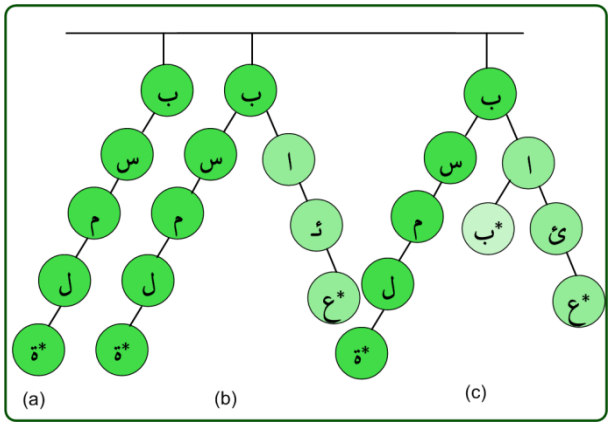

Fig 3: Initial Tree for Second Three Tokens

Second, Spell-Checking Phase

After generating the radix trees for all the words in the corpus, the tested text is entered in the text box. Each word is processed separately and the process of spelling is executed once the button of < **التدقيق الاملائي** > is clicked, The incorrect words are colored by red while the correct words are black.

The spelling process is executed as follows:

1. The first letter of the tested word is extracted in order to determine the suitable radix search tree. This first letter is the root node of the tree.
2. The second letter is examined to check whether it is one of the children of the root, if it is not, the tested word is considered as incorrect word and coloring by red. Otherwise the same checking process is continued for the rest letters until it reaches to the last letter.
3. The last letter node is checked against the (*) attribute. If it has this attribute, then the tested word is considered as a correct word otherwise, it is considered as incorrect word and colored by red.

Example

Suppose a tiny corpus is entered in the text-box as shown in Figure 4. The first token is < **أقلام** >, it is split into five letters, the first letter is compared with all the root values in the generated trees. The roots have the keys (أ , **ب** , **ت** , **ي**). In the case of the current example, the tree with the root key (أ) is processed see Figure 2-a.

The second letter is taken and search on it over the children of the same root, if it is existed in the children then the third letter is taken; else the token will be colored by red, in this case the letter (**ق**) is stored in the tree so the searching process is continued in the same track until it is reached to the final letter which must has the (*) attribute and it is satisfied in this token so the word < **أقلام** > is considered as correct word.

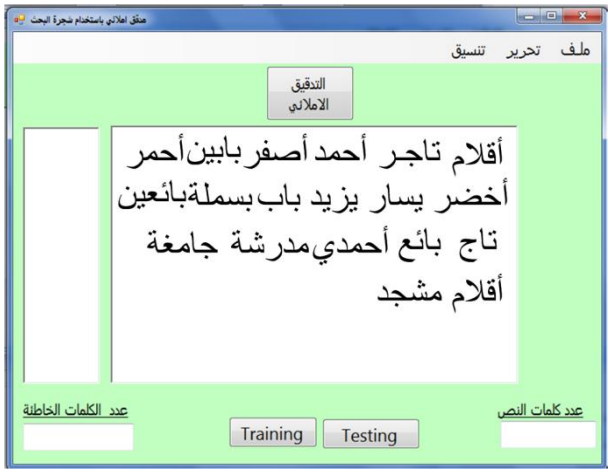

Fig: 4 Spell-Checker Inputs Using Radix Tree

The second token is < **تاجر** >, it begins with the letter (**ت**), so the checking process is stabled in the tree in Figure 5-b which has a root node with (**ت**) value. The first 3 letters are existed in the tree, but the last letter does not exist in the tree so this word is colored by red.

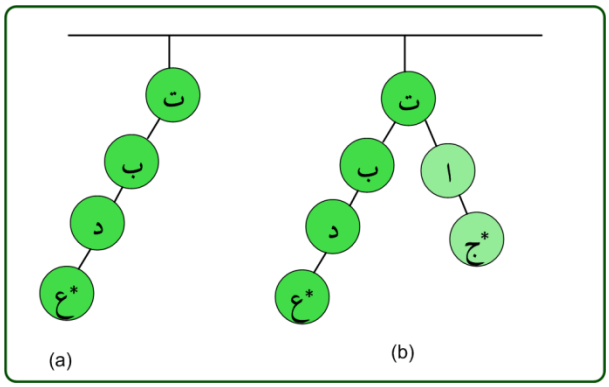

Fig 5: Initial Tree for Third two Tokens

The third token is < **أحمد** >, the search tree should has a root with (أ) value, so the checking process is stabled in the tree in Figure 2-c. All the letters of this word are existed in tree but the last one is stored without the (*) attribute since it is not in the tiny corpus; so this token is colored by red. The result of this example is shown in Figure 6.

