PERSONAL IDENTITY MATCHING

Mazin Al-Shuaili[1] and Marco Carvalho[2]

[1,2]Florida Institute of Technology, Melbourne, USA
Malshuaili1994@my.fit.edu, mcarvalho@cs.fit.edu

ABSTRACT

Despite all existing methods to identify a person, such as fingerprint, iris, and facial recognition, the personal name is still one of the most common ways to identify an individual. In this paper, we propose two novel algorithms: The first one uses sound techniques to create a multi-dimensional vector representation of names to compute a degree of similarity. This algorithm compares names that are written in different languages (cross-language) and transliterated names in English. The second algorithm builds on the first one and measures the similarities between full names, taking into account the full name structure. We evaluate the algorithms for transliterated names and across languages using Arabic and English as an example. Significant results are achieved by both algorithms compared with other existing algorithms.

KEYWORDS

Border Security, Cross-Language Information Retrieval, Information Retrieval, Security Screening System.

1. INTRODUCTION

There are many methods to identify a person, including biometrics, such as fingerprint, iris, and facial recognition. Despite the availability of these techniques, the name of the person is more commonly used to identify an individual, especially in border control measures, criminal investigations, and intelligence analysis. The personal name is used when people apply for visas or on their arrival at a port (land-port, seaport, or airport). The International Civil Aviation Organization (ICAO) recommends all passports to contain personal names in the English language; therefore, countries have worked to provide the names of passport holders in the English alphabet. This transliteration causes a problem with spelling variations, such as with name "Mohammed," as shown in Table 1. In this case, some systems, such as security screening systems, need to deal with those differences when one-to-one comparison does not work correctly.

There are many screening systems that identify a given name, such as that of a traveler or suspect, with a list of names. The No- Fly List, for example, was created by the Transportation Security Administration (TSA) and is now used by the Terrorist Screening Center (TSC) in the United States [1]. This list contains the names of all people who are not permitted to travel to or from the United States. Also, the manifest is a document listing passengers and crew information that used to be sent by fax. Today, this information is sent electronically and managed by an advanced passenger information (API) system. API sends the passengers' information (manifest) in real time to their destinations while they are checking in for their flights. A response comes from the destination with an indicator saying whether the passenger is allowed to board. In addition, Interpol is an intergovernmental organization that maintains a list of suspects that is shared among most countries. The names in the mentioned lists are checked with the private lists in each country or even each security sector separately using screening systems. Those systems might return a result of comparing two names as negatively matched (False Positive) or negatively unmatched (False Negative) as a consequence of name variations.

Table 1. Spelling Variations of Mohammed

Name	Nationality
MAHMMED	EGYPT
MOHMED	INDIA
MHOHAMMED	BANGLADESH
MUHAMMADE	INDIA
MAHOMED	SULTANATE OF OMAN
MUHMMET	TURKEY
MUHAMMET	TURKEY
MOHD	MALAYSIA

There have been many cases in which a person was arrested or stopped on suspicion of his or her name on the wanted lists. Senator Edward Kennedy was stopped serval times while boarding an aircraft because his name matched someone on the No-Fly List [2]. In another case, Congressman John Lewis was stopped for an additional security check. Sixty-tow-year-old Sister McPhee is an education advocate for the Catholic Church who was stopped repeatedly for nine months because her name matched that of an Afghani man whose name was included on the No-Fly List. There have been several occasions when infants and toddlers were stopped from boarding airplanes because their names negatively matched with names on the No-Fly List [2]. Screening systems must minimize these types of errors.

The side effect of screening systems is that common names are often written with different spellings for several reasons, including natural spelling variations, phonetic variations, and compound names [3], [4]. Transliteration causes spelling variations for a single name due to phonetic similarity [5], [6]. The names on the Interpol list and the API list are Romanized (transliterated). In 1918, Soundex was published as the first algorithm that tried to solve this type of problem by indexing names based on their sounds. Soundex studied the letters of the English alphabet and grouped them into seven groups, one of which is ignored by the Soundex algorithm. Section 2 discusses Soundex in more detail.

