# Address Management in The Logistics Industry: Address Singularization with Text Similarity Algorithms

**Ahmet Ustaoğlu, Hasan Güney, Ahmet Yesevi Türker, Tuğçe Elçi, Deniz Kantar**

Borusan Logistics, 34700 İstanbul, Türkiye

## Abstract

In logistics and supply chain management, addresses are the basis of management systems that provide move and delivery from somewhere to somewhere. The logistics sector purposes to have a simple, straightforward, explanatory, and truthful address pool. The increase in logistics operations continuously leads to the creation of new addresses. However, route optimization can be negatively affected when too many addresses refer to the same location. Also, incorrect addresses may cause considerable costs due to the re-delivery of the shipments. This research proposes eliminating duplicated and unclear addresses using an address management algorithm developed by the Borusan Logistics Artificial Intelligence team. This approach includes the singularization of addresses and the control of addresses. The singularization part consists of the deduplication of addresses and the detection of unclear addresses. In this part, the Grid Search algorithm is developed to find duplicated and unclear addresses. Then, the control part of addresses includes the most beneficial text similarity algorithm to avoid pollution of the address pool. In this part, the Jaro-Winkler Distance, Hamming Distance, Levenshtein Distance, Fuzzy String Matching, String Score, and Sequences Matching algorithms are calculated and considered. Then, to finalize the smart decision algorithm, in terms of their performance, String Score, Jaro-Winkler Score, and 2 of Fuzzy Score are gathered and used to calculate the final similarity score of each address. As a result, an end-to-end address management system was built to deduplicate and control addresses, ensuring a controlled address pool.

**Keywords:** Address Similarity, Data Deduplication, Address Management, Grid Search, String Similarity Algorithms

## 1. Introduction

The logistics sector has been globalized with each passing day. With technological improvements in every field, digitalization has become a more popular topic in logistics. Therefore, new topics such as optimization, machine learning, and deep learning algorithms are vital in logistics and supply chain management (Shavaki & Ghahnavieh, 2022). To use these algorithms in logistics processes, such as optimization, needs precise data, especially for addresses. Since e-commerce, urbanization, and global trade volume have rapidly increased, the demand for logistics has grown. Thus, this situation increased the number of addresses in address management systems.

The existence of different address texts in the system, despite pointing to the same place, may create a challenge in terms of the accuracy and reliability of addresses. In some cases, errors can occur due to data inaccuracies or data obtained from multiple integrations (Koumarelas et al., 2018). Also, some address texts may contain different structures or deficient information, such as the absence of specific street names and the omission of region identifiers. Therefore, there might be hundreds of variations of an address for just one location. These variations of the address pollute the address pool. An uncertain address pool is a significant challenge to move forward in this sector. These address complexity, which causes difficulty in supply chain management and inefficiency in technical solutions such as optimization. As a result, this problem brings on excessive amounts of addresses and redundant usage of storage, fuel oil, employees, and time. In relation to that, extra cost and extra carbon emission are shown. Methods such as conditional random fields and word2vec, utilized in a text-based address-matching task, have been evaluated in the literature. (Comber & Arribas-Bel, 2019). Random forest classification was also used in similar studies, and success was measured with the f1 score at the end of the modeling (Koumarelas et al., 2018). The characteristics of address data in Turkey show differences compared to countries such as Germany, the UK, and the USA. Unlike these countries, a single postal code is often assigned to larger areas in Turkey. Consequently, postal codes are generally not used for locating addresses as they do not provide significant assistance. Instead, a combination of neighborhood (mahalle), district (ilçe), and street is commonly used for address identification. Additionally, in Turkey, local governments frequently change road names, and there are instances of multiple streets having the same name (Yildirim et al., 2014). Moreover, changes in the administrative assignment of streets and avenues to districts are common. These factors contribute to the absence of a comprehensive and unique dataset for accurately comparing address data in Turkey. This scarcity impacts the applicability of the methods discussed in (Koumarelas et al., 2018) for Turkey, as these methods often rely on comprehensive and accurate datasets, which are not readily available for Turkey, unlike the USA. Since existing studies in the literature did not provide a solution to the specific problem, a new method has been attempted instead. In this paper, with coming parts, a

technical solution will be proposed to solve or relieve this problem. However, to make a brief introduction, it can be said that this technical solution consists of 2 crucial steps.

