Applying Optimized Algorithms and Technology for Interconnecting Big Data Resources in Government Institutions

https://doi.org/10.3991/ijoe.v19i08.39661

Genc Hamzaj[1], Artan Mazrekaj[2(✉)], Isak Shabani[2]
[1]SouthEast European University, Tetovo, North Macedonia
[2]University of Pristina, Prishtina, Kosovo
artan.mazrekaj@uni-pr.edu

Abstract—The quality of the data in core electronic registers has constantly decreased as a result of numerous errors that were made and inconsistencies in the data in these databases due to the growing number of databases created with the intention of providing electronic services for public administration and the lack of the data harmonization or interoperability between these databases. Evaluating and improving the quality of data by matching and linking records from multiple data sources becomes exceedingly difficult due to the incredibly large volume of data in these numerous data sources with different data architectures and no unique field to create interconnection among them. Different algorithms are developed to treat these issues and our focus will be on algorithms that handle large amounts of data, such as Levenshtein distance (LV) algorithm and Damerau-Levenshtein distance (DL) algorithm. In order to analyze and evaluate the effectiveness and quality of data using the mentioned algorithms and making improvements to these algorithms, through this paper we will conduct experiments on large data sets with more than one million records.

Keywords—data quality assessment, Levenshtein distance (LV) algorithm, data quality improvement

1 Introduction

High data quality has become a crucial component of data management within a business institution or organization. Since the beginning of the twenty-first century, there have been numerous notable technological advancements in the information technology sector, including cloud computing, the Internet of Things, IoT technologies-based Healthcare [12], and social networking. The advancement of these technologies has caused the rise of the volume of data in an exponential way [1]. In order to provide electronic services for public administration, such is online recruitment, a huge number of databases were created [11]. However, because these databases were not connected or their data was not standardized, this resulted in a high number of mistakes and inaccuracies, which decreased the quality of the data.

Since the electronic services provided directly depend on the quality of the data that is available, analyzing and improving the quality of data contained in information systems is an important and difficult process for e-government entities. When we try to offer electronic services, since data are saved in different data sources, it is a necessity to interconnect datasets from these data sources using appropriate algorithms specifically when interconnection is obliged to be done without existing unique field for interconnection. There are numerous current algorithms for data matching and connecting records between various data sources so it is very important to decide which algorithm is better to use for assessing quality and performance of data during treatment of the datasets. We will concentrate on advanced algorithms that handle enormous amounts of data, such as the Levenshtein distance (LV) algorithm and the Damerau-Levenshtein distance (DL) algorithm.

2 Related work

Ensuring the highest quality data is achieved through continuous actions of measurement, analysis and improvement of data quality. In general, DQ assessment includes of numerous phases that an organization, users, and developers must perform, shown in Figure 1 [2]:

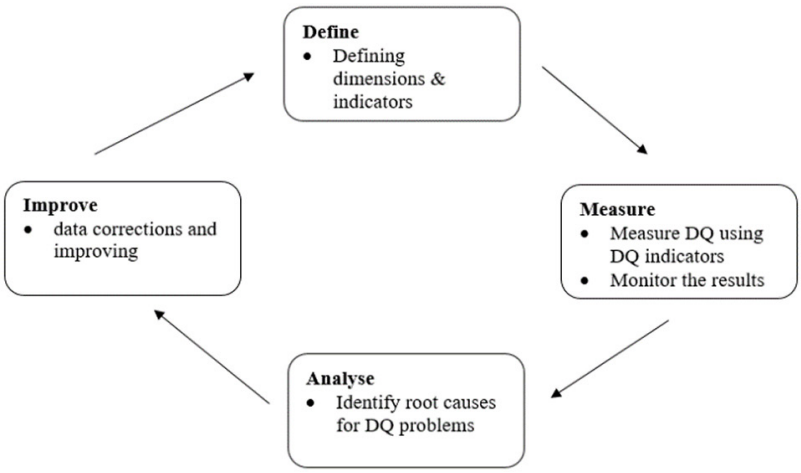

Fig. 1. Process to insure high quality data [2]

When you have data and datasets located in different sources with different structure of the data, the main challenge is to integrate and relate these data sources without having any unique field with the aim to offer better e-services.