Cross-Language Information Retrieval (CLIR) also struggles to deal with proper names because they are out of vocabulary (OOV) [1], [7], [8], [9]. Furthermore, the authors in [7] demonstrated that about 50% of OOV words are proper names. Proper names might affect the performance of English-Arabic cross-language transliteration by 50% [10]. One solution for this problem is to transliterate all names from one language to another language so that all names have the same format (characters) to be compared, as applied in [10]. Transliteration of names from one language to another is not easy and is expensive due to phonetic variations in some languages [1], [6]. Also, a static solution, such as a dictionary, gives high performance but needs to be updated periodically with new names. When people borrow names from other cultures or invent new names, the number of names increases. We need a solution that can deal with these permanent changes. Names in all languages must be standardized in order to be compared accurately and efficiently for CLIR.

The proposed algorithm, Personal Identity Matching (PIM), defines names as points in seven-dimensional space. The dimensions represent six groups of characters that are classified by Soundex plus a seventh group that represents the characters that are ignored by Soundex. Once PIM sets the names in space, then the distance is calculated between those points. The distance is converted to the similarity between 1 for an exact match and close to zero for no match. It is clear that PIM is unlike Soundex in how it compares names; PIM uses distance, whereas Soundex uses code. Our result shows that PIM has 33% better accuracy than Soundex and 11% better accuracy than all of the other algorithms involved our testing. PIM computes the similarity between individual names, but for a full name we create another algorithm (PIM-Full name or PIMF). PIMF breaks a full names into tokens (individual names), then it sends a pair of names into PIM, taking into account the full name structure.

2. Related Work

Name matching is divided into two approaches. First, a phonic algorithm tries to avoid common problems of name variation by encoding names based on their sound [5]. Many phonetic algorithms have been introduced since Soundex, such as Metaphone, Cologne, NYSIIS**,** and Caverphone. They have gone through several stages since the early 1900s, when Margaret K. Odell and Robert C. Russell introduced the Soundex algorithm. The latest phonetic algorithm is Metaphone 3, created by Lawrence Philips in 2009 [11]. The same author created three versions of Metaphone, the third of which (Metaphone 3) was made for commercial purposes. Although all of the algorithms have different rules for converting a name into a code, they have common criteria: 1) They target a specific language or even for a specific database, such as Caverphone, which was created to match the elections lists generated between the late 19th century and early 20th century. 2) They generate code for each name: two names are matched only when both of their codes are the same; otherwise they are unmatched. 3) Most of them have codes of a fixed length, except Refined-Soundex has an unlimited length of code. Soundex has the fewest rules of all phonetic algorithms.

Soundex primarily groups characters based on their sounds, as shown in the next table (Table 2). The sounds of those letters are based on English pronunciation, and all of the letters' sounds are clustered into six groups. Other characters (Group 0), including {a, e, i, o, u, w, h, y}, are ignored unless they appear in the first character of the name. The Soundex algorithm keeps the first character without changing it because the first character is clearly pronounced; therefore, there are fewer errors made at the beginning of a name [12]. The Soundex code is a combination of a letter and three digits. The name "Jackson," for example, is converted to the code "J250."

In phonetic algorithms that generate code for each name, the comparison between names is a one-to-one relation (match or mismatch), and there is no concept of distance or similarity between two names. Therefore, alternative algorithms are needed to provide a better result that gives a degree of similarity between one string (*S1*) and the other string (*S2*). The most common algorithms that calculate a distance or a similarity are the Levenstein distance (edit distance), Jaro-Winkler, Smith-Waterman, Damerau-Levenstein, and N-gram algorithms.

Table 2. Soundex for English and Arabic

No	English Letters	Arabic Letters
0	A, E, I, O, U, W, H, Y	ا,ع,ي,هـ,ح,و
1	B, F, P, V	ف,ب
2	C, G, J, K, Q, S, X, Z	س,ش,ك,غ,ج,خ,ق,ص,ز
3	D, T	د,ض,ظ,ت,ذ,ث,ط
4	L	ل
5	M, N	م,ن
6	R	ر