The first one is cleaning the address pool by deduplicating and singularizing the addresses after getting rid of the wrong and deficient ones. In this step, address texts are first filtered in terms of their cities and counties. Then, addresses are gathered after their string similarities are considered. The string similarity is considered by looking at the similarity of address text and address accounts and their string patterns for each address found in the pool. The comparison and evaluation of addresses were made by using string similarity algorithms such as Levenshtein, FuzzyWuzzy, Jaro-Winkler, and Hamming Distance scores. With some of these algorithms, which give high performance, a final smart string similarity algorithm is created and used for making decisions. After collecting the same address together, one of them is assigned as the main address and labeled as 1. Then, the others are linked and referred to this main address and labeled as 0. With these steps mentioned above, the pool becomes clean. Now, the pool has fewer addresses but more efficient and accurate ones.

The next and final step is to prevent the address pool from getting dirty again. The main focus in this step is that when a new address comes to the system or server or address pool, at first, it is considered with the addresses surviving in the address pool after the first step is executed. The newly coming address is checked with cleaned addresses, and if there is a high similarity between the address texts, the second phase offers a link to a similar address found in the pool. With this method, the new address is linked to the reference address offered with high similarity, and as a result, the pool is kept clean. These two crucial steps and their process will be mentioned in detail in the following sections.

When it comes to the pros offered by this solution, with clean and singularized data, technical solutions, including both AI (Artificial Intelligence) and ML (Machine Learning) models, can be implemented in an appropriate way and work more efficiently. Especially in optimization problems, when the address database is clean and well-maintained, optimization can result in not only fewer address points, reduced complexity, decreased fuel consumption, and reduced workforce requirements but also enhanced efficiency, significant time savings, and improved environmental friendliness.

## 2. Methodology

This paper focuses on a system that uses text similarity algorithms to identify and eliminate duplicate addresses. This approach prevents the storage of multiple data entries in the database that, despite pointing to the same address, could be stored as distinct entities, leading to confusion in subsequent processes. The accuracy of address data is critical in an industry such as logistics, where geographical information is essential. For this reason, finding and deduplicating multiplexing addresses in large databases is a necessary step in making the data clear and obtaining more accurate and precise results in future projects or analytical processes. This study presents the methodology of the developed method using text similarity algorithms. This method can make addresses more precise and reliable and prevent

complexity in the address pool. Thus, it makes it possible to achieve more accurate results in applications where addresses are used.

In this study, the sample dataset was selected from the addresses previously used by the company. The entire system is automated by running in the cloud environment. Microsoft Azure Cloud Services were used for this. Text similarity algorithms were used in the data analysis process, including well-known similarity scores such as Levenshtein, FuzzyWuzzy, Jaro Winkler, Hamming, and Sequence Matcher. Additionally, an in-house developed String Score algorithm was included. While some of these algorithms are suitable for the established structure, some are not. For this reason, some algorithms have been selected. In this section, after the working structure and general information of the algorithms, it is also explained why they were chosen or not.

### 2.1 Levenshtein Distance Score

One of the most used methodologies in the field of natural language processing is the Levenshtein Similarity Score. This score is effective in assessing the similarity between two text strings using edit distances, thus giving an idea of how closely they are related. Vladimir Levenshtein first created the Levenshtein similarity score in 1965 (Levenshtein, 1966). He created the score to compare DNA sequences as part of genetic research studies. This method, however, enables precise and efficient analysis and comparison of textual data, and this metric is used in many NLP algorithms, improving performance in a variety of applications such as plagiarism detection, spell-checking algorithms, and even music analysis nowadays (Giap et al., 2019).

This tool basically calculates the minimum number of changes required to convert one string into another. These changes include adding, removing, or replacing. This tool is capable of finding texts with minor differences with high accuracy. However, it may not perform well when processing large amounts of textual data. It is also possible to use the Levenshtein algorithm with various modifications depending on specific user needs or restrictions (Mangnes, 2005). For example, the Damerau-Levenshtein distance is a string measure for measuring the arrangement distance between two strings; the Jaro-Winkler distance uses character-matching probabilities instead of arranging distances; The Smith-Waterman algorithm focuses on local rather than global alignments. There are different tools used according to each need and requirement, and these tools have their strengths and weaknesses. In summary, Levenshtein shows that the similarity score is a versatile method with numerous applications in many fields, from bioinformatics research to the development of natural language processing algorithms.

Levenshtein similarity algorithm uses a two-dimensional matrix to calculate the similarity of two texts. This matrix shows the similarities of the characters of each text with the characters in the other text. The first row of the matrix represents all the characters of the first text, and the first column represents all the characters of the second text. Other cells show the number of edits between the characters of the two texts.

