

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/339979921>

# Designing an Accurate and Efficient Algorithm for Matching Arabic Names

Conference Paper · December 2019

DOI: 10.1109/ICOICE48418.2019.9035184

CITATIONS

7

READS

364

4 authors:



**Salah Al-Hagree**

Sana'a University

24 PUBLICATIONS 112 CITATIONS

SEE PROFILE



**M. Sanabani**

Thamar university

22 PUBLICATIONS 115 CITATIONS

SEE PROFILE



**Doris Chen**

Stanford University

38 PUBLICATIONS 230 CITATIONS

SEE PROFILE



**Mohammed Hadwan**

Qassim University

63 PUBLICATIONS 774 CITATIONS

SEE PROFILE

# *Designing an Accurate and Efficient Algorithm for Matching Arabic Names*

**Salah AL-Hagree**  
Department of Computer Sciences  
&Information Technology  
Ibb University ,Yemen  
[s.alhagree@gmail.com](mailto:s.alhagree@gmail.com)

**Maher Al-Sanabani**  
Faculty of Computer Science and  
Information Systems  
Thamar University, Yemen  
[M.sanabani@gmail.com](mailto:M.sanabani@gmail.com)

**Khaled M.A.Alalayah**  
Computer Science, IBB  
University, Yemen , Computer  
Science, Najran, KSA,  
[Kmalalayah@nu.edu.sa](mailto:Kmalalayah@nu.edu.sa)

**Mohammed Hadwan**  
Information Technology  
department, college of Computer,  
Qassim University.  
Computer science Department  
Taiz University  
[M.hadwan@qu.edu.sa](mailto:M.hadwan@qu.edu.sa)

**Abstract**—A great deal of research has been done to find out an accurate algorithm for name matching that would play a major role in the application process. Researchers have developed several algorithms to measure the similarity of string, but most of them are designed mainly to deal with Latin-based languages. However, dealing with the Arabic context is a challenging task, owing to the nature and unique features of the Arabic language. This can explain why the name matching algorithms in the Arabic context are rare. Therefore, this paper aims at designing an accurate and efficient algorithm for matching Arabic names. In this paper, a framework for matching Arabic names has been designed to provide a platform for the current and future investigations, involving matching Arabic names. This framework deals with specific characteristics of Arabic language and the various levels of similarities for Arabic letters, mainly keyboard similarities, letter forms and phonetic similarities. Moreover, the proposed algorithm accounts for the operation of transposition and the enhanced states of substitution, deletion and insertion operations. Therefore, the proposed algorithm reduces the storage space of the process, saves the time of processing time and reduces the time complexity from  $O(N^3)$  to  $O(N^2)$ . Besides, the experiments show that the proposed algorithm is more efficient and more accurate than the other algorithms.

**Keywords:** Matching Arabic names, String matching, Character N-gram, Levenshtein distance.

## I. INTRODUCTION

String matching is a universal technique to solve problems in different areas such as text mining, natural language processing (NLP), computer vision, image processing, and pattern recognition. It has been a challenge for all communities of scientists to devise a more accurate algorithm to solve such problems [1,2,3,4]. Since this topic is of great significance, experts in this field of study have developed and used many string-matching algorithms. The exact string-matching algorithms [5,6,41] and approximate string matching (inexact) algorithms [1,2,7] are classified as two kinds of string-matching algorithms. Identity matching systems always employ name search algorithms to specify related information about a given person effectively. These systems are successfully used for applications such as Criminal Investigation (CI), Customer Data Integration (CDI), Customer Relation Management (CRM), HealthCare (HC), Anti-Money Laundering (AML), Genealogy Services (GS), Tax Fraud Detection (TFD) and Immigration Control (IC). Using names to retrieve information makes such systems vulnerable to problems arising from the typographical errors. That is, accurate match search methods will not find instances of misspelled names or the names which have more than one accepted spelling [4]. So, an approximate name matching should be applied instead of exact matching to get more accurate results. Thus, the importance of such name-based matching applications has resulted in the improvement of name matching algorithms for English which utilizes string information, but these language-dependent techniques have not

been applied to Arabic. Therefore, the driving motive behind this paper is to provide a matching algorithm for names, which is an approximate string-matching algorithm that can deal with the technician errors that are permitted in the field of computer science. Such algorithms are utilized in many applications, including Computational biology “DNA” [8,9], Spelling correction [10,11,12,13], Text retrieval [14,15] Handwriting recognition [16] and Linking database [17,18] and Name Recognition [4] and etc.

The organization of this paper is as follows: section II illustrates several challenges of Arabic Language, section III describes the related literature, section IV describes in details the proposed framework for matching Arabic names, section V presents the proposed HL-Big-A algorithm for matching Arabic names, and section VI. presents the experiments, results and discussion of the performance analysis.

## II. CHALLENGES OF ARABIC LANGUAGE

As one of the world's major languages and used by nearly 2 billion people, Arabic language proves to be of vital importance in today's technology. Therefore, many nations, Arabic or Islamic, speak Arabic language which has several characteristics and features. Arabic language can be categorized into three main types: Colloquial/Dialects Arabic (C/DA), Modern Standard Arabic (MSA) and Classical Arabic (CA) [19]. In this paper, we deal with MSA. Moreover, Arabic-like scripting is used in several languages, such as Pashto (the language spoken in Afghanistan), Persian (the language spoken in Iran) and Urdu (the language spoken in Pakistan). In general, the Arabic alphabet system consists of 28 basic letters. In some cases, researchers add the 4 Lam-Alif “لا” ligatures to form 36 Arabic letters (viz., the basic 28, the Hamza “ء” letters, Ta-Marbuta “ة” and Alif-Maqsurah “ى”), which results in a total of 40 letters [20,21,22]. Generally speaking, the word feature in Arabic is shorter in terms of length than the word in Latin. Many words in Arabic can be driven from short roots. What confuses most Arab writers is locating hamza “ء” /ʔ/ (glottal stop) which has 8 different forms (أ - إ - ؤ - ئ - ؤ - آ - ء - ـأ). Alef Maksoorah and Yaa (ى - ي) are sometimes perplexing at the end of words, as one of them has 2 dots under it, and the other one does not have. The same happens with kHaa and Tdaa Marboota (ة - ه), as one of them has 2 dots on it, and the other does not have. There are other letters with such similar form like , {ح - ص}, {غ - ف}, {أ , إ , ؤ}, , {ع - م}, {ت - ن - ي - ب}, {ر , ز} and {ذ , د}.

The name matching poses several issues for Arabic and English languages in the following aspect:

- First, the reasons for appearing different include typing and OCR errors such as “David Smith” is

misspelled as “Davod Smith” and “بصبة” is misspelled as “بصببة”.

- Second, the cause of these errors comes as result of the keyboard adjacencies such as “David” is misspelled as “Davjd” and “صباح” is misspelled as “صلاح” and “Osama” is misspelled as “Usama”.
- Third, the pronunciation of some words might lead to the duplication of some letters such as “almost” is misspelled as “allmost” and “عمار” is misspelled as “عمار”.
- Fourth, the pronunciation of some words might lead to the deletion of some letters such as “really” is misspelled as “realy” and “embarrass” is misspelled as “embarass”.
- Fifth, the deletion or insertion cost of a blank is defined to be 0 beneficial to segmentations that might occur in names. Some examples are “the letter” is misspelled as “the letter”, “عبد الله” is misspelled as “عبدالله”, “ابو بكر” is misspelled as “ابوبكر” and “رئيس الجمهورية” is misspelled as “رئيسالجمهورية”.