Edit distance (Levenstein distance) is the minimum number of edit operations needed to convert S_1 into S_2; where each insertion, deletion, or substitution counts as one operation [13]. The minimum number of edits is divided by the longest string to calculate the distance between two strings. Damerau-Levenstein **distance includes** extra operation, a transposition, in addition to the other operations that are used by edit distance. The Smith Waterman algorithm is similar to the edit distance one but uses a phonetic algorithm to enhance the edit distance; e.g., if $a_i \approx b_j$, then the score is 2 ($a_i \& b_j$ are in the same group in Soundex table), or If $a_i = b_j$, then the score is 5; otherwise it is -5 [3], [14]. Jaro distance is based on common characters between two strings, S_1 and S_2, with half of the longest string used for transposition [3], [12]. It is commonly used with duplicate detection (data linkage) when a name needs to match other names within some distance threshold [3]. The Jaro-Winkler algorithm improves the Jaro algorithm by boosting the error score in the prefix characters. Some other algorithms that calculate distance divide a name into tokens such as n-gram and longest common sub-string (LCS). All those algorithms use a threshold to determine matching names.

The algorithms mentioned cannot deal with a full name structure. The full name is composed of more than one name usually a combination of a first name and a surname. In addition, a middle name (often the father's name) might be added between the first name and surname to increase the determination of personal identity. The middle name can be a father's name that is followed by his father's name, and more ancestors' names can be added recursively. In addition, in some countries the last name is followed by the first name, then the other names. We generalize the full name structure using a regular expression as follows: $F = n(m_i)^* l \mid (l, |l) n(m_i)^*$ where n is the first name, l is the last name, and m_i represents all other names in the sequence, starting from i=1; 1 is for the father, 2 is for the grandfather, and so on.

$$sim_{MongeElkon}(A,B) = \frac{1}{|A|}\sum_{i=1}^{|A|} \max\{sim'(a_i,b_i)\}_{j=1}^{|B|} \qquad (1)$$

There are a few algorithms that deal with full names, and the most interesting one is the Monger-Elkan algorithm. Monge-Elkan distance measures the similarity between two full names that contain at least given names and last names. Each token (name) is calculated using a similarity function, which is $sim'(a_i, b_i)$, such as edit distance or the Smith Waterman algorithm. Then, the average number of tokens in A is computed. The above equation is the Mange Elkan distance to measure A and B, where A, the first full name, contains $\{a_1 \ldots a_n\}$ tokens and B, the second full name, contains $\{b_1 \ldots b_m\}$ tokens.

3. ARABIC-TO-ENGLISH MAPPING

There are more than four hundred million Arabic language speakers who live in 22 countries from the Arabian Peninsula across North Africa to the Atlantic Ocean [15]. Arabic is the main language of newspapers, TV news, books, and official works, even though there are different accents across countries and even within countries. In addition, there are more than 1.6 billion Muslims, and most of them have an Arab name. Table 1 shows various spellings of the name "Mohammed" in different nations; there are more than 300 different spellings for the name "Mohammed" alone [2].

This section explains in detail the mapping of each Arabic letter to its closest English letter, as well as their relationships with Soundex groups. There are four groups of Arabic letters that are mapped to English letters:

- Exact match: some of the Arabic letters can be mapped to English letters that have almost the same sound. The EXACT column in Table 3 shows these letters.
- Exact match with two characters: "TH" can be mapped to two Arabic letters (ذ, ث), while the letter "ش" is mapped to either "SH" or "CH."
- Close match: these letters do not have an equivalent sound in English; they are mapped to a character that produces a similar sound (see the Close column in Table 3).
- Diacritic sound: Hamza (ء) and three more diacritics change the sound of some Arabic letters, such as "ا," which can be mapped to A, O, U, E, and I (see the Others column in Table 3).

Table 3. English Arabic Characters Mapping

	EXACT	CLOSE	OTHER
A	ا	ع	ا أ آ ى ء
B	ب		
C			س ش ك
D	د		ض ظ
E			إ ي ئ
F	ف		
G			ج غ
H	ح هـ		
I			إ ى ئ ي
J	ج		
K	ك		خ
L	ل		
M	م		
N	ن		
O			ع و ؤ أ
P			ب
Q		ق	ك
R	ر		
S	س	ص	ش
T	ت	ط	ظ ض ذ ث
U			ع و أ
V			ف
W	و		
X			
Y	ي		
Z	ز		ظ

The authors in [16] demonstrate that Arabic and English lack the corresponding sounds to allow one-to-one matching that ends ambiguity in letter mapping, as the results show in the CLOSE column of Table 3.

