Journal of Theoretical and Applied Information Technology
30th June 2016. Vol.88. No.3
© 2005 - 2016 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPLEMENTATION OF IMPROVED LEVENSHTEIN ALGORITHM FOR SPELLING CORRECTION WORD CANDIDATE LIST GENERATION

[1]HANAN NAJM ABDULKHUDHUR, [2]IMAD QASIM HABEEB, [3]YUHANIS YUSOF, [4]SHAHRUL AZMI MOHD YUSOF

[1]Master student, School of Computing, Universiti Utara Malaysia, Malaysia

[2]PhD student, University of Information Technology & Communications, Iraq

[3]Prof. DR, School of Computing, Universiti Utara Malaysia, Malaysia

[3] DR, School of Computing, Universiti Utara Malaysia, Malaysia

E-mail: [1]hanan_nagem@yahoo.com, [2]emadkassam@yahoo.com, [3]yuhanis@uum.edu.my, [4]shahrulazmi@uum.edu.my

ABSTRACT

Candidates' list generation in spelling correction is a process of finding words from a lexicon that are close to the incorrect word. The most widely used algorithm to generate the candidate list is the Levenshtein algorithm. However, the algorithm consumes high computational cost, especially when there is a large number of spelling errors. The reason is that calculating Levenshtein algorithm includes operations that create an array and fill the cells of this array by comparing the characters of an incorrect word with the characters of a word from a lexicon. Since most lexicons contain millions of words, such operations will be repeated millions of times for each incorrect word in order to generate its candidates' list. This study proposes an improved Levenshtein algorithm that reduces the operation steps in comparing characters between the query and lexicon words. Experimental results show that the proposed algorithm outperformed the Levenshtein algorithm in terms of processing time by having 32.43% percentage decrease.

Keywords: *Levenshtein Algorithm, Processing Time, Word Candidate List Generation, Spelling Correction, Edit Distance*

1. INTRODUCTION

Candidates' list generation requires too much time when there is a large number of spelling errors [1, 2]. Some researchers speed up the process of generating candidates' list by using fast approximate distances, such as N-gram distance [3]. However, approximate distances can produce in some cases incorrect candidates list. Approximate distances are used when a correction process allows a tolerance of some errors to correct a large number of errors quickly. Also, in some systems, they are used to generate candidates' list while a human will select the best candidate manually [1]. Levenshtein algorithm (LA) [4, 5] is one of the exact algorithms, and it is widely used to generate a list of candidates for incorrect words from a lexicon [2, 6-9]. However, it requires high computational time, especially when there is a large number of spelling errors [1, 2, 10].

In general, Levenshtein algorithm is designed for measuring edit distance [4]. The term "edit distance" is used for calculating the difference between two strings. In other words, it counts the minimum number of operations required to transform one string to another [4, 11]. The operations of Levenshtein algorithm are performed on a single symbol or a single character, and they consist of insertion, deletion, and substitution. Each operation of a single symbol is considered as a single edit [12]. For example, given a query of "*csp*" that is a non-English word, the Levenshtein algorithm needs to perform one substitution to transform the word into "*cup*" which is an accepted English word [9]. Hence, processing time of LA will increase as its requires the creation of array and filling up each cell in the array by comparing characters of an incorrect word with characters of a word from a lexicon [1, 10]. With this, the Levenshtein algorithm when used in generating candidates for spelling correction requires a million calculations for each incorrect word because most lexicons contain millions of words [13, 14]. Each cell value in Levenshtein array needs eight operations: compare (3), add (3) and assign (2) [1]. Such inefficiency motivates for Levenshtein algorithm improvement that reduces the operational process without affecting its accuracy.

The study was organized into five main sections: section 1 presented the introduction. Section 2 discusses related work on Levenshtein algorithm. Section 3 explains the proposed algorithm and its implementation. In section 4, experimental results and discussion are presented. The last section includes conclusions and future work of our research.