Besides, there are additional five issues for Arabic language in the following aspects:

- The first one deals with the fact in the case the Hamza letter /ʔ/ (glottal stop) is placed after long vowels (Alif, Waw and Yaa) in a name, the Hamza can be taken away without any negative effect. Both names, with and without Hamza are correct. For instance, when we spell the word “اسماء” in the place of “اسماء” and “انباء” in tge place of “انباء”.
- The second one is concerned with the fact that if duplicate or identical letters, a diacritic called “Tashdid” is used above the letter which is pronounced twice. Generally, particularly in names, “Tashdid” is also taken out from the name such as “محمد” that should be written as “محمّد”, but it is put in the form of “محمد” in nominal names and the letter of “م” appears in pronunciation.
- The third issue is that Arabic writers tend to commit spelling mistakes or typographical errors frequently according to certain characters. For example, “الإسلامي” / The Islamic with “إ” can be written “الاسلامي” with “ا” and “أساسي” / The basic with “أ” can be written “اساسي” with “ا”.
- The fourth issue is Editing Errors. That is, learners tend to misspell a word by adding, transposing, omitting, substituting or duplicating a letter within a given word, leading to spelling errors. For example, spelling the word “مكتبة” instead of the “مكتبة”, “محمد” instead of “محمد”, “قائمة” instead of “قائمة”, “كتاب” instead of “كتاب”, “عاصم” instead of “عاصم”.
- The fifth one is that the analysis of the characters, that leads to substitution and transposition errors, has enlightened us to conclude that substitution and transposition can happen due to three level of similarity as shown in the following and in table1:
  - The first level is the keyboard similarity due to the proximity between the English and Arabic keys, so the

operator clicks directly on the key adjacent to the correct one, such as “operator” is misspelled as “opreator” and “صلاح” is misspelled as “صباح”.

- The second level is the similarity of the letter form connected with the similarity between Arabic characters, as the similitude among specific Arabic characters is clear evident. It is supposed that one can safely illustrate the similarity in calligraphy between: (ع;غ) and (ط;ظ), (ف;ق), (ر;ز), (د;ذ), (ص;ض), (ح;ج;خ): such as “خروج” is misspelled as “حروج” and “مجهود” is misspelled as “مجهوذ”, respectively.
- The third level is the similarity of phonetic (ص,س) and (ض,ذ), such as “عصفور” is misspelled as “عسفور” and “ضجاجة” is misspelled as “ضجاجة”, respectively.

TABLE 1. TYPES OF ERROR FOR TYPOGRAPHIC

Type error	Misspelling	Correct spelling
Insertion	- cress - عمار - العلوم	- acress - عمار - العلوم
Deletion	- مكتبة - Acress	- مكتبة - actress
Transposition	- teh - Sydeny - اجتماع - Father - Acress	- the - Sydney - اجتماع - Father - Caress
Space Deletion	- رئيسالجمهورية - Theletter	- رئيس الجمهورية - the letter
Substitution	- Volkswagon - خرير - acress - cotd - صلاح	- Volkswagen - خرير - across - cord. - صباح
Substitution and Deletion or Insertion	- Vaccume - unconscience - المر	- Vacuum - Unconscious - المرء
Transposition and Deletion or Insertion Space	- مجيد	- مجدي

### III. RELATED WORK

In this section, a brief review of the related literature is presented. In [23], it is found that Levenshtein distance (LD) editing algorithm is based on binary symbols (0,1). The distance factor is used to compute the similarity between two strings. The distance is computed by computing the minimum number of operations we need to convert a string to another series. These processes are addition, substitution, and deletion. However, this algorithm did not give accurate results because it did not take into account similarity levels but gave value 1 even if the two words were similar. The algorithm was not suitable for Arabic and the transposition operation of 2 adjacent characters is not considered by this work-study. Damerau [24]

improves Levenshtein Distance algorithm (LD) by the additional operation to examine the distance between strings, that's, a transposition of 2 adjacent characters. The averter said 'that DLD algorithm has matched more than 80% of human misspellings' by taking operations for the 4 string (Substitution, Transposition, Insertion and Deletion). This algorithm is also used in biology for measuring the variation between DNAs. The multi states of the operation of transposition are not considered by this work. For automatic correction of errors in spelling, [24] identified four main types. As mentioned above, in table 1, with examples in Arabic and English. Despite the fact that Damerau's research main limitation lies in scope which is English, and the identified categories are similar in other languages, Arabic of course is included [25]. [26] Presented a new approach to match the Arabic name and personal name specifically. The matching algorithms have been classified into three algorithms based on distance, character-based, and audio algorithms. The AEDA algorithm, which is classified within distance-based algorithms to measure the similarity between two Arabic series, is based on the LD that is based on dynamic programming, and takes into account the unique features of the Arabic language similarity of voice and similarity of the keyboard. This algorithm is compared to the levenshtien algorithm and found to produce more accurate results from levenshtien as it gives different weights to the editing processes and takes into account different levels of similarity of Arabic characters. Levenshtein gives the same weight to all edits and does not give any attention to similarity.

Moreover, a number of applications or papers that used LD algorithm for different applications like checking the spelling of Arabic texts [10,11,12,13,27]. In [28] uses Modified Damerau-Levenshtein Distance (MDLD) which is a support block (word) transpositions of multiple characters that have been applied on oracle database. This algorithm increases the complexity of time to  $O(N^3)$ . Moreover, this study cannot deal with possible spelling variations and errors. The block-based measures compare text strings as sequences of the block instead of sequences of characters. This approach can be successful when it is applied to the comparison of text strings with many blocks and with a different order of the block or missing blocks(words). In [1,2,29], Q-grams, which are sub-strings of length q of a longer string, is used. Depending on q, q-grams are called unigrams, bi-grams, tri-grams, and so on. For example, all bi-grams of the string "hadramout" are "ha", "ad", "dr", "ra", "am", "mo", "ou", "ut". The similarity between strings can be computed using the Dice coefficient as in Equation (1):

$$DICE(a,b) = 2 \cdot \frac{A(a) \cap A(b)}{A(a) \cup A(b)} \quad (1)$$

where  $A(x)$  is the bigrams set of a given string x, i.e., the measure is acquired by sub-dividing the number of bi-grams relevant to the 2 strings by the average number of bi-grams in every string. It is difficult for this algorithm to handle such spelling variations and errors because this algorithm deals with words but not characters. Moreover, the stemming process is required to obtain a big similarity between two words. In [14,15], the n-grams algorithm is used in Arabic text retrieval. However, these algorithms show a low similarity between

words when using pure n-grams. For example, the similarity between "استفسر" and "الاستفسارات" is 0.533. So, they improved the results by applying a two-step approach, in which the N-gram approach is combined with a stemming technique. In [30], Kondrak proposed a luminal method (N-DIST) that is fusing the components of LD and N-gram algorithms and suggested a similarity measurement. It is evaluated based on Genetic cognate words of the same origin that belong to different languages. For instance, English "father", Norwegian "far", and German "vater" constitute a set of cognates, confusing drug names, and translational cognates. The algorithm has the advantage of levenshtein and n-grams algorithms. However, this algorithm increased the complexity of Arabic to  $O(N^3)$  and the transposition operation of adjacent characters is disregarded. In [31,32,33], N-DIST algorithm is used in different applications based on English language, while [21] designed an algorithm to match the Arabic names. This algorithm has two steps. The computing of similarity is the first step in which it computes the cost and output. The second step is to compute the similarity ratio. The cost is computed using the n-gram method in computing the similarity ratio taking into consideration the characteristics which are characterized by the Arabic language as well as the similarity of voice and formality and the similarity of the keyboard. The algorithm suggested in this study gave more accurate results to compare the Arabic names and better results in the retrieval of information from large databases and other processes within the corresponding period so as to take into account the features of Arabic.

In [42] studied and discussed the algorithms of measuring string similarities for the Arabic language. The proposed framework in the study took advantage of unique features of the Arabic language and the different levels of similarity of the Arabic characters, the similarity of sound, the similarity of the form and the similarity of the keyboard distance. The proposed framework was also considered a transfer and enhanced case for insertions and deletions. Experiments in the study showed that the proposed framework gives more accurate results. However, this algorithm increased the storage space of the process and does not give any attention to saving the time of processing time. Another study found that the A-N-DIST [34], which combines N-DIST & LD matching techniques to produce much better and more accurate results. The technique proposed by the study as well as the editing process is a new process of exchanging vowels (i, u, y, e, a, o) to compute the most common misspellings in vowels due to the conversion of names into another language.