The rows in Table 2 are grouped based on Soundex representation, where each row represents a number in the Soundex table. Table 2 has an extra column (Arabic Letters) that represents Arabic letters after they are clustered. Consequently, the results show there is no overlap based on Soundex groups except for the letter "ظ," which is due to some countries' diverse dialects or non-Arabic languages, such as Persian or Urdu.

All letters agree in both languages and have similar enough sounds to generate the same Soundex groups. Therefore, Soundex is the best choice over all the other phonetic algorithms that are English dependent. Adding more groups, such as Refined Soundex, introduces overlap for same Arabic characters; e.g., "ق" (Qaf) points to both K and Q, which are in two different groups in Refined Soundex. This causes a comparison problem for Arabic names such as "Qaddafi" and "Kaddafi," or "Qasim" and "Kasim."

4. Personal Identity Matching (PIM) and Preprocessing

The PIM algorithm converts names to points in space that have seven attributes (seven-dimension) representing the groups in the Soundex table, including the ignored characters. The following steps show how we transform Soundex groups to a seven-dimensional vector:

1. Use each row (group) in the Soundex table as an independent group.

2. Add all letters that are ignored by Soundex into the table as group 0 (silent char.).

3. PIM now has 7 groups (0 to 6); refer to Table 2.

4. Assign a value for each character. The difference between two characters ($|\alpha - \beta|$) represents the distance between these characters in the same group. We created a handcrafted model that contains each character value.

5. Create a Vector-Consonant (V_c) with 7 dimensions to hold consonant values, group 1 to group 6, including group 0 if and only if it is the first character in the name. For example: $V_c(\text{'Nasser'}) = \{null, null, \{S, S\}, null, null, \{N\}, \{R\}\}$; and $V_c(\text{'Ali'}) = \{A, null, null, null, \{L\}, null, Null\}$.

6. For each attribute in V_c, create a vector of a 3-dimensional representation to hold group 0 characters when they do not appear as the first character. The characters are divided into three groups: "e", "i", and "y" in the first group, "a" and "h" in the second group, and "o", "u", and "y" in the third group. For example, the name "Nasser" is converted to $\{\{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}, \{E, \phi, \phi\}, \{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}, \{\phi, A, \phi\}, \{\phi, \phi, \phi\}\}$; and $V_g(\text{'Ali'}) = \{\{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}, \{\phi, I, \phi\}, \{\phi, \phi, \phi\}, \{\phi, \phi, \phi\}\}$ where ϕ is a null value.

Soundex retains the first letter of any name at the beginning of the code, including characters in group-0. In addition, the Jaro-Winkler algorithm improves the Jaro algorithm by boosting the error score in the prefix characters. The Jaro-Winkler algorithm adds more values with a common prefix at the beginning of the names, up to four letters [12]. The first character(s) of the name is pronounced clearly, and it is rarely spelled incorrectly. Therefore, the PIM algorithm treats each letter in group-0 the same as the letters in the other groups (consonants) if and only if they appear as the first letter. For this reason, there are two values for each character: one value is for the first character of the name, and the second value is for any character except the first character. However, some letters have very close pronunciation even if they appear as the first character of the name, e.g., Auto and Otto. The beauty of PIM is that it can easily manage the similarity between characters. If A and O contain the same value, then they are identical; if the different value between these two characters increases, the similarity decreases (see step 4).

5. Proposed Algorithm

Each name passes to a function that is called a vector function generator. This function accepts three parameters: Name (in), V_c (out), and V_g (out). After the first name is passed to the function, two vectors (V_{c1}, V_{g1}) represent the first name. The second name is then passed to the same function to produce two other vectors (V_{c2}, V_{g2}). Once the vectors are generated for all names, Euclidean distance (Ed) is used to calculate the distance between the names. The calculation performed as following:

1. Calculate → $Ed_C\ (V_{c1}, V_{c2})$

2. Calculate → $Ed_G\ (V_{g1}, V_{g2})$

3. G (Total silent char.) → G = G / 2

4. N (Total char.) N = G + C

5. C (Total consonant) → C = N – G, add more weight to C if silent chars. are not equal in both names

6. Convert both distances to similarities: Sim_C and Sim_G

7. Finally use (2) to calculate PIM similarity

$$Sim_{pim} = \frac{C(Sim_C) + G(Sim_G)}{N} \tag{2}$$

C=Consonant, G=Group-0, & N=Total chars.