2.1 Algorithms for matching and linking records from multiple resources

In order to perform data matching and linking from different sources, different existing algorithms can be used, which greatly facilitate this process. The data collected

from different sources often do not have good quality, so the intention is to improve this data with the main aim of providing better e-services.

Some of these algorithms are:

- Levenshtein distance algorithm
- Damerau Levenshtein distance algorithm
- Optimal String Alignment
- Q – gram distance
- Longest Common Substring
- Jaccard distance Cosine distance.

The Levenshtein distance is an algorithm used to measure the difference between two given sequences. Informally, the Levenshtein distance is the minimum number of operations or modifications (e.g. Insertion, Deletion or Substitution) required for a single-character of the first word until it will be the same as the second word [3].

According to Nikhil Babar, through the Levenshtein algorithm it is possible to determine the least amount of operations required to change one string and turn this string into another. It can be calculated effectively using below approach [3]:

- In order to initialize a matrix, the (m, n) cell's distance between a word's m- and n-character prefixes must be determined.
- The upper left to bottom right corners of the matrix can be filled in.
- An insert or a deletion is represented by each hop, whether it is horizontal or vertical.
- Normally, the cost for each operation is set to 1.
- If both characters in the row and column match, it will be either one or zero. Every cell always attempts to reduce local costs.
- In this situation, the Levenshtein distance between the two words is represented by the number in the lower right corner.

According to Rishin Haldar and Debajyoti Mukhopadhyay, following steps must be taken by the Algorithm 1 [4]:

Algorithm 1: The Levenshtein Distance Algorithm
Step 1: Initialization phase
1. a) Set n to be the length of s, set m to be the length of t
2. b) Construct a matrix containing 0..m rows and 0..n columns
3. c) Initialize the first row to 0..n
4. d) Initialize the first column to 0..m
Step 2: Processing phase
5. a) Examine s (i from 1 to n)
6. b) Examine t (j from 1 to m)
7. c) If s[i] equals t[j], the cost is 0
8. d) If s[i] doesn't equal t[j], the cost is 1
9. e) Set cell d[i,j] of the matrix equal to the minimum of:
10. i) The cell immediately above plus 1: d[i-1,j] + 1
11. ii) The cell immediately to the left plus 1: d[i,j-1] + 1
12. iii) The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost
Step 3: Result
Step 2 is repeated till the d[n,m] value is found

The Damerau Levenshtein distance represents a variant of another form of the Levenshtein distance, where it pertains to algorithms of the Edit type. As was previously mentioned, the "edit-distance" category determines how distinct two strings are by turning one string to the other and calculating the operations needed. The Damerau-Levenshtein distance, compared to the classic Levenshtein distance, during the character edit, in addition to operations such as Insert, Delete and Substitution, also uses the transposition operation [5].

Wagner and Fischer [6] created a trace notion as a function of cost in several structures, in order to simplify the process of finding the distance between the first string and the second string.

This trace can be illustrated as a diagram as in the Figure 2.

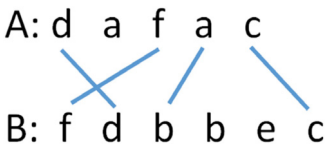

Fig. 2. DL trace example [6]

In Algorithm 2 it is presented the pseudocode of the Damerau Levenshtein algorithm, where the H value is calculated, whereas last_row_id[c] represents the last trace of character c in A and last_col_id represents the last trace of ai in B [7].