2. RELATED WORK

Several solutions have been proposed to solve the problem of generating candidates list for a large number of errors, but most of them are based on using approximate distances such as n-gram distance [3]. The approximate distance may sometimes produce incorrect candidates list [1]. They are used when a correction process allows a tolerance of some error to correct a large number of errors quickly [14]. Other solutions are based on using other exact algorithms such as hamming distance algorithm [11]. However, each algorithm suffers different limitation, and selecting the appropriate algorithm depends on where it will be used.

Several works improve Levenshtein algorithm to be reliable for specific purposes. For example, Pal and Rajasekaran [15] improved Levenshtein algorithm to find motif in a set of biological strings. The motif is a substring that appears in a set of input biological strings. The characters in motif can be not neighbored. For example, the string "GT***G" is a motif. The symbol "*" can be referred to any character in sequence. However, other characters should appear in all input strings with the same context. This improvement makes Levenshtein algorithm suitable for finding motif in biological strings. However, it becomes slower than the original. Navarro, et al. [16] improved Levenshtein algorithm to be faster for music information retrieval by ignoring deleting and inserting operations from the calculation. The author only considers substitution operation because of characteristics of music pieces. These characteristics do not allow deleting and inserting operations in music pieces. Therefore, the author ignores them from the calculation. The improved Levenshtein algorithm is faster and reliable only for measuring the difference between two pieces of music. Other improvements in Levenshtein algorithm include making it approximate in order to be executed quickly [17-21]. On the other hand, the original Levenshtein algorithm [5] is still in use to generate candidates' list even if it takes too much time for a large number of errors. This indicates that improvement in Levenshtein algorithm in terms of processing time is still an open problem [2, 7, 10, 22].

3. PROPOSED TECHNIQUE

This study will refer to the improved Levenshtein algorithm as ILA-OT. The term ILA-OT represents improved Levenshtein algorithm using a proposed operational technique. This proposed technique reduces the operations required to measure cells' values in Levenshtein array. The ILA-OT aim is to remove the first row and the first column of LA array. In addition, it can predict the cell values in the second row, second column, third row, and third column in LA array. This is because there is a context to measure values in these cells until a shared character in them is identified. Once the context is changed, it will measure cells values in these rows and columns based on the new context. The context will only be changed once, and it will not change if there is an additional shared character in them.

Since an average word length for the English language is 5 characters [23], and by assuming the symbols c1, c2, c3, c4, and c5 refer to the characters in the English word, then Levenshtein array will be in the size of 36 cells (6 rows * 6 columns). Therefore, 27 cells from 36 will be affected by the proposed technique of this study as shown in Figure 1. This will reduce the processing time of Levenshtein algorithm execution. Details of the rules based on the proposed ILA-OT are presented in the following subsections.

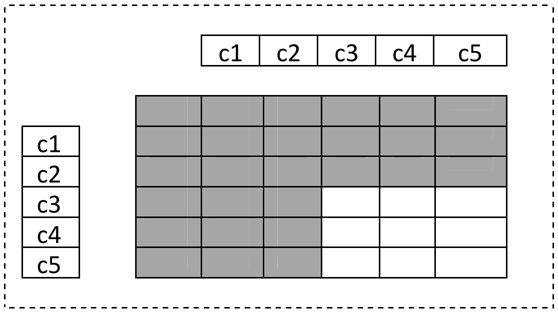
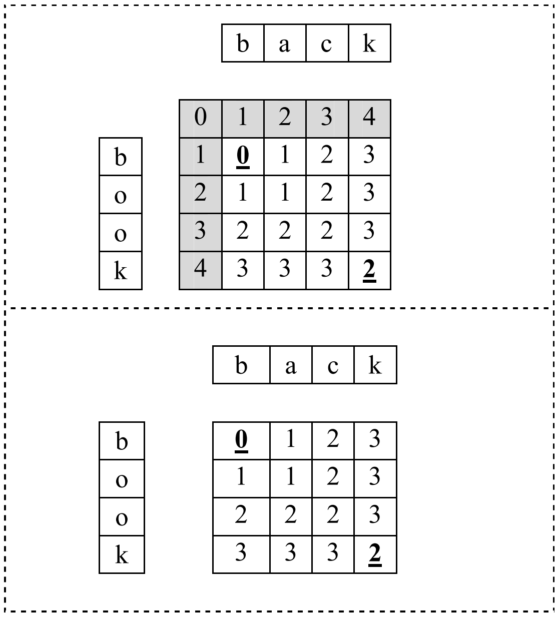


Figure 1: Cells Affected By Three Rules Proposed Of This Study

3.1 ILA-OT Rules

The first rule removes the first row and first column of Levenshtein array. This is because values of these cells will not be used to measure values of second row and second column in Levenshtein array. Figure 2 shows the Levenshtein array before and after applying the first rule.