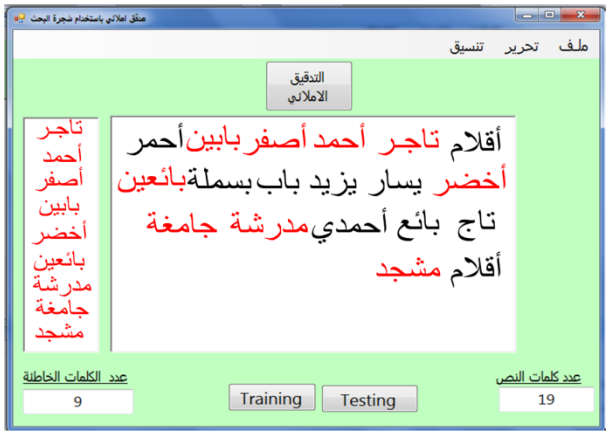

Fig 6: The Result of Spell-Checker Using Radix tree

So the words in this method have a red color in two possible cases:

- It is a correct Arabic word, but it is not stored in the tree.
- It is already an incorrect Arabic word.

4. EXPERIMENTAL RESULTS

Two stages are presented to test the results and measure the system performance.

4.1 The Training Stage

As mentioned, the size of Muaidi Corpus consists of 101,987 words. These words are considered as a dataset to train and to test the performance of the developed spell-checker. This dataset is divided into two unequal parts, bulk part (70%) which is used as a training dataset for the training stage and the remaining part (30%) is taken as a testing dataset for the testing stage.

The training stage depends on training set to train the developed Radix tree spell-checker. While the testing stage depends on testing set to evaluate the performance of the developed Radix tree spell-checker. The training dataset consists of 71,390 Arabic words. While the testing dataset consists of 30,597 Arabic words. The evaluation process is done on an Intel core 2 dual processor with a speed 1.80 GHz. The RAM capacity is 1,016 GB and the operating system is WINDOWS XP.

The methodology of evaluation the current research study is organized as follows:

(1) The training dataset is used to build the radix trees.

(2) To assure the ability of learning, the training dataset is used to train the radix tree technique.

(3) Testing dataset (which is unseen data) is used to check the performance of the spell-checker.

(4) The words which are considered as incorrect words from the previous step are reentered to the system.

(5) The performance of the spell-checker is recomputed again.

(6) The implemented evaluation methodology for the developed spell-checker is based on the ability of it to successfully spell the correct Arabic words.

The developed radix tree technique is correctly spelled 100% of the words in the training dataset. Table 1 summarizes the evaluation of the results in the training dataset. While Figure 7 demonstrates these results in a column format chart.

Table 1. The Evaluation of Results in Training Dataset (Radix tree Approach)

Number of words	71.390
Number of correctly words	71.390
Number of colored words	0.0
Success rate	100%

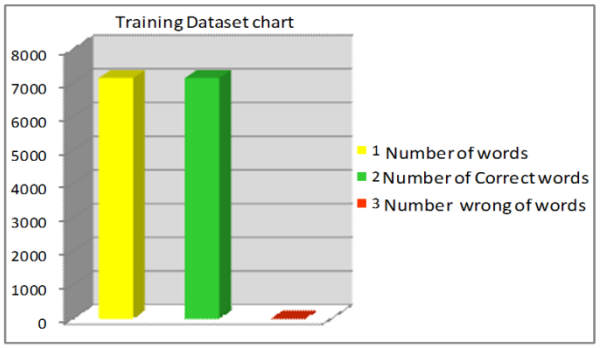

Fig 7: The Evaluation of the Training Dataset for Radix Tree Approach

4.2 The Testing Stage

In testing stage a hidden dataset (testing dataset) is used to indicate the accuracy of the developed spell-checker system. As mentioned before the testing dataset consists of 30,597 Arabic words. To test the performance of the radix tree technique, the accuracy is calculated using the success rate measure $\mathbf{S_R}$. This measure compatible for the developed spell-checker with checking the error words and colored them. Success rate measure is calculated as shown in Equation 1.

$$\mathbf{S_R} = \frac{\boldsymbol{CW}}{\boldsymbol{N}} \mathbf{100\%} \quad (1)$$

Where:

$\mathbf{S_R}$ = The success rate.

CW = The number of correctly words.

N = The size of the testing dataset.

The experiment is performed on the developed spell-checker system using the testing dataset and the success rate is obtained 24.24%. Table 2 summarizes the evaluation of the results in the testing dataset. While Figure 8 illustrates these results in a column format.