Multiple states of transposition to correct errors as mentioned in N-DIST algorithm is not taken into account by this N-DIST-A and A-N-DIST algorithms [21,30,34].

The N-DIST and N-DIST-A and A-N-DIST algorithms have been discussed above. These algorithms have two problems:

- These algorithms are dependent on the value of N for N-grams where  $N=2,3,4$ . etc. Therefore, increasing the value of N means high storage space for processing and more time processing. Furthermore, these algorithms have been tested when  $N=2$ , Bigram "BI" and  $N=3$  "TRI", Trigram in [21].

- These algorithms have computed the weights for estimate cost for each case as in the following steps:
  - The first step is by giving initialized value for the weight, for example,  $w_1=N$  in case 1 when  $N=2$ .
  - The second step is each comparison between the letter in Case 1 does (If there is matching between the letters, the cost value will be decreased  $w_1$  by one else do nothing for value  $w_1$ ).
  - In the final step is that the estimated cost is the value  $w_1$  divided by  $N$ .

As mentioned above, these steps of the computed weights of estimate cost have increased the time complexity of these algorithms by  $N$ . Therefore, these algorithms have  $O(n^3)$  complexity in which LD and DLD algorithms have  $O(n^2)$  complexity.

#### IV. THE PROPOSED FRAMEWORK

A proposed framework that has been designed to provide a platform for current and future investigation involving Arabic name matching is introduced in this section. The framework has been specially designed to deal with issues of matching Arabic name algorithms, unique features, and characteristics of the Arabic language. Therefore, this framework can be used for studying current and proposed algorithms. The purpose of this paper is adjusted to get a more accurate and efficient algorithm for matching Arabic names.

The main contribution intended in this work is maximizing the efficiency of the Arabic name matching algorithm through the following:

- Designing a framework to be a platform for the performance evaluation of this paper.
- Reducing the storage space of the process.
- Saving the time of processing time.
- Reducing the time complexity from  $O(N^3)$  to  $O(N^2)$ .

Hence, designing a matching Arabic name algorithm that can achieve both space and time efficiency poses a major challenge. The proposed framework is used for the performance evaluation of this paper. From an empirical point of view, Figure 1. presents the framework architecture which is designed to reflect the different procedures gathered with the interacts. Figure 1. exhibits the component of the proposed framework that consists of four parts. The first part contains the affixing procedure. The second part contains a joint procedure and splits procedure. The third part is the main core of the framework that contains the proposed algorithms for Arabic name matching to be evaluated. The final part contains the efficiency of the implemented algorithms. The details of the process in the proposed framework parts are to be elaborated are the following sub-sections.

##### A. Affixing Part

The major aim behind the affixation way is to stress the first segments, which seems to be much more significant than the final segments in specifying the similarity of names. A special sign is measured for each letter of the original alphabet.

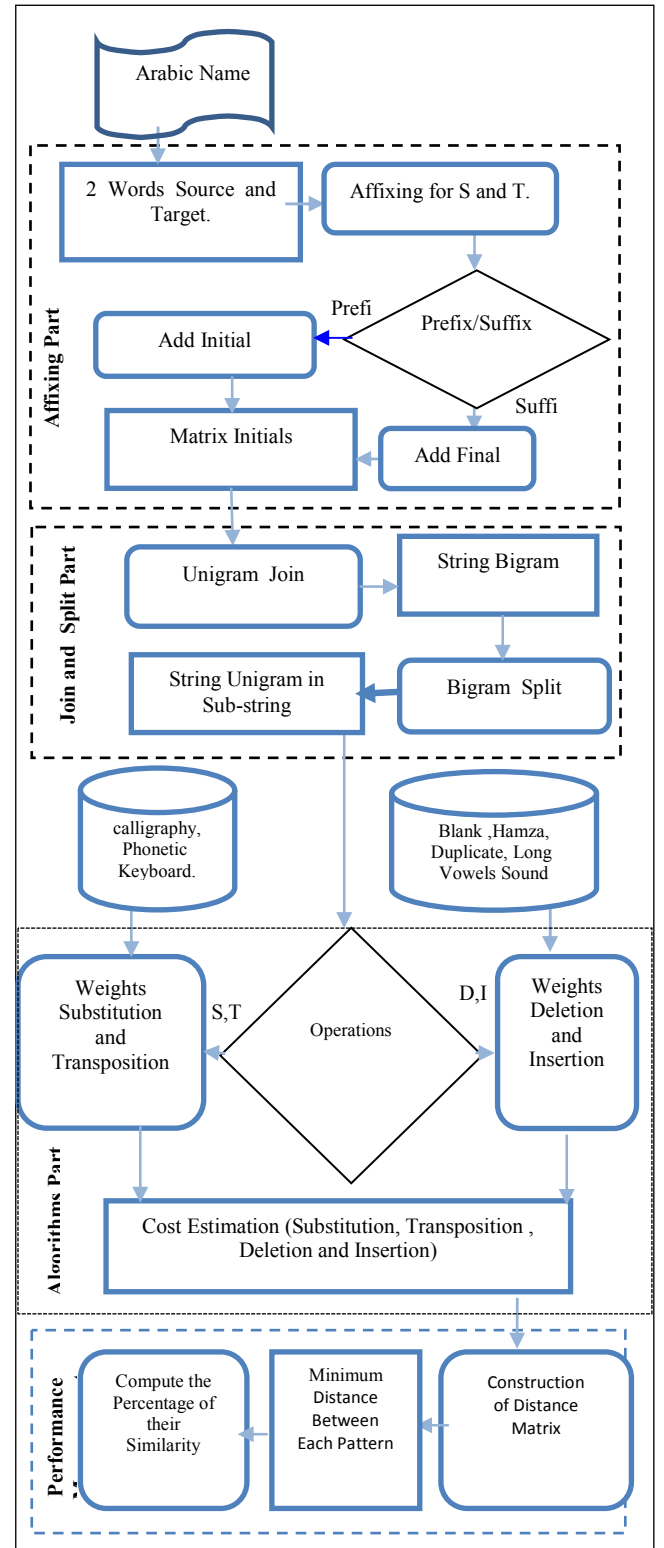


Fig 1. The proposed framework for matching Arabic name.

. Each name is extended with a prefix composed of  $n-1$  copies of the special symbol that resembles the initial letter of the name. For example, 'حضر موت' is changed into 'حضر موت -'. That begins by forming an  $(n+1) \times (m+1)$  matrix MD consisting of all 0s. So, the  $MD(i,j) = 0$ , for  $i = 0$  to  $n$  and  $j = 0$  to  $m$ . The



affixation process with initials of MD matrix of source and target names is reflected in figure 2.

---

**Function.** The Affixing and Matrix Initially  
**Input:** The symbol Letters (symbol Letter)  
**Output:** String symbol (Letters)  
**String** Affixing and Matrix Initially Op (S,T)  
**Read**(symbol)  
**If** (symbol = prefix) **then**  
    S= - & source   // add Initial for source and target .  
    T= - & target  
**Else If** (symbol = suffix) **then**  
    S= source & -   // add final for source and target.  
    T= target & -  
**End if**  
**MD** [0,0] = 0   // Matrix initially  
**Return** S,T

---

Fig 2. The function of affixing

### B. The Joint and Split Part

The main idea is that to assign values MD(i,0)=i where i=0 to n and x[i]= S.Sub-string(i, 2), also MD(0,j) = j where j = 0 to m and y[j]= T.Sub-string(j, 2). These transactions are shown in figure 3. The joint function has been shown in figure 4. This function converts the Unigrams to Bigram s for each source and target names.