To convert a single name to a vector that has seven attributes, the algorithm needs to insert the value of each character into a suitable attribute. This conversion costs $O(|n|)$, where n is the number of characters in the name. If the algorithm compares two names, the complexity is $O(|n|+|m|)$, where n is the length of the first name, and m is the length of the second name. Similarity calculation is constant, as described above. Therefore, total complexity of PIM is $O(|n|+|m|)$. In addition, if there is a list of names, the comparison performance can be improved by storing the vectors of the names in a repository. For future comparison, PIM needs only to convert a given name to a vector and it calculates the similarity between the given name vector and the vectors in the repository. PIM is used for a single name and cannot understand full name structure. Therefore, we need another algorithm that can deal with such structure.

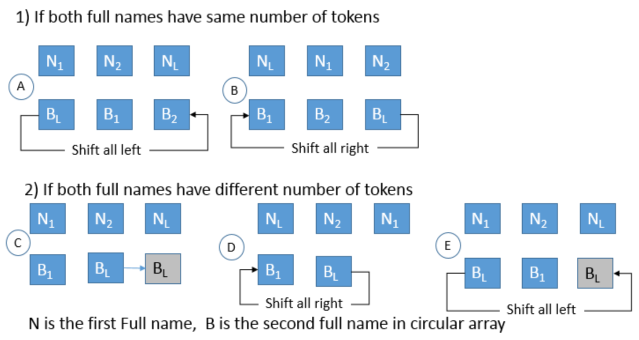
1) If both full names have same number of tokens
A
B
Shift all left
Shift all right
2) If both full names have different number of tokens
C
D
E
Shift all right
Shift all left
N is the first Full name, B is the second full name in circular array

Figure 1. Unmatched Token Adjcement (Names).

We create another algorithm (PIMF) that is an extension of PIM. PIMF comprehends full name structure, as we explained earlier. PIMF accepts the two full names and breaks them into tokens (names). Then, it sends the tokens to PIM to calculate the similarity. PIMF uses the following process:

1. Break both full names into two lists of tokens: $a_1, a_2 \dots ,\ a_n$ and $a_1, a_2, \dots ,\ a_m$

2. Set FNS = Full name with least tokens

3. For i=1 to FNS Length TotalSum = TotalSum + PIMSimilarity($a_i, b_{i,}, thershold$)

4. If the average of TotalSum > threshold, return TotalSum / FNS length;

5. Return the maximum (Similarity * threshold) based on the comparisons after the following movements:

    a. Shift all tokens in FNS to the left (see Figure. 1 A & E)

    b. Shift all tokens in FNS to the right (see Figure. 1 B & D.)

    c. If the number of tokens is not equal in both names, then
    Move the last token in FNS to the last position of the long name (Figure. 1 C).

PIMF returns a similarity between 1 and threshold for a positive exact match. It also returns a similarity between the threshold and (threshold * threshold) for positive match with token adjustment; otherwise, it returns zero as a negative match.

5. EXPERIMENT

We were provided an excellent dataset by an organization that collects names from many countries. The organization gave us single names only and not full names for privacy and security reasons. The single names are sufficient for our experiment, and we can generate full names from those names. The dataset contains names and their spelling variations that were entered from different countries, such as the name "Mohammed" in Table 1.

Each instance in the dataset is the tuple of an Arabic name and its equivalent name in English. The total number of tuples in the dataset is that 29,821. Most of the names are Arabic names selected from 21 Arabic countries. These names are written in Arabic modern language, but when the names are transliterated into English, some of them are written based on the pronunciation of dialects. For example, the Arabic name "ذهب" is Romanized as DAHAB, DHAHAB, THAHAB, and THAHB. We checked the dataset manually and inserted a number (Cluster-Id) to each tuple, grouping all English names that referred either to the same Arabic name or to Arabic names that share several names in English. We used the "Cluster-Id" value as a ground truth for our test. In addition, the dataset contains a small fraction of noise (spelling errors) due to the actual data entry. We kept some of this noise to represent natural data entry errors.