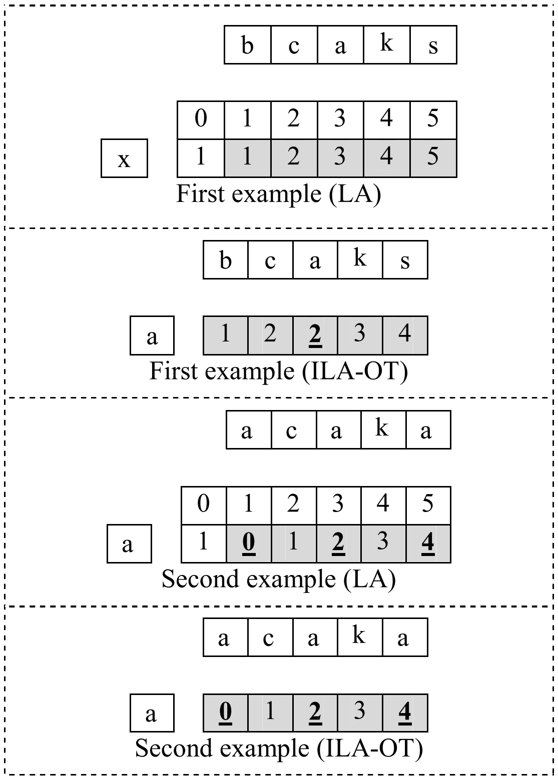
		b	a	c	k
	0	1	2	3	4
b	1	$\underline{0}$	1	2	3
o	2	1	1	2	3
o	3	2	2	2	3
k	4	3	3	3	$\underline{2}$

	b	a	c	k
b	$\underline{0}$	1	2	3
o	1	1	2	3
o	2	2	2	3
k	3	3	3	$\underline{2}$

Figure 2: Levenshtein Array Before (Up) And After (Bottom) Applying The First Rule

This study will use the term d(i, j) to refer to any cell in ILA-OT array, where the character "i" represents the position of a row in ILA-OT array and the position of each character in second token "t", while character "j" represents the position of a column in ILA-OT array and the position of each character in first token "s". Furthermore, starting from this section, this study will refer to the second row and second column as the first row and first column in the ILA-OT array. This is due to the employment of the first rule. The second rule is applied to measure cells values of the first row in the ILA-OT array (second row in LA array) and first column in the ILA-OT array (second column in LA array). This rule requires fewer operations than the ones in original LA.

Figure 3 shows two examples that explain the implementation of the second rule by comparing values of the second row in LA array with values of the first row in ILA-OT array. The first example in Figure 3 shows that if there is no shared character between characters of first token "s(j)" and the first character in second token t(0), then all cells of the first row will take the values of the context (1+j) in the ILA-OT array. Note that the order of characters in any string is 0 for the first character, 1 for the second character, 2 for the third character, and so on. The second example shows that if there is a shared character, then all cells value of the first row in the ILA-OT array, starting from the cell that has shared a character, will take the value of j until the last cell in the first row without additional comparison. This is because the first shared character will assign shared cell value to the j instead of (j+1). Furthermore, the context will be constant and it will not change if there is an additional shared character in the same row.



		b	c	a	k	s
	0	1	2	3	4	5
x	1	1	2	3	4	5

First example (LA)

	b	c	a	k	s
a	1	2	$\underline{2}$	3	4

First example (ILA-OT)

		a	c	a	k	a
	0	1	2	3	4	5
a	1	$\underline{0}$	1	$\underline{2}$	3	$\underline{4}$

Second example (LA)

	a	c	a	k	a
a	$\underline{0}$	1	$\underline{2}$	3	$\underline{4}$

Second example (ILA-OT)

Figure 3: Second Rule Examples For The First Row

Figure 3 also shows that if the proposed second rule is performed, the values of the second row in LA array match with the values of the first row in ILA-OT array. However, the operations required to measure cells values in the first row of ILA-OT is less than operations required to measure cells values in the second row in LA. The second rule can also be applied to measure cells value of the first column in the ILA-OT array (second column in LA array) with a simple modification. The modification is to replace j by i.