Algorithm 2: Damerau Levenshtein Distance Algorithm

```
1. DL(A[1:m], B[1:n])
2. for j ← 0 to n do
3.  H[-1][j] ← maxVal; H[0][j] ← j
4. end for
5. for i ← 1 to m do
6.  H[i][-1] ← maxVal; H[i][0] ← i
7.last_col_id ← -1
8.    for j ← 1 to n do
9.     diag ← H[i-1] [j-1] + c(A[i],B[j])
10.     left ← H[i] [j-1] +1
11.     up ← H[i-1] [j] +1
12.     k = last_row_id[B[j]], l = last_col_id
13.     transpose ← H[k-1] [l-1] + (i - k - 1) +1 + (j - l -1)
14.     H[i][j] ← min{diag, left, up, transpose}
15.     if A[i] = B[j] then
16.        last_col_id ← j
17.     end if
18.   end for
19. last_row_id[A[i]] ← i
20. end for
21.     return H[m][n]
```

3 Improvements of algorithms for matching and linking records from multiple resources

Many researchers used different ways and methods with the aim of improving algorithms for matching and linking records from multiple resources.

According to H.N. Abdulkhudhur & I.Q. Habeeb, Levenshtein's algorithm is the most used algorithm for finding words that are most similar to the incorrect word based on a certain lexicon. By sequentially contrasting the letters of the incorrect word with the characters of the correct word from a lexicon, it calculates a sequence of operations that fill the cells of an array. Such actions will be done millions of times for each wrong word to create the list of viable options. In order to reduce this large number of operations created due to the comparison of the characters of the incorrect word and the lexicon words, the authors propose an improved Levenshtein algorithm. Compared to Levenshtein's algorithm, the proposed so-called ILA-OT algorithm, based on experimental results, has a reduction in processing time of 32.43% [8].

From Figure 3, it can be seen that in terms of processing time, the proposed ILA-OT algorithm is faster than the LA algorithm in percentage by about 32.43%, while not changing the number of comparisons between both algorithms with the total number of the comparisons as shown in Figure 4, with 100% accuracy [8].

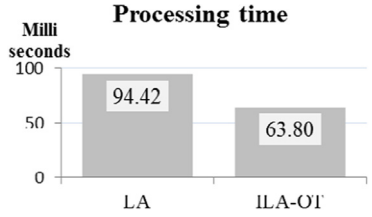

Fig. 3. Processing time LA and ILA-OT [8]

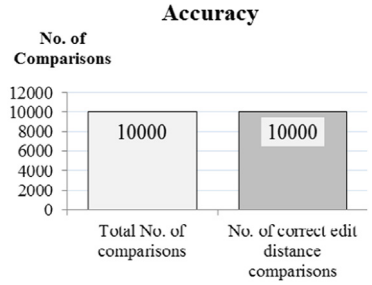

Fig. 4. Accuracy between LA and ILA-OT [8]

According to Z. ZHAO & Zh. YIN, extending the transposition procedure in the current method reduces the number of edit operations. Improving the algorithm in such a way that by enabling transposing isolated symbols even after the calculation position and not only before the calculation position, then as a result a better edit distance can be obtained [9].

According to Shama Rani & Jaiteg Singh, by removing stop words such as "also, is, am, are, they, them, their, was, were" etc., Levenshtein's edit distance algorithm can be modified and improved [10].

The following conditions lead to the removal of stop words [10]:

- Each manuscript has around 20–25% stop words
- Eliminating stop words increases the effectiveness of the document
- Text mining and searches do not benefit from stop words
- Aim is to reduce indexing.

Levenshtein's Edit distance algorithm is utilized to calculate the inputs and the amount of words in all the manuscripts, as shown in Table 1. The time taken to calculate Levenshtein's distance with Stop words is shown in Table 2. The time taken to compare documents is calculated in milliseconds [10].

Table 1. Text length of Document A and B with and without using stop words [10]

Text Length of Document A	Text Length of Document B	Document A after Removing Stop Words	Document B after Removing Stop Words
51	62	27	38
103	90	59	53
203	192	124	119
395	410	242	233
798	750	470	474

Table 2. Length of time needed to determine LV distance once stop words are removed [10]

Text Length of Document A	Text Length of Document B	Time Taken to Calculate Levenshtein's Distance Withstop Words (In Milliseconds)
51	62	14
103	90	16
203	192	23
395	410	62
798	750	218

Figure 5 compares the lengths of Document A and the identical document after the stop words were deleted. Additionally, Figure 5 shows the text length with and without stop-words [10].