In the working logic of the algorithm, first of all, the texts to be compared are placed in the first row and column of the matrix. Then, the cells of the matrix are filled. If the two characters are the same, the corresponding cell of the matrix is the same as the value of the upper left cell. If the characters are different, the corresponding cell of the matrix is the smallest value, with 1 added to the values of the cell in the upper left, left, and upper corners. Finally, the value in the lower right corner of the matrix indicates the similarity of the two texts. These processes continue recursively with the following formula.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & , \text{ if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & , \text{ otherwise} \end{cases} \quad (1)$$

Here i and j are the indexes of the last character of the substring that will be compared. The second term in the last expression is equal to 1 if these characters differ or 0 if they are identical (Lhoussain et al., 2015).

Since this algorithm compares the syntax of the given strings, it is highly dependent on the word formulation and organization of the strings. If the lines basically say the same thing, but different words are used to express these meanings and/or the word structure is very different, similarity cannot be detected (Hussain, 2012). Therefore, in the method used, the repeated words in the address data cause the similarity score to be relatively low. Furthermore, while the Levenshtein similarity algorithm works well in short texts or small datasets, it may not show the desired performance in long texts and large datasets. In addition, it has a costly structure because the processing steps are performed on a character-by-character basis. This may cause more costs than necessary for a system that must operate continuously and contain large amounts of data. For these reasons, the Levenshtein Distance Score was not included in the established structure.

### 2.2 FuzzyWuzzy

One of the notable tools in the field of natural language processing is FuzzyWuzzy algorithm, a robust algorithm designed to increase the accuracy and efficiency of text data analysis. FuzzyWuzzy is, at its core, a string-matching library that uses fuzzy string-matching algorithms to compare input strings and determine their similarity. Tools like FuzzyWuzzy can leverage machine learning algorithms to visualize patterns within a body of text, opening new ways to understand complex information (Westgate, 2019). By doing this, it can quickly identify potential matches, even given imperfect or incomplete datasets. This makes it a valuable resource for anyone working with large amounts of textual data. However, what really sets FuzzyWuzzy apart from other similar tools is its ability to handle complex queries and deliver accurate results in real-time.

The Fuzzy string-matching algorithm is an open-source Python library that uses the Levenshtein Distance metric (Seatgeek, 2021). This library contains functions such as Simple Ratio, Partial Ratio, Token Sort Ratio, Token Set Ratio, and WRatio. Among these functions, the average of the results was taken by using the most suitable functions for the dataset. In this way, it is aimed to obtain a more reliable result. As a result, it is vital for the established system that the FuzzyWuzzy algorithm is customizable for different situations and contains

appropriate functions. It is an open-source library, is easily accessible, and can easily capture the similarity between texts. For this reason, this algorithm was included in the established structure.

### 2.3 Jaro-Winkler

The Jaro-Winkler similarity algorithm is a text-matching algorithm that includes a set of criteria used to measure the similarity between two text strings. This algorithm was initiated by William E. Winkler in 1989 with the development of the Jaro similarity measure.

This algorithm measures string similarity between two text strings and calculates this similarity score as a value from 0 to 1. This score indicates the level of similarity between two text strings (Wang & Wang, 2017).

The Jaro-Winkler similarity algorithm uses the following formulation to calculate the similarity between two text strings (Tinaliah & Elizabeth, 2018) (Cohen et al., 2003):

$$Jaro - Winkler\ Score = Jaro\ Similarity\ Score + (L \times p \times (1 - Jaro\ Similarity\ Score) \quad (2)$$

Here, the Jaro similarity score refers to the Jaro similarity criterion between two text strings and uses the following formulation (Jaro, 1989):

The Jaro similarity $sim_j$ of two given strings $s_1$ and $s_2$ is

$$sim_j = \begin{cases} 0 & , \ if\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & , \ otherwise \end{cases} \quad (3)$$

Where:

- $m$ refers to the number of matching characters between two strings of text.
- $s_1$ refers to the length of the first text string.
- $s_2$ refers to the length of the second text string.
- t refers to the number of different characters between two strings of text.

L and p are additional measures of the Jaro-Winkler similarity algorithm. L is a factor that increases the similarity between two text strings when there is a long common prefix. P is another factor that increases similarity between two text strings and takes a value like 0.1.

As a result, the Jaro-Winkler similarity algorithm uses the l and p metrics in addition to the Jaro similarity metric to calculate the similarity between two text strings (Winkler, 1990). Also, the algorithm is suitable for large datasets. Therefore, it is included in our final score.

### 2.4 Hamming Distance

Hamming distance is a similarity measure used to measure the differences between two different strings. This distance is calculated by comparing the characters in the two strings. The two strings are assumed to have the same length and each character is assumed to represent one bit. Hamming distance is determined by calculating the number of different characters in the same position (Hamming, 1950).