Lastly, the third rule can be applied to measure cells values of the second row in ILA-OT array (third row in LA array) and second column in the

ILA-OT array (third column in LA array) as shown in the three examples of Figure 4. Note, only gray cells in Figure 4 represent the third row in LA array and second row in ILA-OT array. The third rule also finds a context in measuring values of second row and second column in ILA-OT array. However, the context of the third rule is different from the context of the second rule. The context of the third rule needs values of first row and first column to measure values of second row and second column while the context of the second rule does not need values from any row or column.

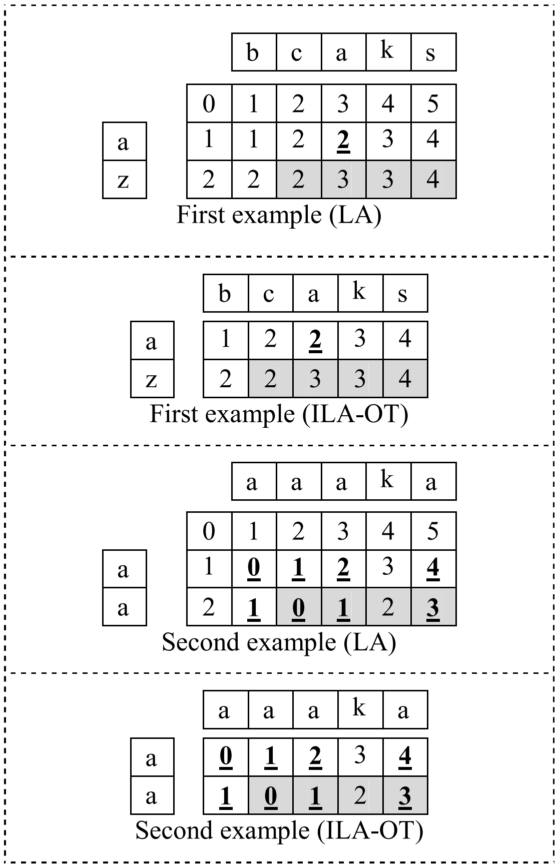


Figure 4: Third Rule Examples For The First Row

Example one in Figure 4 shows that if there is no shared character between the characters of first token "s(i)" and second character in second token t(1), then all values of cells in the second row will take the values of the context: (1+d(0, j-1)). The second example shows that if there is a shared character, then the shared cell will take the value of d(0, j-1), while all cells values of the second row, starting from the cell that follows the shared cell, will take the value of (1+d(1, j-1)) until the last cell in the second row without additional comparison. Note that the context of (1+d(1, j-1)) will be constant and it will not be changed if there is an additional shared character in the same row.Figure 4 also shows that if the proposed third rule is performed, the values of the third row in LA array are matched with the values of the second row in ILA-OT array. However, the operations required to measure cells values in the second row of ILA-OT is less than operations required to measure cells values in the third row in LA for the reasons mentioned in the next section. The third rule can also be applied to measure cells values of the second column in the ILA-OT array (third column in LA array) with a simple modification. The modification is to replace j by i.