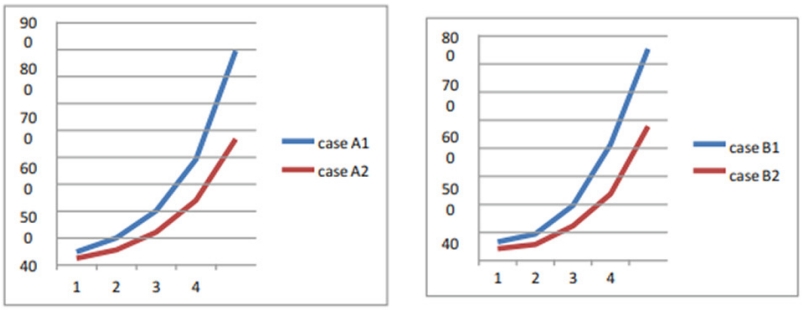

Fig. 5. Document A and B by using or avoiding stop-words [10]

Figure 6 shows the time taken to calculate LA with stop-words (case TW1) and the time needed to compute edit distance following the removal of stop words (case TW0) [10].

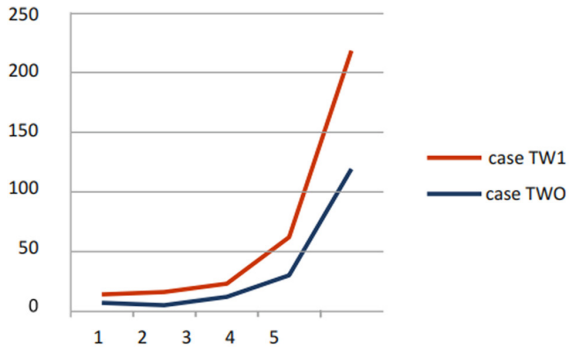

Fig. 6. Time spent on calculations both before and after stop words were removed [10]

According to R. Haldar and D. Mukhopadhyay, in cases where the letters are not recognized by Optical Character Readers, dictionary lookup methods are mostly used. However, these methods increase the cost of searching due to the complexity in the calculation, so the Levenshtein distance is an effective algorithm with the aim of string approximation [4].

As is known, the Levenshtein Distance Algorithm, for any operation (Insert, Delete, or Substitute) gives the uniform distance value (ie, 1) when comparing two different characters. An improvement of this method by grouping characters with similar appearance and calculating the difference of the characters of this group with a value smaller than the value 1, as a result would enable closest matches to be more accurate. For example, during the use of any operation for the characters O, D, Q, the weight may be given with a value of 0.4 and not 1 as for the other characters. As a result, we will have an improved version compared to the initial version of Levenshtein's algorithm [4].

The groups identified are [4]:

1. O, D, Q
2. I, J, L, T
3. U, V
4. F, P
5. C, G

For example, it may happen that OCR has mistakenly read the word BODY as BDQY due to the similarity of the characters O and D. Figure 7 explains this case quite clearly. From the word BDQY, according to Levenshtein's algorithm, all other words (BODY, BUSY, BURY, BONY) have a distance of two. However, if we use the improved version of Levenshtein Distance Algorithm, it turns out that the distance between BDQY and BODY has the shortest distance compared to the other words, because D, Q is in the same group as O, D, so the word BODY is chosen as the answer [4].