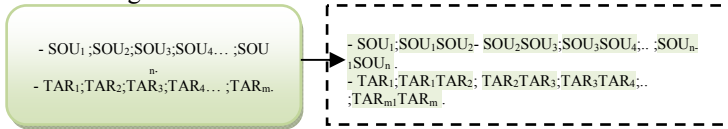


Fig 3. Join Unigram to Bigram

---

**Function.** The Convert From Unigram to Bigram  
**Input:** Unigram Letters (S,T)  
**Output:** Bigram Letters (x,y)  
**String** Join Unigram to Bigram Op (x, y)  
    **for** i from 0 to n  
        **MD** [i, 0] = i ;   // Matrix initially  
        x[i]= S.Sub-string(i, 2) ; // Join Unigram to Bigram for source  
    **for** j from 0 to m  
        **MD** [0, j] = j ;   // Matrix initially  
        y[j]= T.Sub-string(j, 2) ; // Join Unigram to Bigram for target  
**Return** x, y

---

Fig 4. Join Function

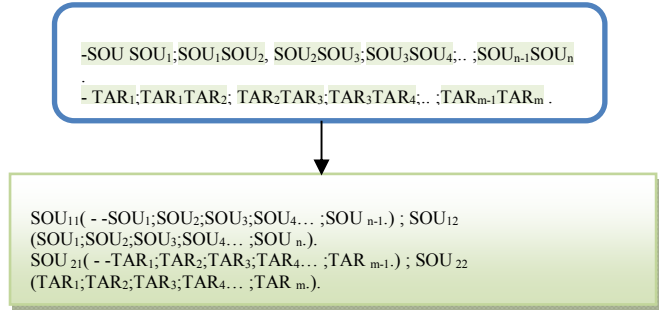
Figure 5. shows the process of splitting Bigram to Unigram. It starts by filling the values of MD (i,j) from the second top row and going from left to right according to the following steps:

- The first step is giving  $S_1=x[i-1]$  and  $S_2=y[j-1]$ .
- The second step is  $S_{11}=S_1$ .Sub-string (0,1);
- $S_{12}=S_1$ .Sub-string(1,1);  $S_{21}=S_2$ .Sub-string(0,1);  
 $S_{22}=S_2$ .Sub-string(1, 1).

The splitting function is shown in figure 6. This function splits the Bigrams to Unigrams for each source and target names

### C. Proposed Algorithms Part

The proposed algorithm is described in details in section V because these algorithms are the main work in this paper.



- The HL-Big-A algorithm computes the estimated cost as operations of deletion and insertion depend on the number of one procedure, while the original N-DIST-A algorithm depends on the number of two procedures to compute the estimated cost as of operations deletion and insertion.

- In the following sub-section, A. introduces a three-level of similarities for the Arabic language while the sub-sections B. and C. are to solve the problems of storage space, long matching time and time complexity during the name matching process of N-DIST-A by proposing a new matching Arabic name algorithm based on high-efficiency.

### A. The Levels of similarity in Arabic

- 1) Letter's form similarity

1. A letter standing alone.
2. like the initial letter in a word.
3. Infix, between two other letters.
4. like the final letter in a word.

1. like the initial letter in a word.
2. like the final letter in a word.

The Arabic letters can be shown in 108 forms. These forms can be classified into 6 categories depending on their form similarity. Table 3. presents the groups of Arabic letters for

TABLE 2. TWENTY TWO OF ARABIC LANGUAGE ALPHABETS HAVE 4 FORMS

Isolated	Character	Initial	Middle	Final
ث	Tha	تـ	ـتـ	ثـ
ج	Jeem	جـ	ـجـ	جـ
ر	Ra			رـ
ز	Za			زـ
س	Seen	سـ	ـسـ	سـ
ع	Ayn	عـ	ـعـ	عـ
غ	Ghain	غـ	ـغـ	غـ
ف	Fa	فـ	ـفـ	فـ
ه	He	هـ	ـهـ	هـ
و	Waw			وـ
ي	Ya	يـ	ـيـ	يـ

TABLE 3. SUCH CATEGORIES OF ARABIC ACCORDING TO FORM SIMILARITY.

No.	Groups of Similar	Index of similarity
1	{أ - آ - إ}, final form of {ة - ه} and {ي - ي}	1
2	{ى - ئ}, {ؤ - و}, {ء - أ}, {ء - إ}, {ء - أُ}, {ء - ا}, {ء - ء}	0.8
3	{ن - ب}, {ث - ت}, {ح - ج - خ}, {ذ - ز}, {ر - د}, {غ - ع}, {ق - ف}, {ظ - ط}, {ب - ي}, initial and medial form of {ب - ي}	0.6
4	{س - ش}, {ب - ث}, initial and medial form of {ت - ي}, {ث - ي}	0.4
5	medial form of {م - ه}, final form of {ل - ك}	0.2
6	Any other combination of Arabic letters	0

## 2) *Phonetic Similarity*

The Arabic language is very rich in its lexicon because of its phonetic capacity. The Arabic letters are pronounced from nearly seventeen variant points. Hence, any pronunciation deviation may change meaning. The pronunciation relies on Arabic language types, i.e. modern standard, classical or colloquial. The Arabic letters can be classified into 6 categories based on their phonetic similarity. Table 4. presents the groups of Arabic letters for each category based on their similarity index starting from 100% and ending with 0% similarity.

TABLE 4. THE CATEGORIES OF ARABIC LETTERS ACCORDING TO PHONETIC SIMILARITY.

No.	Groups of similar	Index of similarity
1	{ء-أ-إ-ي}, {ي-ى}	1
2	{ط-ظ-ض}, {ز-ظ-ذ}, {س-ص}, {ث-س}, {ت-ب}	0.8
3	{ن-م}, {ق-ك}	0.6
4	{ج-د}, {ق-ي}, {ح-ج}	0.4
5	{ع-ي}, {ر-ي}	0.2
6	Any other combination of Arabic letters	0

### 3) The Similarity of keyboard

The operator will be more likely to make a mistake on the keyboard for the proximity letters. That is, the similarity

between 2 keys increases if they have closer locations on the keyboard. Thus, the keyboard similarity is related to the distance. Therefore, the keyboard similarity has the following equation (4).

$$\text{sim}_{\text{keyboard distance}}(a, b) = 1 - \frac{\sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2}}{\varphi} \quad (4)$$

Where  $X_a$  &  $X_b$  are the positions of any 2 keys (a and b) on the X axis,  $Y_a$  &  $Y_b$  are the positions of the 2 keys (a and b) on the Y axis. The highest possible distance on the Arabic keyboard is approximately 12 like the distance between “د” and “ذ”. The keyboard distance similarity between “ص” and “ض” is 0.9 unit, which is, consequently, the similarity between them while the keyboard cost component will be 0.1. The Keyboard distance similarity, which is computed by equation (4), for some Arabic letters is shown in table 5.

#### B. The Storage of Space and Time Processes

For reducing the storage space of the process and saving the time of processing time takes three steps.

- The first step is to fix  $N=2$  "BI".
- The second step is that the affixing method followed in section A. helped to a great extent to fix initial or final segments in the source (S) and target(T) as a unique symbol for each letter of the original alphabet. For example, 'محمد' is transformed into 'محمد'. In table 6. (a) and (b) show the storage space of the process.

TABLE 5. KEYBOARD DISTANCE SIMILARITY

ط	ظ	ض	ص	ب	أ	ا	
0.6	0.7	0.6	0.7	0.8	1.0	1.0	ا
0.6	0.7	0.6	0.7	0.8	1.0		أ
0.4	0.5	0.7	0.8	1.0			ب
0.2	0.3	0.9	1.0				ص
0.2	0.2	1.0					ض
0.9	1.0						ظ
1.0							ط

As in table 6. (a) The storage space of process for which can lower memory access, save storage space and reduce matching time when  $N=2$ , Bigram. The main reason for this is that the affixing method when  $N=2$ , Bigram uses only one initial or final segment. Hence, affixing segments can reduce a great amount of memory access. Therefore, it is noticed that the storage space of the process increases with the increase in the value (N) as shown table 6.(b) and to reduce these processes as shown in table 6.(a) that the state of Bigram when N equal 2 is the best.