We select 2000 names randomly from the dataset. Each of the random names is compared with all of the names in the dataset. Recall and precision are computed for all 2000 names. Then, we measure the F-Measure as our test's accuracy for 17 algorithms, including PIM. We repeat the test five times with 2000 different random names each time. Each algorithm is tested with a threshold between 0 and 1, and then the best average is recorded.

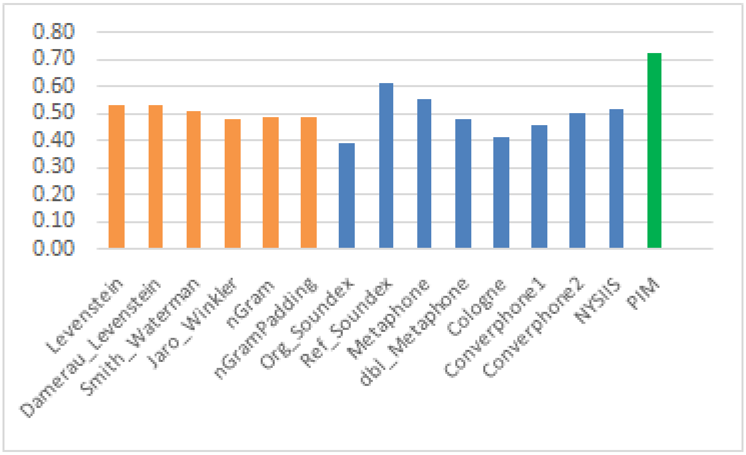

Figure 2. F-Measure for all algorithms.

Figure 2 shows the F-Measure for all of the algorithms, including PIM. The results are clustered by colors in Figure 2: Orange represents the pattern match algorithm, and blue represents phonetic algorithms. PIM records the best result with the F-Measure of 0.72, followed by Refined-Soundex, Metaphone, and Levenstein (Edit distance) with F-Measures of 0.61, 0.55, and 0.53, respectively. Soundex has the lowest score 0.39. PIM achieves 11% better accuracy than Refined-Soundex and 33% better accuracy than Soundex. The accuracy of 72% for the dataset that contains names with very similar pronunciation including some noise is considered good performance compared with the17 other algorithms.

Our aim is to identify a person using his/her name. Of course, a person cannot be identified by using a single name, such as a given name or family name only. The full name of a person is needed to be more precise in identification. Whenever a full name is combined with more names, it increases the ability to accurately distinguish that individual. All of the algorithms used in the first test do not take full name structure into consideration, such as “first-name last-name” or “last-name first-name.” Therefore, we use our PIMF algorithm and the Monge-Elkan algorithm. Both these algorithms use other algorithms that calculate the distance between two names, to compute full name similarity. We use PIM to calculate the similarity between two single names for PIMF and Monger-Elkan because PIM has the best result of all comparable algorithms.

Table 4. Results of the First Experiment

Algorithms	Recall	Precision	F-Measure
Caverphone	0.79	0.33	0.46
Caverphone 2	0.85	0.35	0.50
DamerauLevenstein	0.47	0.62	0.53
Double-Metaphone	0.89	0.34	0.48
Jaro-Winkler	0.40	0.60	0.48
Levenstein	0.46	0.61	0.53
Metaphone	0.73	0.44	0.55
Ngram	0.45	0.54	0.49
NGram-Padding	0.45	0.54	0.49
NYSIIS	0.60	0.46	0.52
PIM	0.73	0.72	**0.72**
Ref-Soundex	0.77	0.51	0.61
SmithWaterman	0. 49	0. 53	0.51
Soundex	0.91	0.25	0.39

We generate a dataset containing full names, each of which was composed of two names (tokens) and four names (tokens). Our assumption is that each full name has the right structure, as was explained earlier. So the first and second tokens can be either a given name or a last name. This dataset contains about 74,866 full names, 25% of which are composed of already existing tokens in different order. The revised order might contravene the full name structure; for example, the name “Ali Nasser Mohammed” cannot match “Nasser Ali Mohammed” but matches “Mohammed Ali”, “Ali Mohamed”, “Ali Nasser”, and “Mohammed Ali Nasser.” This dataset is used to test the performance of the PIMF and Monger-Elkan algorithms.