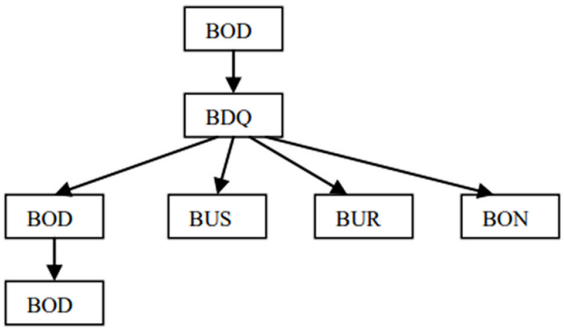

Fig. 7. An example of possible outcomes [4]

4 Experimental results

Attributes or fields that are used while applying the algorithm for matching and linking of personal data are: First Name, Last Name, DOB, Birthplace, Father's First Name, Father's Last Name, Mother's First Name, Mother's Last Name.

To execute algorithms it was needed to create a high performance hardware infrastructure. Testing infrastructure properties and volumes of data that are compared are shown in Table 3:

Table 3. Hardware infrastructure – testing environment

No.	Testing Infrastructure – Hardware	
1	RAM Memory	128 GB
2	HDD	14 TB
3	Processor	Intel® Xeon® Platinum 8164 2.00 GHz (16 CPUs)
	Datasets Volume	
1	Dataset 1	2.5 Million Rows
2	Dataset 2	1.85 Million Rows

4.1 Improving algorithms for matching and linking of personal records by comparing similar letters in Albanian alphabet

During the process of analyzing the data that has been compared for the purpose of matching the data from different datasets, we noticed that in the Albanian language there are some letters that are similar or that are often used when writing names, surnames, birthplaces and other important fields when filling in the citizens' data. For example, it is often wrong when writing the name Qerim when this name is written as Çerim. Both of these letters in the Albanian language have the same pronunciation but are used in specific cases.

Through the improvement of Levenstein's algorithm, we managed to reduce the distance from 1 to 0.6, for such errors and other errors listed in the table below, while the distance for other letters that are not in the Table 4 the distance is 1.

Table 4. Similar letters in Albanian alphabet

Number	First Comparative Letter	Second Comparative Letter
1	"e"	"ë"
2	"ë"	"e"
3	"i"	"j", "y"
4	"j"	"i","y"
5	"y"	"i","j"
6	"q"	"ç"
7	"ç"	"q"

The Algorithm 3 is used to generate results for improving the Levenshtein Distance for similar letters in the Albanian alphabet is shown below.

Algorithm 3: Similar letters in the Albanian alphabet

```
1. USE[DBIMPROVED]
2. GO
3. SET ANSI_NULLS ON
4. GO
5. SET QUOTED_IDENTIFIER ON
6. GO
7. Create FUNCTION [dbo].[neighbors](@s1 nvarchar(100), @s2 nvarchar(100))
8. RETURNS bit
9. AS
10.   BEGIN
11.   declare @c bit
12.   set @c=case when @s1 in ('e') and @s2 in ('ë') then 1 else
13.   case when @s1 in ('ë') and @s2 in ('e') then 1 else
14.   case when @s1 in ('i') and @s2 in ('j','y') then 1 else
15.   case when @s1 in ('j') and @s2 in ('i','y') then 1 else
16.   case when @s1 in ('y') and @s2 in ('i','j') then 1 else
17.   case when @s1 in ('q') and @s2 in ('ç') then 1 else
18.   case when @s1 in ('ç') and @s2 in ('q') then 1 else
19.   0
20.   end end end end end end end
21.   RETURN @c
22.   END
```

Table 5 displays the outcome of putting the stated changes into practice for the matching higher than 50%.

Table 5. Results for range more than 50% after implementing improvements for similar letters in Albanian alphabet

Range of Matching in %	Number of Matching	Total Percentage
100%	32,929	1.26%
90%–99.99%	622,458	23.92%
80%–89.99%	599,954	23.03%
70%–79.99%	334,478	12.84%
60%–69.99%	379,127	14.57%
50%–59.99%	645,663	24.80%
Total	**2,614,609**	**100%**

4.2 Improving algorithms for matching and linking of personal records by specifying importance of each field

Based on the needs from researchers many new features in algorithms and functions are added with the aim that the result will be more accurate.