- The final step is to join and split the method in section I.V to help fix N equal two as shown in table 7. saving the time of the processing time.





TABLE 6.(A) THE STORAGE SPACE OF PROCESS

Affixing part	BI is N=2	TRI is N=3	N=4
Source	- S <sub>1</sub> .	-- S <sub>1</sub> ., - S <sub>1</sub> S <sub>2</sub> .	--- S <sub>1</sub> ., --- S <sub>1</sub> S <sub>2</sub> ., - S <sub>1</sub> S <sub>2</sub> S <sub>3</sub> .
Target	- T <sub>1</sub> .	-- T <sub>1</sub> ., - T <sub>1</sub> T <sub>2</sub> .	--- T <sub>1</sub> ., --- T <sub>1</sub> T <sub>2</sub> ., - T <sub>1</sub> T <sub>2</sub> T <sub>3</sub> .

TABLE 6.(B) THE STORAGE SPACE OF PROCESS FOR N.

Affixing part	N=n
Source	--- ... - n S <sub>1</sub> ., --- ... - n-1 S <sub>1</sub> S <sub>2</sub> ., --- ... - n-2 S <sub>1</sub> S <sub>2</sub> S <sub>3</sub> ., ..., - S <sub>1</sub> S <sub>2</sub> S <sub>3</sub> .. S <sub>n</sub>
Target	--- ... - n T <sub>1</sub> ., --- ... - n-1 T <sub>1</sub> T <sub>2</sub> ., --- ... - n-2 T <sub>1</sub> T <sub>2</sub> T <sub>3</sub> ., ..., - T <sub>1</sub> T <sub>2</sub> T <sub>3</sub> .. T <sub>n</sub>

TABLE 7. SAVING THE TIME OF PROCESSING TIME.

Part	Join and Split	BI is N=2	TRI is N=3	N=4	N=n
Case 1					

As in shown table 7. the time of the processing time for which can reduce matching time when  $N=2$ , Bigram. The main reason is that join and split method when  $N=2$ , Bigram uses only seven states. Hence, join and split can reduce a reduce matching time. Therefore, it is noticed that the time of the processing time increases with the increase in the value (N) as shown in table 7. when  $N= n$ . From in [21], have been observed that N-DIST and N-DIST-A algorithms with N equal to Bigram =2 have given more accurate results than the with N equal to Trigram  $N=3$  for all datasets and to saving the time of processing time where the Bigram when N equals 2 is the best state.

#### C. The Time Complexity

For reducing the time complexity from  $O(N^3)$  to  $O(N^2)$  after taking the following two steps: the first step is to compute the estimated costs of substitution, transposition and multi-transposition operations depending on giving fixed states which is equal to 7 as shown in figure 7.

Case one. Case two; Case three; Case four; Case five; Case six; Case seven ;



Cost= $w_1$ ; Cost= $w_2$ ; Cost= $w_3$ ; Cost= $w_4$ ; Cost= $w_5$ ; Cost= $w_6$ ; Cost= $w_7$

Fig 7. Different weighted cases of operations.

As shown in figure 8. the function of substitution, transposition and multi-transposition operations compute the weight of the different cost components. The symbols  $w_1$ ,  $w_2$ ,  $w_3$ ,  $w_4$ ,  $w_5$ ,  $w_6$  and  $w_7$  are substitution, transposition and multi-transposition operations cost components. The different weights  $w_1$ ,  $w_2$ ,  $w_3$ ,  $w_4$ ,  $w_5$ ,  $w_6$  and  $w_7$  are used to reflect the effect of different states. Case 1: the first case proves the exact matching of source sub-string ( $S_{11}$ ,  $S_{12}$ ) with the target sub-string ( $S_{21}$ ,  $S_{22}$ ). The estimated cost is given the value  $w_1$ . Case



2: the second case proves the not matching of source sub-string ( $S_{11}$ ,  $S_{12}$ ) with the target sub-string ( $S_{21}$ ,  $S_{22}$ ). V estimated cost is given the value  $w_2$ . Case 3: the third case proves the matching of source sub-string ( $S_{11}, S_{12}$ ) with the target sub-string ( $S_{22}, S_{21}$ ). The estimated cost is given the value  $w_3$ . Case 4: the fourth case proves the not matching of source sub-string ( $S_{11}$ ) with the target sub-string ( $S_{21}$ ) and matching of source sub-string ( $S_{12}$ ) with the target sub-string ( $S_{22}$ ). The estimated cost is given the value  $w_4$ . Case 5: the fifth case proves the matching of source sub-string ( $S_{11}$ ) with the target sub-string ( $S_{21}$ ) and not matching of source sub-string ( $S_{12}$ ) with the target sub-string ( $S_{22}$ ). The estimated cost is given the value  $w_5$ . Case 6: the sixth case proves the matching of source sub-string ( $S_{11}$ ) with the target sub-string ( $S_{22}$ ) and not matching of source sub-string ( $S_{12}$ ) with the target sub-string ( $S_{21}$ ). The estimated cost is given the value  $w_6$ . Case 7: the seventh case proves the not matching of source sub-string ( $S_{11}$ ) with the target sub-string ( $S_{21}$ ) and matching of source sub-string ( $S_{12}$ ) with the target sub-string ( $S_{22}$ ). The estimated cost is given the value  $w_7$ . These weights by handling a different type of spelling errors for Latin based languages, especially English language such as substitution, transposition and multi-transposition operations, and these weights handle the problems which are mentioned in section V.I. Therefore, the above cases have reduced the time complexity for algorithm N-DIS-A from  $O(N^3)$  to  $O(N^2)$ . The second step is to compute the estimated costs of insertion and deletion operations dependent on the modified Case 6 and Case 7 as shown in figure 9.

**Function.** To Compute the Cost of Substitution and Transposition of four Letters in Arabic.

**Input:** four Letters (Letters1, Letters2, Letters3 and Letters4)  
**Output:** Cost Substitution and Transposition Bigram Distance (cost)  
**Decimal** Substitution and Transposition Op ( $S_{11}$ ,  $S_{12}$ ,  $S_{21}$ ,  $S_{22}$ )  
*// Figure 6. in [21]*  
 $W_1 = \text{call Substitution Op1}(S_{11}, S_{21})$   
 $W_2 = \text{call Substitution Op1}(S_{11}, S_{22})$   
 $W_3 = \text{call Substitution Op1}(S_{12}, S_{22})$   
 $W_4 = \text{call Substitution Op1}(S_{12}, S_{21})$   
**if** ( $S_{11} = S_{21}$ ) **and** ( $S_{12} = S_{22}$ ) **then**  
    Cost=0 *//  $W_1 + W_3$  is 0*  
**Else if** ( $(S_{11} \neq S_{21})$  **and** ( $S_{12} \neq S_{22}$ )) **and** ( $(S_{11} = S_{22})$  **and** ( $S_{12} \neq S_{21}$ )) **then**  
    Cost = ( $(W_1 + W_3) / 2$ )  
**Else if** ( $S_{11} = S_{22}$ ) **and** ( $S_{12} = S_{21}$ ) **then**  
    *// Transposition  $W_2 + W_4$  is 0*  
    Cost = 0  
**Else if** ( $S_{11} \neq S_{21}$ ) **and** ( $S_{12} = S_{22}$ ) **then**  
    Cost =  $W_1$   
**Else if** ( $S_{11} = S_{21}$ ) **and** ( $S_{12} \neq S_{22}$ ) **then**  
    Cost =  $W_3$   
**Else if** ( $S_{11} = S_{22}$ ) **and** ( $S_{12} \neq S_{21}$ ) **then**  
    Cost =  $W_4$   
**Else if** ( $S_{11} \neq S_{22}$ ) **and** ( $S_{12} = S_{21}$ ) **then**  
    Cost =  $W_2$   
**Else** *//  $W_5 + W_6$  is 0*  
    Cost = 1  
**End if**  
**Return** Cost

Fig 8.. The function of substitution and transposition operations .