3.2 Comparison between Operations of LA and ILA-OT

In this section, a comparison between LA and ILA-OT regarding the operations required to measure each cell value in their arrays is presented. The comparison is based on mathematical expression of both LA and ILA-OT. The equations 1 and 2 show how can measure each cell value in the first row and first column respectively for LA array, while both equations 3 and 4 show how can measure each cell value for other rows and columns in LA array. The equations 5, 6, 7 and 8 show how can measure each cell value in the first row, first column, second row, and second column respectively for the ILA-OT array.

$$d(0, j) = j \quad \text{for LA} \tag{1}$$

$$d(i, 0) = i \quad \text{for LA} \tag{2}$$

$$\cos t = \begin{cases} 0 & , \text{if } t(i) = s(j) \\ 1 & , \text{ if } t(i) \neq s(j) \end{cases} \tag{3}$$

$$d(i, j) \text{ for LA} = \min \begin{cases} 1 + d(i, j-1) \\ 1 + d(i-1, j) \\ \text{cost} + d(i-1, j-1) \end{cases} \tag{4}$$

$$d(0, j) = \begin{cases} j+1 & , \text{before } t(0) = s(j) \\ j & , \text{ if } t(0) = s(j) \text{ or after it} \end{cases} \tag{5}$$

$$d(i, 0) = \begin{cases} i+1 & , \text{before } t(i) = s(0) \\ i & , \text{ if } t(i) = s(0) \text{ or after it} \end{cases} \tag{6}$$

$$d(1,j) = \begin{cases} 1 + d(0, j-1) & \text{, before } t(1) = s(j) \\ d(0, j-1) & \text{, if } t(1) = s(j) \\ 1 + d(1, j-1) & \text{, after } t(1) = s(j) \end{cases} \quad (7)$$

$$d(i,1) = \begin{cases} 1 + d(i-1, 0) & \text{, before } t(i) = s(1) \\ d(i-1, 0) & \text{, if } t(i) = s(1) \\ 1 + d(i-1, 1) & \text{, after } t(i) = s(1) \end{cases} \quad (8)$$

As mentioned previously “d” represents the array required to measure edit distance, while the symbol “i” refers to the position of a row in d array and symbol “j” refers to the position of the column in “d” array. Also, as mentioned previously, second row and second column in LA array mean first row and first column in ILA-OT array, while third row and third column in LA array mean second row and second column in ILA-OT array. This is due to the removing first row and first column from the ILA-OT array as described in the previous section. Note this study ignores increasing operations required to increase i and j from the calculation because they are almost same for ILA-OT array and LA array. Furthermore, it ignores any single operation does not require a loop to measure it in order to make the comparison of operations easier.

Equations 1 and 2 show that it need one assigning operation to fill each cell value in the first row and first column of LA array. To fill other rows and columns in LA array, it requires measuring equation 3 and equation 4. To identify the value of cost in equation 3, it needs two operations: one comparing operation and one assigning operation, while to identify the value of d(i, j) in equation 4, it needs six operations: three summing operations, two comparing operations and one assigning operation. The total number of operations is 8 for each cell in LA array excluding first row and first column. On the other hand, equations 5 and 6 show that each cell value in the first row and first column in the ILA-OT array (second row and second column in LA array) requires three operations: one summing operation, one comparing operation and one assigning operation. Furthermore, it requires fewer operations if there is a shared character between characters of first token “s(j)” and the first character in second token t(0). This means the proposed technique will decrease 5 or more from 8 operations needed in a normal way to measure each cell value in the first row and first column.

On the other hand, equations 7 and 8 show that each cell in the second row and second column in the ILA-OT array (third row and third column in LA array), also requires three operations: one summing operation, one comparing operation and one assigning operation. Furthermore, it also requires fewer operations if there is a shared character between characters of first token “s(j)” and second character in second token t(1). This means the proposed technique will also decrease 5 or more from 8 operations needed in a normal way to measure each cell value in the second row and second column.

To summarize above, ILA-OT decreases the operations of cells in LA array in six positions. The first position is to remove all cells of the first row in LA array with their operations while the second position is to remove all cells of the first column in LA array with their operations. The third position is to decrease operations of the second row in LA array by almost (5 * number of cells in the second row) while the fourth position is to decrease operations of the second column in LA array by almost (5 * number of cells in the second column). The fifth position is to decrease operations of the third row in LA array by almost (5 * number of cells in the third row) while the sixth position is to decrease operations of the third column in LA array by almost (5 * number of cells in the third column).