When we implemented the Levenshtein algorithm in our personal records datasets, we noticed that all the fields have the same % of weight or importance. One example is shown in Table 6:

Table 6. Personal records with same weight or importance for all columns

First Name	Last Name	DOB	Birth Place	F. First Name	F. Last Name	M. First Name	M. Last Name	Matching %
Azem	Maxhuni	15.12.1985	Gjilan	Hasan	Maxhuni	Have	Maxhuni	
14.28%	14.28%		14.28%	14.28%	14.28%	14.28%	14.28%	
Adem	Magjuni	15.12.1985	Gjilan	Hasan	Maxhuni	Have	Maxhuni	
10.71%	10.20%		14.28%	14.28%	14.28%	14.28%	14.28%	**92.31%**

As we can see in this example, the percentage of mistakes is the same no matter if the mistake is in the fiend First Name or in the field Mathers's First Name because all the fields have 14.28 % in total percentage.

The outcome of the process is displayed in Table 7 after the following stage of removing all duplicate data for the percentage matching higher than 50% (each field have same importance or weight).

Table 7. After removing duplicate data, results for a range greater than 50%

Range of Matching in %	Number of Matching	Total %
100%	32,929	1.26%
90%–99.99%	528,043	20.20%
80%–89.99%	539,429	20.63%
70%–79.99%	270,601	10.35%
60%–69.99%	296,158	11.32%
50%–59.99%	947,449	36.24%
Total	**2,614,609**	**100%**

For the datasets that we compared, some fields are more important that the other to show the accuracy of the records. For the same record compared, we can notice that when we add feature Weight or Importance for every column, we have different results in percentage.

One example is shown in Table 8 below:

Table 8. Personal records with different weight for specific columns

First Name	Last Name	DOB	Birth Place	F. First Name	F. Last Name	M. First Name	M. Last Name	Matching %
Azem	Maxhuni	15.12.1985	Gjilan	Hasan	Maxhuni	Have	Maxhuni	
20%	20%		14%	11%	12%	11%	12%	
Adem	Magjuni	15.12.1985	Gjilan	Hasan	Maxhuni	Have	Maxhuni	
15%	14%		14%	11%	12%	11%	12%	89.28%

The outcome is presented in Table 9 after the following stage of removing all duplicate values for the percentage matching higher than 50% (each fiend have specific importance or weight).

Table 9. After removing duplicate data, results for a range greater than 50%

Range of Matching in %	Number of Matching	Total %
100%	32,929	1.26%
90%–99.99%	592,124	22.71%
80%–89.99%	580,289	22.24%
70%–79.99%	291,357	11.15%
60%–69.99%	346,159	13.19%
50%–59.99%	771,751	29.45%
Total	**2,614,609**	**100%**

The difference between two approaches is illustrated in the Figure 8 for same set of data.

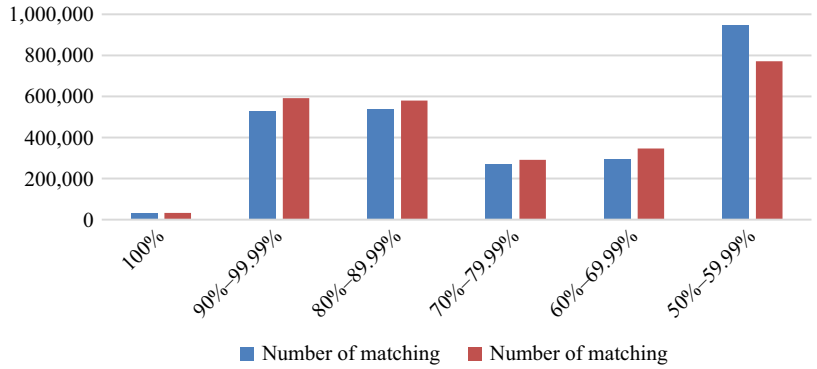

Fig. 8. Compared results before and after specifying weight or importance of each field

4.3 Improving algorithms for matching and linking of personal records by specifying distance of edit operations

By applying Levenshtein's approach, the distance between two documents is estimated as the least amount of changes that need to be done in order to create one document from another document.