Furthermore, the Case 8 and Case 9 is for computing the estimated costs of insertion and deletion operations.

As shown in figure 10, the function of insertion and deletion operations computes the weights of the different cost components. The symbols  $w_8$  and  $w_9$  are insertion and deletion cost components.

**Function.** To Compute the Cost of Deleting and Inserting in Arabic.  
**Input:** Bigram Letters (Letter)  
**Output:** Cost Deleting and Inserting Distance (cost)  
**Decimal** Deleting and Inserting Op ( $S_{11}, S_{12}, S_{21}, S_{22}$ )  
**If** ( $S_{11} = S_{22}$ ) **and** ( $(S_{12} = "\text{ء"})$  or ( $S_{12} = \text{Blank}$ )) **then**  
    CostID  $\leftarrow w_8$  *//  $w_8 = 0$*   
**Else If** ( $S_{12} = S_{21}$ ) **and** ( $S_{22} = "\text{ء"}$ ) or ( $S_{11} = \text{Blank}$ ) **then**  
    CostID  $\leftarrow w_9$  *//  $w_9 = 0$*   
**Else if** ( $S_{11} = S_{22}$ ) **then**  
    CostID  $\leftarrow w_8$  *//  $w_8 = 0.25$*   
**Else if** ( $S_{12} = S_{21}$ ) **then**  
    CostID  $\leftarrow w_9$  *//  $w_9 = 0.25$*   
**Else if** ( $S_{11} \neq S_{22}$ ,  $S_{12} \in \epsilon$ ) **then**  
    CostID  $\leftarrow w_8$  *//  $w_8 = 0.50$*   
**Else if** ( $S_{12} \neq S_{21}$ ,  $S_{21} \in \epsilon$ ) **then**  
    CostID  $\leftarrow w_9$  *//  $w_9 = 0.50$*   
**Else** CostID  $\leftarrow 1$ .  
**End if**  
**Return** CostID

Fig 9. The function of insertion and deletion operations.

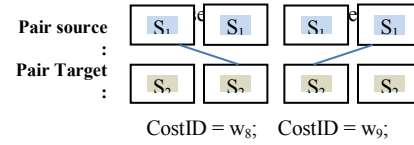


Fig 10. Modified weighted case 6 and case 7 of operations

Figure 11. shows the pseudo code of the HL-Big-A algorithm. The suggested HL-Big-A algorithm is a hybrid method of mixed component, Edit Distance and Bigram algorithms. Taken into consideration the storage space of process, saving the time of processing time and the time complexity, this proposed HL-Big-A algorithm will give accuracy and efficiency for Arabic names matching when applied on the Arabic language. The main goal of the proposed HL-Big-A algorithm is to achieve more accurate and efficient Arabic name matching.

**Algorithm.** Hybrid Levenshtein and Bigram in Arabic (HL-Big-A)  
**Input:** Two words (source, target)  
**Output:** Minimum Distance (MD)  
**Decimal** Hybrid Levenshtein based on Bigram in Arabic (source, target)  
 $S = - \& \text{source}$  *// add prefix for source and target.*  
 $T = - \& \text{target}$  *// initial segments*  
 $MD[0,0] = 0$   
**for**  $i=0$  **to**  $m$   
     $MD[i, 0] = i$   
     $x_{[i]} = S.\text{Sub-string}(i, 2)$  *// join unigram to Bigram ,n=2 for source*  
**end for**  
**for**  $j=0$  **to**  $n$   
     $MD[0, j] = j$   
     $y_{[j]} = T.\text{Sub-string}(j, 2)$  *// join unigram to Bigram ,n=2 for target*

---

```

end for
for i from 1 to length(source)
  for j from 1 to length(target)
    s1 = x[i-1];
    s2 = y[j-1];
    s11 = s1.Sub-string (0, 1); // split Bigram to unigram for
source and target.
    s12 = s1.Sub-string (1, 1);
    s21 = s2.Sub-string (0, 1);
    s22 = s2.Sub-string (1, 1);
    Cost = Call Cost Substitution and Transposition Op(S11,S12,S21,S22)
// Figure 9.
    CostID= Call Deleting and Inserting Op(S11,S12,S21,S22)
// Figure 11.
    MD [i, j] =minimum(MD(i-1, j) + CostID, MD (i, j-1) + CostID, MD
(i-1, j-1) + Cost)
  end for
end for
Return MD [n, m]

```

---

Fig 11. The pseudo code of the HL-Big-A algorithm.

## VI. THE EXPERIMENT AND RESULTS

This section presents the experiments that have been implemented in this paper to show the proposed HL-Big-A algorithm and compare it against different compared algorithms. The performance measurement has been elaborated in sub-section D.

In the following sub-sections, A presents the data preparation of Arabic datasets which are used in this paper and sub-sections B. a variety of weights is presented: ( $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8$  and  $w_9$ ) followed by a thorough analysis, while the sub-sections C. is experimental results.

### A. Preparation of data in Arabic Datasets

This section describes the names of Arabic being used to examine the suggested algorithms for matching Arabic names. For more investigation, a collection of datasets has been used in these experiments for testing the proposed HL-Big-A algorithm and comparing it against different compared algorithms. Because no standard collection of names exists, therefore, 5 datasets have been extracted manually as shown in table 8.

Table 8. Arabic names of datasets.

No	Database Name	Description	Source
1	Dataset 0 (10pairs)	Arabic Names	[26]
2	Dataset 1 (11 pairs)	Arabic Text Recognition (ATR) data, which is generated from an OCR system developed at KFUPM	[39]
3	Dataset 2 (50 pairs)	Arabic offline handwritten text database	[38]
4	Dataset 3 (10 pairs)	is prepared by manual transliteration of foreign words from English to Arabic.	[40]
5	Dataset 4 (600 pairs)	These obtained from the employees database of Ministry of Education in Yemen.	[21])

All the datasets have Arabic. Each dataset cont All the datasets have Arabic. Each dataset contains some names with different

possible variations (such as spelling errors and typographical) of same names. A collection of the all kinds of variation has been considered in the variety of datasets

### B. The Weights ( $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8$ and $w_9$ ) Analysis

These cases are weighted as symbols  $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8$  and  $w_9$ . These weights depend on tables 3., 4. and 5 as aforementioned, which are used to adapt to the operational environment and get more accurate and efficient results on different situations [33].

### C. Experimental Results

In the section, a comparison study is carried out to evaluate the performance of the proposed HL-Big-A algorithm.

The first experiment was carried out for the proposed HL-Big-A algorithm and the original N-DIST-A algorithm as a compared algorithm with N equals to Bigram ( $B_i=2$ ). This experiment was carried out to evaluate the performance of the proposed HL-Big-A algorithm. This experiment has been carried out for the proposed HL-Big-A algorithm and LD, MDL, MDLD, N-DIST and N-DIST-A algorithms as compared algorithms with  $B_i$  when N equals 2. This experiment is carried out based on Dataset 0 which has pairs 10 of names. The result of this experiment is shown in table 8. The HL-Big-A Algorithm gives better results than the LD, MDL, MDLD, N-DIST and N-DIST-A algorithms, especially when comparing names containing “ء”, “hamza”, or dots as shown in table 9. For examples, the names in 3 and 9 rows. Unlike the LD, MDL, MDLD, N-DIST and N-DIST-A algorithms the HL-Big-A algorithm is sensitive to phonetic similarity as shown in 2, 3, and 5 rows. The HL-Big-A Algorithm gives better results than the LD, MDL, MDLD, N-DIST and N-DIST-A algorithms, especially when comparing names transposition as shown in 8 rows. The HL-Big-A algorithm shows many advantages over the LD, MDL, MDLD, N-DIST and N-DIST-A algorithms as aforementioned.