4. EXPERIMENTS RESULTS

Experiments have been conducted to compare the accuracy and processing time (PT) of the ILA-OT algorithm against the LA employed in [2, 7, 22]. The testing dataset contains different word lengths, ranging from 3 to 12 [24]. The dataset has different words length in order to measure the impact of improved Levenshtein algorithm for a different length of characters. Each length of characters contains 1000 words. Equation 9 is used to measure the percentage decrease (PD) in processing time [23] while Equation 10 is used to measure the accuracy between LA and ILA-OT [14].

$$PD = \frac{PT(LA) - PT(ILA\text{-}OT)}{PT(LA)} * 100 \quad (9)$$

$$Acc. = \frac{\text{Number of equal edit distances}}{\text{Total number of comparisons}} * 100 \quad (10)$$

Note the accuracy was not measured individually for each algorithm, but it measured depending on edit distances of both algorithms. This means two words will be sent in each comparison for LA and for ILA-OT to calculate edit distance for each algorithm, then the variable "*number of equal edit distances*" in equation 10 will increase by one for each comparison if the edit distances of both LA and ILA-OT are equal. The experimental results are shown in Figure 5 and Figure 6.

From Figure 5, it can be seen that the proposed ILA-OT algorithm outperforms LA in terms of the processing time. The percentage decrease in processing time by ILA-OT is 32.43%. In addition to that, the number of comparisons having equal edit distances between both algorithms is identical to the total number of the comparisons as shown in Figure 6. This means that the accuracy between both algorithms is 100%. This indicates that the proposed algorithm ILA-OT had a significant reduction in the processing time of LA without affecting its accuracy.

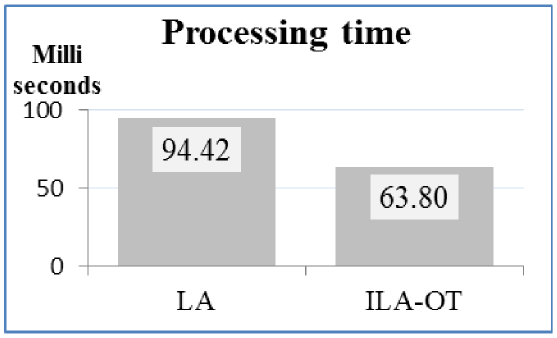


Figure 5: Processing Time of Both LA and ILA-OT

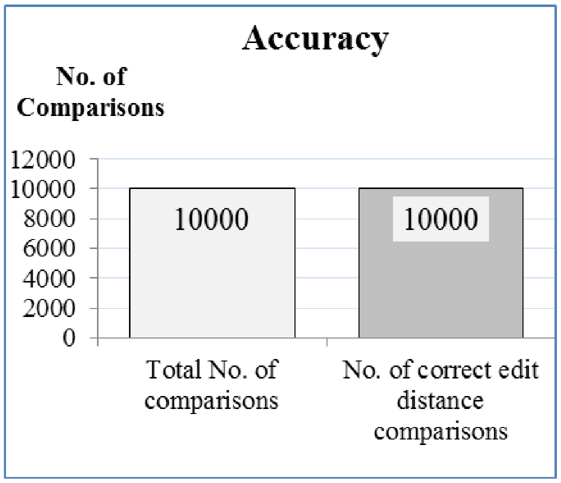


Figure 6: Accuracy between LA and ILA-OT

4. CONCLUSION AND FUTURE WORK

The goal of this study is to improve Levenshtein algorithm by reducing its processing time in generating candidates list for spelling correction. Therefore, this study has designed an ILA-OT algorithm, which is based on the concept of finding patterns to use in predicting the cells' values in LA array instead of measuring values of them by using the traditional way of LA. Experimental results proved that the patterns proposed by this study are able to decrease processing time of LA operations by 32.43% without affecting its accuracy. The proposed ILA-OT is hoped to contribute various applications that initially employed Levenshetin algorithm and this includes DNA searching, sequences' alignment, word-processing programs, speech processing systems, and optical character recognition systems. Future research of this study is to design a technique that can help ILA-OT in filtering lexicon words quickly.

REFERENCES:

[1] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, 2nd ed.: Pearson Education India, 2009.