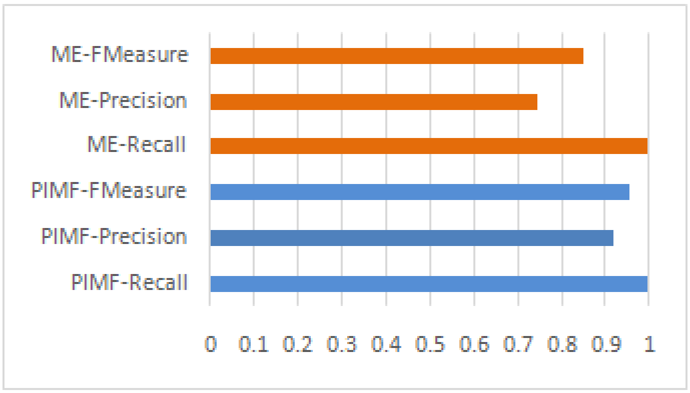

Figure 3. Recall, Precision, and F-Measure for PIMF and Monge-Elkan (ME)

We select 2000 instances randomly from the new dataset, and then the algorithms compare each instance with the entire dataset of 74,866 full names. The F-measure is calculated to check the algorithm performance. This process is repeated five times, and the final results are averaged. Figure 3 displays the recall, precision, and F-Measure for both the PIMF and Monger-Elkan (ME) algorithms. Both algorithms record very high recall, with 0.997 and 0.995 for PIMF and Monger-Elkan, respectively. PIMF has better accuracy than Monger-Elkan with an F-measure of 0.96 versus 0.85. As a result, PIM shows good accuracy at identifying individuals.

Table 5. Results of the Third Experiment

Lang. Match	**Algorithm**	**F-Measure**	**Recall**	**Precision**
English-Arabic	ASC	0.18	0.23	0.14
	PIM	0.50	0.67	0.40
Arabic-English	ASC	0.23	0.23	0.23
	PIM	0.53	0.61	0.48
Arabic-Arabic	ASoundex	0.78	0.86	0.71
	PIM	0.89	0.84	0.95

The third experiment is to evaluate name comparisons across languages. We implemented PIM to match the names that are written in Arabic with the English alphabet and vice versa. We use the algorithm created by [17,] and label it ASC. This algorithm is built to match personal names in English with names in Arabic script. Table 5 presents the results for PIM and ASC, which score 0.5 and 0.18, respectively, when comparing each English name to the list of Arabic names. When we reverse the test to compare each Arabic name to English names, we get only 3% better accuracy for both PIM and ASC, as shown Table 5. The last test compares names in Arabic to Arabic script. We compare ASoundex, which can be found in [15], with PIM. Table 5 shows the result of ASoundex and PIM for an Arabic-to-Arabic test and end up with F-Measures of 0.78 and 0.89, respectively. Most of the tests show good performance, especially for full names, which are our main targets.

5. CONCLUSION

There are many algorithms for name matching that try to solve the name variation challenges, included name transliteration. Also, cross-language information retrieval cannot deal with OOV words, 50% of which are personal names. Yousef [1] tried to resolve OOV proper names by adding names automatically into a dictionary, and the authors in [8] used probability to transliterate names “on the fly” from English to Arabic.

The experimental results indicate that our proposed algorithms (PIM and PIMF) perform better than most known name-matching algorithms to compare transliterated names and full names. Also, the PIM shows a good outcome when the Arabic language is used as an example to compare names across languages. This comparison was between names that were written in Arabic and English and vice versa. PIMF provides excellent accuracy of up to 96% for full name comparison.

This algorithm can contribute to improving performance in many fields. Security screening systems are one area that can benefit from this algorithm. A given name and a list of names (suspects’ names) can be given as input to this algorithm, which will then return a new list (sub-list) that contains all matched names with a degree of similarity to the given name. This helps to identify the closest personal names to a given name. In addition, our algorithms contain an excellent environment to accommodate other languages. Cross-language information retrieval can benefit from this algorithm because comparisons can be done "on the fly" for any language that can map its letters to Soundex categories (Table 2) that are converted into a vectors that are standard for all languages.