Therefore, the HL-Big-A algorithm gives more accurate results than the LD, MDL, MDLD, N-DIST and N-DIST-A algorithms for all pairs in dataset 0 as shown in table 9.

In order to understand how the editing operations in the HL-Big-A algorithm works with the variation of names, it will be elaborated in detail in the following examples and as shown in table 10 and table 11. Table 10, shows how the HL-Big-A algorithm measures the distance between the name 1 “عصار” and name 2 “عسار” as the first step. The distance is 0.17; therefore, the similarity between them is 96%. It is obvious that the HL-Big-A algorithm gives a very low cost for replacing ‘ص’ with ‘س’ from name 2 into name1 due to their form similarity. Table 11, shows how the HL-Big-A algorithm measures the distance between the name1 “كامل” and name 2 “كمال” as the first step. The distance is 0.50; therefore, the similarity between them is 88 % that is calculated by equation (3). It is obvious that the extended algorithm gives a very low cost for transposition “ل” with “ك” from the target (t) into the source(s) due to their form similarity.

TABLE 9. COMPARISON BETWEEN ALGORITHMS FOR ARABIC DATASET.

	String.		Compared Algorithms					Proposed Algorithm
No.	<i>S</i>	<i>T</i>	LD	DLD	MDLD	N-DIST	N-DIST-A	HL-Big-A
1	عمار	عمار	0.80	0.80	0.80	0.80	0.95	0.95
2	عصار	عصار	0.75	0.75	0.75	0.75	0.89	0.96
3	انباء	امباء	0.80	0.80	0.80	0.80	0.90	0.97
4	باهر	ناهر	0.75	0.75	0.75	0.63	0.86	0.83
5	سنا	ثناء	0.50	0.50	0.50	0.38	0.89	0.94
6	مراد	موراذا	0.60	0.60	0.60	0.60	0.75	0.78
7	اسماء	اسما	0.60	0.60	0.60	0.50	1.00	1.00
8	كامل	كمال	0.50	0.75	0.75	0.50	0.63	0.88
9	المرء	المر	0.60	0.60	0.60	0.50	0.80	1.00
10	عبدالله	عبد الله	0.88	0.88	0.88	0.81	1.00	0.97
AVERAGE (PERCENTAGE SIMILARITY)			0.68	0.70	0.70	0.63	0.87	<b>0.93</b>

TABLE 10. THE DISTANCE BETWEEN عصار → عصار IN THE HL-Big-A ALGORITHM WITH BI.

		ع-ع	س ع	ا س	ر ا
	0	1	2	3	4
ع-ع	1	0.00	1.00	1.50	2.50
ص ع	2	1.00	0.16	0.71	1.71
ا ص	3	1.50	0.71	0.11	1.09
ر ا	4	2.50	1.71	1.08	<b>0.17</b>

TABLE 11. THE DISTANCE BETWEEN “كامل” → “كامل” IN THE HL-Big-A ALGORITHM.

		ك-ك	م ك	ا م	ل ا
	0	1	2	3	4
ك-ك	1	0.00	0.50	1.00	2.00
ك ا	2	0.50	0.25	0.58	1.08
ا م	3	1.50	0.83	0.25	0.92
م ل	4	2.50	1.33	0.92	<b>0.50</b>

Furthermore, more experiments have been carried out with a variety of datasets to get evidence of HL-Big-A algorithm ability. Four datasets are selected and applied on the LD, DLD, MDLD, N-DIST, N-DIST-A and HL-Big-A algorithms with N=2, Bigram as shown in table 12. That gives the evidence of HL-Big-A algorithm ability in Arabic name matching. table 12 shows the accuracy of the percentage similarity as a mean for all datasets. In this table, the HL-Big-A algorithm gets 93.4% while LD, DLD, MDLD, N-DIST and N-DIST-A algorithms get 79%, 79%, 79% and 76% and 87.3%, respectively.

Therefore, the HL-Big-A algorithm gives more accurate results than the LD, DLD, MDLD, N-DIST and N-DIST-A algorithms for all datasets, because N-DIST-A algorithm has not considered the transposition and multi-transposition operations of the Arabic language, for example, the result in 6 rows as shown in table 13.

TABLE 12. THE MEAN SIMILARITY OF LD, DLD, MDLD, N-DIST, N-DIST-A AND HL-Big-A ALGORITHMS WITH DIFFERENT DATASET.

Dataset names	Compared Algorithms					Proposed Algorithm
	LD	DLD	MDLD	N-DIST	N-DIST-A	HL-Big-A
Dataset 1	0.76	0.76	0.76	0.73	0.823	0.895
Dataset 2	0.80	0.80	0.80	0.81	0.901	0.938
Dataset 3	0.81	0.81	0.81	0.77	0.800	0.916
Dataset 4	0.79	0.80	0.80	0.74	0.967	0.986
Similarity Mean	0.79	0.79	0.79	0.76	87.3	<b>93.4</b>

TABLE 13. THE DISTANCE AND SIMILARITY BETWEEN مجدي → مجيد IN THE ALL ALGORITHMS.

No.	Source (مجيد) and Target (مجدي) algorithms	Distance	Similarity
1	LD	3	0.40
2	DLD	3	0.40
3	MDLD	3	0.40
4	N-DIST	2.5	0.50
5	N-DIST-A	0.918	0.816
6	HL-Big-A	0.796	0.841

## VII. CONCLUSION

String matching is a tedious task and a big field because of the development of many algorithms, the study of methodology, the complexity, and the limitations imposed on them. To improve the matching of Arabic names accurately and efficiently, this work focuses on a proposed framework, employing a new methodology. The proposed framework includes affixing, joining unigram to bi-gram and split bi-gram to unigram, and proposing algorithm for characteristics and special features of the Arabic language of many weighted, including keyboard distance, phonetic similarity, and character form similarity and the performance measurement. The main contribution of this paper is the methodology of dealing with the transposition, substitution, deletion and insertion operations. The transposition, substitution, deletion and insertion are handled differently at the different states of the matching Arabic name to be improved. Furthermore, substitution, transposition, insertion and deletion operations are given different weights. These weights depend on three levels of similarity that are used to adapt to the operational environment and get more accurate results in different situations. Another contribution is reducing the storage space of the process, saving the time of processing time and reducing the time complexity from  $O(N^3)$  to  $O(N^2)$ . Therefore, the given algorithm (HL-Big-A) gives more efficient and accurate results than the previous algorithms. In the future, this algorithm can be improved so that it can applied to the token name in Arabic or any other language.