[2] A. F. R. Al-Masoudi and H. S. R. Al-Obeidi, "Smoothing Techniques Evaluation of N-gram Language Model for Arabic OCR Post-processing," *Journal of Theoretical and Applied Information Technology,* vol. 82, pp. 432-439, 2015.

[3] Y. Bassil and M. Alwani, "Context-sensitive Spelling Correction Using Google Web 1T 5-Gram Information," *Computer and Information Science,* vol. 5, pp. 37-48, 2012.

[4] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, 1966, p. 707.

[5] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM),* vol. 21, pp. 168-173, 1974.

[6] W. Magdy and K. Darwish, "Effect of OCR error correction on Arabic retrieval," *Information Retrieval,* vol. 11, pp. 405-425, 2008.

[7] M. Al Azawi, "Statistical Language Modeling for Historical Documents using Weighted Finite-State Transducers and Long Short-Term Memory," PhD dissertation, Technical

University of Kaiserslautern, Kaiserslautern, Germany, 2015.

[8] T. Naseem, "A Hybrid Approach for Urdu Spell Checking," National University, 2004.

[9] J. F. Daðason, "Post-Correction of Icelandic OCR Text," (Master's thesis, University of Iceland, Reykjavik, Iceland), 2012.

[10] I. Q. Habeeb, S. A. Yusof, and F. B. Ahmad, "Two Bigrams Based Language Model for Auto Correction of Arabic OCR Errors," *International Journal of Digital Content Technology and its Applications,* vol. 8, pp. 72 - 80, February 28 2014.

[11] S. G. J. Vargas, "A Knowledge-Based information Extraction Prototype for Data-Rich Documents in the Information Technology Domain," National University, 2008.

[12] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys (CSUR),* vol. 33, pp. 31-88, 2001.

[13] I. Q. Habeeb and S. A. Yusof, "Design of Automatic Bilingual Lexicon for Arabic OCR Post-Processing Errors Correction," in *International Conference on Rural ICT Development*, Malacca, MALAYSIA, 2013.

[14] Y. Bassil and M. Alwani, "Ocr post-processing error correction algorithm using google online spelling suggestion," *Journal of Emerging Trends in Computing and Information Sciences,* vol. 3, pp. 90-99, 2012.

[15] S. Pal and S. Rajasekaran, "Improved algorithms for finding edit distance based motifs," in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, 2015, pp. 537-542.

[16] G. Navarro, S. Grabowski, V. Mäkinen, and S. Deorowicz, "Improved time and space complexities for transposition invariant string matching," in *Technical Report TR/DCC-2005-4, Department of Computer Science*, ed: University of Chile, 2005.

[17] A. Andoni and R. Krauthgamer, "The smoothed complexity of edit distance," *ACM Transactions on Algorithms (TALG),* vol. 8, p. 44, 2012.

[18] A. Andoni and K. Onak, "Approximating edit distance in near-linear time," *SIAM Journal on Computing,* vol. 41, pp. 1635-1648, 2012.

[19] S. Burkhardt and J. Kärkkäinen, "One-gapped q-gram filters for Levenshtein distance," in *Combinatorial pattern matching*, 2002, pp. 225-234.

[20] M. Huldén, "Fast approximate string matching with finite automata," *Procesamiento del lenguaje natural,* vol. 43, pp. 57-64, 2009.

[21] S. Mihov and K. U. Schulz, "Fast approximate search in large dictionaries," *Computational Linguistics,* vol. 30, pp. 451-477, 2004.

[22] Z. Q. Al-Zaydi and H. Salam, "Multiple Outputs Techniques Evaluation for Arabic Character Recognition," *International Journal of Computer Techniques (IJCT),* vol. 2, pp. 1-7, 2015.

[23] W. B. Lund, "Ensemble Methods for Historical Machine-Printed Document Recognition," PhD dissertation, Brigham Young University, Utah, United States, 2014.

[24] Y. Batawi and O. Abulnaja, "Accuracy Evaluation of Arabic Optical Character Recognition Voting Technique: Experimental Study," *IJECS: International Journal of Electrical & Computer Sciences,* vol. 12, pp. 29-33, 2012.