REFERENCES

[1] Maureen Cooney, "Report Assessing the Impact of the Automatic Selectee and No Fly Lists," 2006.

[2] Christopher Westphal, Data Mining for Intelligence, Fraud & Criminal Detection: Advanced Analytics & Information Sharing Technologies, 1st ed. CRC Press, Inc. Boca Raton, FL, USA ©2008, 2008.

[3] P. C. P. Christen, "A Comparison of Personal Name Matching: Techniques and Practical Issues," Sixth IEEE Int. Conf. Data Min. - Work., 2006.

[4] Tina Lieu: The Name Matching You Need: A Comparison of Name Matching Technologies, Cambridge, MA (2012).

[5] David Pinto, Darnes Vilariño Ayala, Yuridiana Alemán, Helena Gómez, Nahun Loya, "The Soundex Phonetic Algorithm Revisited for SMS Text Representation," in TSD 2012, 2012, vol. 7499, pp. 47–55.

[6] L. Karl Branting, "Efficient Name Variation Detection", AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection, Arlington, VA, October 13-15, 2006

[7] Ghita Amor-Tijani, "Enhanced english-arabic cross-language information retrieval," George Washington University Washington, DC, USA, 2008.

[8] Larkey, L., AbdulJaleel, N., Connell, M.: What's in a Name?: Proper Names in Arabic Cross Language Information Retrieval. Massachusetts (2003).

[9] K. Darwish, "Named Entity Recognition using Cross-lingual Resources : Arabic as an Example," in Acl, 2013, pp. 1558–1567.

[10] N. AbdulJaleel and L. S. Larkey, "Statistical transliteration for english-arabic cross language information retrieval," in Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03, 2003, p. 139.

[11] Lawrence Philips, "Hanging on the Metaphone", Computer Language, Vol. 7, No. 12 (December), 1990.

[12] W. E. Yancey, "Evaluating string comparator performance for record linkage," Tech. Rep. RR2005/05, US Bureau of the Census 2005.

[13] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg, "Adaptive name matching in information integration," IEEE Intell. Syst., vol. 18, no. 5, pp. 16–23, 2003.

[14] M. Sabbagh-nobarian, "The Review of Fields Similarity Estimation Methods," IJMLC Int. J. Mach. Learn. Comput., vol. 2, no. Icmlc, pp. 596–599, 2011.

[15] S. U. Aqeel, S. Beitzel, E. Jensen, D. Grossman, and O. Frieder, "On the development of name search techniques for arabic," J. Am. Soc. Inf. Sci. Technol., vol. 57, no. 6, pp. 728–739, 2006.

[16] A. Lait and B. Randell, "An assessment of name matching algorithms," Newcastle, UK, 1996.

[17] Freeman, S. Condon, and C. Ackerman, "Cross linguistic name matching in English and Arabic: a one to many mapping extension of the Levenshtein edit distance algorithm," … Comput. Linguist., June, pp. 471–478, 2006

AUTHORS

Mazin H. Al-Shuaili is currently a Ph.D. candidate at Florida Institute of Technology (FIT), in Melbourne, FL/USA. In 1998, he graduated in computer science from FIT. In 2000, he obtained his master's degree in software engineering from FIT. His master had focused on software test automation. From 2000 till 2012, he was a system analyst and project manager at Omani government. In May 2016, he is going to graduate and he going back to work with Omani government, Muscat. His research interests are in general areas of Natural language processing (NLP), social network, and data mining.

Marco M. Carvalho is an Associated Professor at the Florida Institute of Technology, in Melbourne, FL/USA. He graduated in Mechanical Engineering at the University Brasilia (UnB – Brazil), where he also completed his M.Sc. in Mechanical Engineering with specialization in dynamic systems. Marco Carvalho also holds a M.Sc. in Computer Science from the University of West Florida and a Ph.D. in Computer Science from Tulane University, with specialization in Machine Learning and Data Mining. At Florida Tech, Dr. Carvalho is the Executive Director of the Harris Institute for Assured Information, and the Principal Investigator of several research projects in the areas of cyber security, information management, networks, and tactical communication systems.