## REFERENCES

- [1] Navarro, G., "A guided tour to approximate string matching," ACM computing surveys, vol. 33, no. 1, pp. 31-88, . 2001.
- [2] Christen, P., "A Comparison of Personal Name Matching Techniques and Practical Issues", Technical Report TR-CS-06-02, Joint Computer Science Technical Report Series, Department of Computer Science, 2006.
- [3] Cheng, G., Wang, F., Haiyang, L.v, Zhang, Y., "A New Matching Algorithm for Chinese Place Names", IEEE , 2011.
- [4] Delgado, J., Galárraga, F., Fuertes,W., Theofilos Toulkeridis, Cesar Villacís, Fidel Castro, "A Proposal of an Entity Name Recognition Algorithm to Integrate Governmental Databases", 2016.
- [5] Charras,C., Lecroq, T., " Handbook of exact string matching algorithms", King's College Publications, 2004.
- [6] Al-Ssulami, A.,M., , "Hybrid string matching algorithm with a pivot", Journal of Information Science, Vol. 41(1) 82–88, 2015.
- [7] Mark, v., "The stringdist package for approximate string matching". The RJournal 6(1) 111-122, 2014.
- [8] Cantone,D., Cristofaro, S., and Faro, S., "Efficient string-matching allowing for non-overlapping inversions," Theoretical Computer Science, 2012.
- [9] Beernaerts, Jasper., Debever, E., Lenoir, M., De Baets, B., & Van de Weghe, N.). "A method based on the Levenshtein distance metric for the comparison of multiple movement patterns described by matrix sequences of different length". Expert Systems with Applications, 115, 373-385, 2019.
- [10] Gueddah H., Yousfi A., and M. Belkasmi, "Introduction of the weight edition errors in the levenshtein distance," International Journal of Advanced Research in Artificial Intelligence, vol. 1, no. 5, pp. 30–32, 2012.
- [11] Gueddah H. and Yousfi A., "The Impact of Arabic Inter-character Proximity and Similarity on Spell-Checking," in Intelligent Systems: Theories and Applications (SITA), 2013 8th International Conference on, p. 162, IEEE, 2013.
- [12] Gueddah, H., Yousfi, A., & Belkasmi, M. "The filtered combination of the weighted edit distance and the Jaro-Winkler distance to improve spellchecking Arabic texts". IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA) (pp. 1-6). IEEE, 2015.
- [13] Hamza , B. and Gueddah, H. "For an Independent Spell-Checking System from the Arabic Language Vocabulary", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 5, No. 1, 2014.
- [14] Mustafa, S. H., and Al-Radaideh, "Using N-Grams for Arabic Text Searching," Journal of the American Society for Information Science and Technology, vol. 55, no. 11, pp. 1002–1007, 2004.
- [15] Mustafa, S. H., and Al-Radaideh, "Character contiguity in N-gram-based word matching: the case for Arabic text searching," Information Processing & Management, vol. 41, no. 4, pp. 819–827, 2005.
- [16] Chowdhury,S., Bhattacharya U, Parui SK. "Online Handwriting Recognition Using Levenshtein Distance Metric". Document Analysis and Recognition (ICDAR), 12th International Conference;p.79,83, 25-28, 2013.
- [17] El-Shishtawy, T., "Linking Databases using Matched Arabic Names ", Computational Linguistics and Chinese Language Processing ,Vol. 19, No. 1, pp. 33-54, 2014.
- [18] Paleo B. W., C. G. G. Hita, J. d. C. Lima, C. Ribeiro, and J. Jambeiro Filho, "A Modified Edit-Distance Algorithm for Record Linkage in a Database of Companies," 2014.
- [19] Alsayadi, A.H. . and ElKorany, M.A., "Integrating Semantic Features for Enhancing Arabic Named Entity Recognition" ,International Journal of Advanced Computer Science and Applications, Vol. 7, No. 3, 2016.
- [20] Mohamed, A., "An improved algorithm for information hiding based on features of Arabic text: A Unicode Approach" Egyptian Informatics Journal, vol. 15, no. 2, pp. 79–87, 2014
- [21] Al-Sanabani, M., Al-Hagree, S.,"Improved An Algorithm For Arabic Name Matching". Open Transactions On Information Processing ISSN(Print): 2374-3786 ISSN(Online): 2374-3778.2015. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.842.5353&rep=rep1&type=pdf>
- [22] Al-Helali, M.B. and Mahmoud, A.S."A Statistical Framework for Online Arabic Character Recognition", Cybernetics and Systems, 47:6, 478-498, 2016
- [23] Levenshtein V. I., "Binary codes capable of correcting deletions, insertions, and reversals," Soviet Physics, pp. 845–848, 1966.
- [24] Damerau, F., "A Technique for Computer Detection and Correction of Spelling Errors", Communications of the ACM, 7(3), pp.171-176, 1964.
- [25] Aqeel S.U, Beitzel S.M, Jensen E.C, Grossman D. & Frieder O. , " On the Development of Name Search Techniques for Arabic". Journal of the American Society for Information Science and Technology (JASIST) 57(6): hlm. 728-739. 2006.
- [26] Ghafour, H. , EI-Bastawissy , A., Fattah, A. , " AEDA: Arabic Edit Distance Algorithm Towards A New Approach for Arabic Name Matching", IEEE international conference, IEEE Trans. Pattern Analysis and Machine Intelligence, 15(9):926–932,2011. <https://ieeexplore.ieee.org/document/6141061/>
- [27] Aouragh S., Gueddah, H., Yousfi, A." Adaptating the Levenshtein Distance to Contextual Spelling Correction". International Journal of Computer Science &Applications;. p.127-133, 2015.
- [28] Boehmer and Ton, is available on <https://confluence.csiro.au/public/taxamatch/the-mdld-modified-damerau-levenshtein-distance-algorithm> ,2008.
- [29] Ullmann, J.R.: "A binary n-gram technique for automatic correction of substitution deletion, insertion and reversal errors in words". The Computer Journal 20(2), 141– 147, 1977.
- [30] Kondrak, G, "N-gram similarity and distance", In M. Consens and G. Navarro (eds.), Proceedings of the String Processing and Information Retrieval 12th International Conference, Buenos Aires, Lecture Notes in Computer Science, 3772, Springer, Heidelberg, Germany, 115–126, 2005.
- [31] Abdulhayoglu, M. A., & Thijs, B. "Matching bibliometric data from publication lists with large databases using n-grams". In Proceedings of 14th international society of scien to metrics and informetrics conference (ISSI), Vienna, Austria, Vol. 2, pp. 1151–1158, 2013.
- [32] Abdulhayoglu, M. A , Bart Thijs , Wouter Jeuris , "Using character n-grams to match a list of publications to references in bibliographic databases", DOI 10.1007/s11192-016-2066-3,2016.
- [33] Christian Eduardo Millán-Hernández , René Arnulfo García-Hernández(&) , Yulia Ledeneva , and Ángel Hernández-Castañeda , "Soft Bigram Similarity to Identify Confusable Drug Names ", J. A. Carrasco-Ochoa et al. (Eds.), LNCS 11524, pp. 433–442, MCPR 2019.

- [34] Alsurori, M., Al-Sanabani, M., & Salah, A. H. ,“ Design an Accurate Algorithm for Alias Detection ”, ISSN: 2074-9023 (Print), ISSN: 2074-9031 (Online), 2018..
- [35] Gomaa, W. H., Fahmy, A. A., “Automatic scoring for answers to Arabic test questions”, *Computer Speech & Language* 28:4, 833-857,2013.
- [36] Ahmed , F. and Nürnberger ,A. “N-grams Conflation Approach for Arabic,” in *ACM SIGIR Conference*, Amsterdam, 2007.
- [37] Habash, N. “Introduction to Arabic Natural Language Processing”. Morgan & Claypool Publishers, 2010.
- [38] Mahmoud S., Ahmad, I., Al-Khatib,W. G., Alshayeb, M., Parvez, M. T., argner, V. M”, and Fink, G. A., “KHATT:An open Arabic offline handwritten text database,” *Pattern Recognition*, vol. 47, no. 3, pp. 1096–1112, 2014. <https://www.sciencedirect.com/science/article/abs/pii/S0031320313003300>
- [39] Mahdi, Adnan.. “Spell Checking and Correction for Arabic Text Recognition.” Master’s thesis, KFUPM University, Department of Information & Computer Science., 2012, <http://eprints.kfupm.edu.sa/138619/>
- [40] Nwesri, A. F., Tahaghoghi, S., and Schole, F., “Finding Variants of Out-of-Vocabulary Words in Arabic,” in *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, pp. 49–56, Association for Computational Linguistics, 2008 <https://dl.acm.org/citation.cfm?id=1654586>
- [41] Hakak, S., Kamsin, A., Shivakumara, P., Gilkar, G. A., Khan, W. Z., & Imran, M. “ Exact String Matching Algorithms: Survey, Issues, and Future Research Directions. ”, *IEEE Access* ,2019 .
- [42] Salah, A. H ,&Al-Sanabani, M.. “A Framework For Name Matching In Arabic Language”, *1st Scientific Conference on Information Technology and Networks* ,2016. .