The editing techniques used by this algorithm are listed below:

- Insert
- Delete
- Substitute.

By implementing all three described operations into practice, it is possible to create a document from another document by changing, removing, or adding a certain number of characters. During the implementation of this algorithm, we improved the definition of the importance (distance) of editing operations, where we assigned the distance 0.6 to the substitution operation, 1.2 to the Insert operation, and 1.2 to the Delete operation.

The Algorithm 4 is used to generate results for improved Levenshtein Distance Algorithm for edit operations is shown below.

Algorithm 4: Improving Levenshtein Distance Algorithm for edit operations

```
1. BEGIN
2. if (len(@s2) = 0)
3.   begin
4.   return @s2
5. END
6. DECLARE @s1_len int, @s2_len int, @i int, @j int, @s1_char nchar, @c int, @c_temp float, @n bit,
   @nk int, @k int,@cv0 varbinary(8000), @cv1 varbinary(8000)
7. SELECT @s1_len = LEN(@s1), @s2_len = LEN(@s2), @cv1 = 0x0000, @j = 1, @i = 1, @c = 0,
   @nk = 0, @k = 1
8. WHILE @j <= @s2_len
9. SELECT @cv1 = @cv1 + CAST(@j AS binary(2)), @j = @j + 1
10.   WHILE @i <= @s1_len
11.  BEGIN
12.  SELECT @s1_char = SUBSTRING(@s1, @i, 1), @c = @i, @cv0 = CAST(@i AS binary(2)), @j = 1
13.  WHILE @j <= @s2_len
14.  BEGIN
15.  SET @c = @c + 1
16.  SET @c_temp = (CAST(SUBSTRING(@cv1, @j+@j-1, 2) AS int) + CASE WHEN @s1_char =
   SUBSTRING(@s2, @j, 1) THEN 0 ELSE 1 END)
17.  IF @c > @c_temp SET @c = @c_temp
18.  SET @c_temp = CAST(SUBSTRING(@cv1, @j+@j+1, 2) AS int)+1
19.  IF @c > @c_temp SET @c = @c_temp
20.  SELECT @cv0 = @cv0 + CAST(@c AS binary(2)), @j = @j + 1
21.  END
22.  SELECT @cv1 = @cv0, @i = @i + 1
23.  declare @ins int =0, @del int = 0
24.  if @s1_len > @s2_len
25.  BEGIN
26.  SET @del = @s1_len - @s2_len
27.  END
28.  else
29.  BEGIN
30.  SET @ins = @s2_len - @s1_len
31.  END
32.  END
33.  RETURN (@c * 0.6) + (@ins * 1.2) + (@del *1.2) - (@ins * 0.6) - (@del * 0.6)
34.  END
```

By implementing an improved Levenshtein algorithm when distance in edit operations is specified, we reduce the time when comparing data from different data sets and we come to better results in a faster way.

5 Conclusions and future work

After assessing the datasets regarding data quality, we applied a variety of data matching algorithms to connect personal records from different data sources. The conclusion is that the Levenshtein Distance algorithm is the best algorithm for the matching and linking process when the focus in on quality of the data and in performance.

Findings in this paper emphasize the significance of assessing DQ, and highlights the importance of identifying algorithm that suites the most for linking and matching process. In the context of using appropriate algorithm for matching and linking data from multiple resources, many algorithms are used and also many improvements are done in these algorithms with the aim to have better and adequate results. We also added new features to algorithms that treat data from multiple resources with the aim to improve quality of data in order to reduce the time needed to obtain the required result.

There are still challenges in using and improving algorithms such as the Levenshtein distance algorithm and Damerau Levenshtein distance algorithm that comes to the result with fewer steps and less time to have qualitative data as output. The search space between two datasets is attempted to be smaller by selecting the appropriate blocking variable.