Table 2 The Evaluation of Results in Testing Dataset (Radix tree Approach)

Number of words	30,579
Number of correctly words	07,471
Number of colored words	23,126
Success rate	24.24%

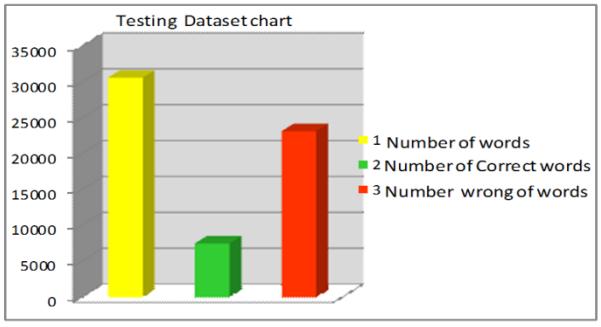

Fig 8 The Evaluation of the Testing Dataset for Radix Tree

4.3 Causes of the Errors

The difference in the accuracy between the two stages (training and testing) is refers to the number of words in each dataset. When the size of the dataset is increased the accuracy will be increased spontaneously; and the system has the ability to recognize a great number of words and automatically the error rate is decreased.

4.4 Discussion of the results

The developed spell-checker system accuracy reached 100\%, using the training dataset. While the accuracy of the developed system reached to 24.24% using the testing dataset. This difference back to the variant between build data and test data; which means that the testing data is unseen from the system, so if we use the testing dataset in rebuilding and retest them again using this dataset, the accuracy will jump tremendously. It jumps from 24.24% to 100% . Table 3 clears this while Figure 9 illustrates these results in a columns format.

Table 3 The Evaluation of Results When Rebuild Testing Dataset (Radix Tree Approach)

Number of words	30,579
Number of correctly words	07,471
Number of colored words	23,126
Success rate	24.24%

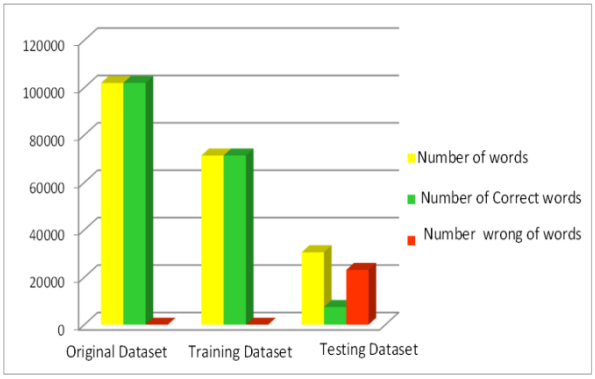

Fig 9 The Difference between the Results Before and After Add Unseen Data(Radix Tree Approach)

The above discussion is clarified that the accuracy is mainly depends on the number of words in the corpus, so to increase the accuracy of the developed spell-checker the number of words should be increased. The overall accuracy of the developed spell-checker based on radix search tree approach is reached to 100% using the whole data in Muaidi corpus dataset. Table 4 summarizes all these results.

Table 4 The Overall Evaluation of Results (Radix Search Tree Approach)

	Training Data	**Testing Before Rebuild**	**Testing After Rebuild**	**All Data**
Number of words	71,390	30,597	30,597	101,987
Number of correctly words	71,390	7,471	30,597	101,987
Number of colored words	0	23,126	0	0
Success rate	100%	24.24%	100%	100%
Execution time (all data)	-	-	-	1,020,000 ms
Execution time (one Word)	-	-	-	10.00 ms

5. CONCLUSION

After developing a spell-checker using radix search tree technique. It is trained using training dataset and the accuracy is calculated for the developed spell-checker accordingly. Thus, the overall accuracy almost reached to 100% ; it provides a high accuracy. The above evidences are sufficient to make sure that the radix search tree could be used efficiently to build a spell-checker for Arabic language. The future scope will convey to find new techniques that can keep spell-checker as highly efficient and accurate as possible.

6. REFERENCES

[1] H Muaidi and R Al-Tarawneh. Towards Arabic spell-checker based on n-grams scores. International Journal of Computer Applications, 53(03):5, September 2012.

[2] Anna Feldman. Computational Linguistics: Models, Resources, Applications. ISBN, 2004.

[3] B. Haddad and M Yaseen. Detection and correction of non-words in arabic: A hybrid approach. International Journal of Computer Processing of Oriental Languages, 30, 2007.

[4] Mohammed kabbani. The arabic spell-checker dictionary from ayaspell project. Technical report, Prix special des troisiemes rencontres africaines du Logiciel Libre, 2008.

[5] S.K Kataria and Sons. The Design and Analysis of Algorithms. N. Upadhyay, 2008.

[6] Muaidi.Hasan. Extraction Of Arabic Word Roots: An Approach Based on Computational Model and Multi-Backpropagation Neural Networks PhD thesis, De Montfort University - UK, 2008.

[7] H Satori, M Harti, and N Chenfour. Arabic speech recognition system using cmu-sphinx4. CoRR 0704.2201, 2007.

[8] Zeina Seikaly. The arabic language: The glue that binds the arab world. AMIDEAST, 2007.

[9] S. Khaled, A. Amin, and G. AbdAllah. Towards automatic spell checking for arabic. In Language Engineering, 2003.

[10] T. Zerrouki and A. Balla, Implementation of infixes and circumfixes in the spellcheckers. In Proceedings of the Second International Conference on Arabic Language Resources and Tools, 2009.