6 References

[1] Cai, L., & Zhu,Y. (2015). The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. Data Science Journal, 14 (2), pp. 1–10. https://doi.org/10.5334/dsj-2015-002

[2] Aljumaili, M. (2016). Data Quality Assessment: Applied in Maintenance. PhD thesis, Lulea University of Technology.

[3] Babar, N. (2022). The Levenshtein Distance Algorithm. Online: https://dzone.com/articles/the-levenshtein-algorithm-1 [Accessed: July 2022].

[4] Haldar, R., & Mukhopadhyay, D. (2011). Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. Computing Research Repository-CORR, Web Intelligence & Distributed Computing Research Lab.

[5] OpenGenus IQ (2022). Damerau Levenshtein Distance. Online: https://iq.opengenus.org/damerau-levenshtein-distance [Accessed: July 2022].

[6] Wagner, R., & Fisher, M. (1974). The String to String Correction Problem. Journal of ACM, 21 (1), pp. 168–173. https://doi.org/10.1145/321796.321811

[7] Zhao, C., & Sahni, S. (2019). String Correction using the Damerau-Levenshtein Distance. BMC Bioinformatics, 20 (11), pp. 20–103. https://doi.org/10.1186/s12859-019-2819-0

[8] Abdulkhudhur, H. N., Habeeb, I. Q., Yusof, Y., & Yusof, Sh.A.M. (2016). Implementation of Improved Levenshtein Algorithm for Spelling Correction Word Candidate List Generation. Journal of Theoretical and Applied Information Technology, 88 (3), pp. 449–455.

[9] Zhao, Z. P., Yin, Z. M., Wang, Q. P., Xu, X. Z., & Jiang, H. F. (2009). An Improved Algorithm of Levenshtein Distance and its Application in Data Processing. Journal of Computer Applications, 29 (2), pp. 424–426. https://doi.org/10.3724/SP.J.1087.2009.00424

[10] Rani, S., & Singh, J. (2017). Enhancing Levenshtein's Edit Distance Algorithm for Evaluating Document Similarity. International Conference on Computing, Analytics and Networks, pp. 72–80. https://doi.org/10.1007/978-981-13-0755-3_6

[11] Yong, W., Yinle, Zh., & Ning, L. (2023). Big Data Analysis and Forecast of Employment Position Requirements for College Students. International Journal of Emerging Technologies in Learning, 18 (4), pp. 202–218. https://doi.org/10.3991/ijet.v18i04.38245

[12] Hamza, R., Abderrahim, M., & Abdelaziz, E. (2023). Data Security Mechanisms, Approaches, and Challenges for e-Health Smart Systems. International Journal of Online and Biomedical Engineering, 19 (2), pp. 42–66. https://doi.org/10.3991/ijoe.v19i02.37069

7 Authors

Genc Hamzaj has earned PhD degree in e-Technology from Southeast European University in the North Macedonia. He has published numerous papers in conferences and journals. His research interests includes Data Science, Databases, Optimized Algorithms and Big Data (e-mail: gh26149@seeu.edu.mk).

Artan Mazrekaj is an Assistant Professor at the Faculty of Electrical and Computer Engineering, University of Prishtina "Hasan Prishtina", Prishtina, Kosovo. He obtained PhD from Southeast European University in the North Macedonia, in the e-Technology program. He has published numerous papers in conferences and journals. His research interests includes Cloud Computing, Distributed Systems and IoT (e-mail: artan.mazrekaj@uni-pr.edu).

Isak Shabani is a Full Professor and Dean at the Faculty of Electrical and Computer Engineering, University of Pristina "Hasan Pristina", Prishtina, Kosovo. He has obtained a PhD degree in computer science and engineering from the University of Prishtina. He is currently the author of several international journals and conferences, and has been engaged as a reviewer in several scientific papers of his narrow filed of research. Also, he is engaged in several scientific projects. His research interests includes Distributed systems, Web services, HCI, Systems design and ICT in Education (e-mail: isak.shabani@uni-pr.edu).

Article submitted 2023-03-12. Resubmitted 2023-04-14. Final acceptance 2023-04-15. Final version published as submitted by the authors.