The Hamming distance can be calculated by the formula:

$$D_H = \sum_{i=1}^{k} |x_i - y_i| \qquad (4)$$

Here, x and y are two strings respectively, n is the length of the string. The symbol Σ represents the addition operation, and xi and yi represent i in the x and y arrays, respectively. represents characters. xi ≠ yi returns true when the characters xi and yi are different. The total number of different characters is called the Hamming distance (Jarrous & Pinkas, 2009).

The Hamming distance is commonly used in the fields of data transmission or coding theory for error detection and correction. However, when generating a similarity score, the position of letters is crucial, and abbreviations or digit shifts occur-ring in two different addresses when calculating the Hamming score would adversely affect the algorithm's success. Therefore, it is not suitable for use in this context.

### 2.5 String Score

The String Score algorithm is an algorithm developed by the Borusan Logistics Advanced Analytics and Artificial Intelligence team. The algorithm generates a score based on the word-level matching rate of two different address texts.

The operational structure of the algorithm is as follows:

· Since the String Score algorithm is case-sensitive, the address texts to be com-pared are converted to uppercase.

· Address texts are split, and the words that encompass the address text are obtained.

· It calculates how much the words from the 1st address text cover the words from the 2nd address text, and a percentage match is calculated based on the word count of the 1st address text.

· The same process is carried out for the 2nd address text, and a percentage match is calculated.

· The average of the two percentage matches obtained is used as the String Score.

**2.6 Sequence Matcher**

Sequence Matcher is a library in Python, and it is used to compare and differentiate between two strings. This program is widely used to determine the similarity of sequences. It takes two parameters, a and b, to calculate the similarity between two sequences. These parameters contain two arrays that need to be compared. After the algorithm calculates the similarity between two sequences, it provides a similarity score as the output.

Sequence Matcher can utilize various methods to calculate the similarity between two sequences. Some of these are listed below:

- Dictionary-based Similarity: This method calculates the similarity between two arrays. If the elements in two arrays are not the same, the Sequence Matcher stores these elements in a dictionary and attempts to find similar elements for each element.
- Serial Similarity: This method calculates the order and similarity of the elements in two arrays. It compares the elements of arrays in the same order and calculates their similarity.

Sequence Matcher provides valuable information about the similarity between two sequences. This algorithm uses various criteria to calculate the similarity scores and presents the result as a percentage. In this way, it provides specific information about the similarity between the two sequences and minimizes errors during the comparison process. Sequence Matcher algorithm has fewer configuration options. Therefore, the algorithm is not preferred in our final score.

## 3. Algorithm Architecture

The address management algorithm consists of two different blocks. These steps make up the address management system. These stages are as follows.

1. Cleaning the Address Database

2. When a new address is registered in the live system, use similarity algorithms with the addresses in the database.

### 3.1 Database Address Cleaning

Due to the complexity of logistics operational processes, our database must first consist of unique and correct addresses before address management in the address database. The developed algorithm operates blindly, enabling the classification and deduplication of addresses within the database. This algorithm works based on grid search. These process steps are seen in Figure 1.

The process steps are as follows.

• Provincial Address Search: In order to optimize the efficiency of big data in address management databases, filtering is conducted at the provincial level.

• Pre-processing Operations: All address texts of the filtered city are first converted to UTF-8. All words are then converted to uppercase letters.

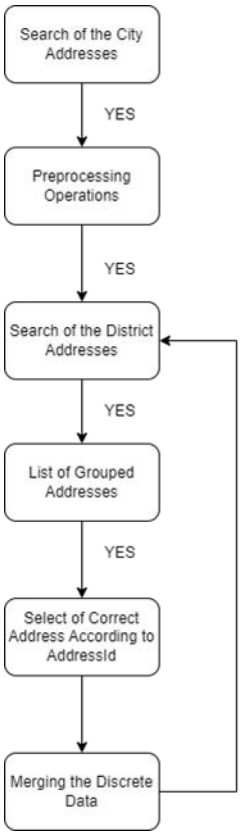
• District Address Search: List all districts belonging to the filtered province, and grid search starts to be applied in this step. The algorithm makes classification based on districts and creates the classified province dataset by induction.

• Listing of Grouped Addresses: The algorithm that works blindly makes use of Accountname variations while classifying.

• Consolidation of Classified District Addresses into Provincial Addresses: The classified data of each district obtained are combined based on the province they belong to.

The example of the result of the Grid Search Architecture given in Figure 1 is the city of Konya. According to the algorithm architecture, all addresses in this city are filtered, and the singularization addresses are grouped by the district as a color frame.

*Figure 1: Grid search structure of the algorithm architecture.*

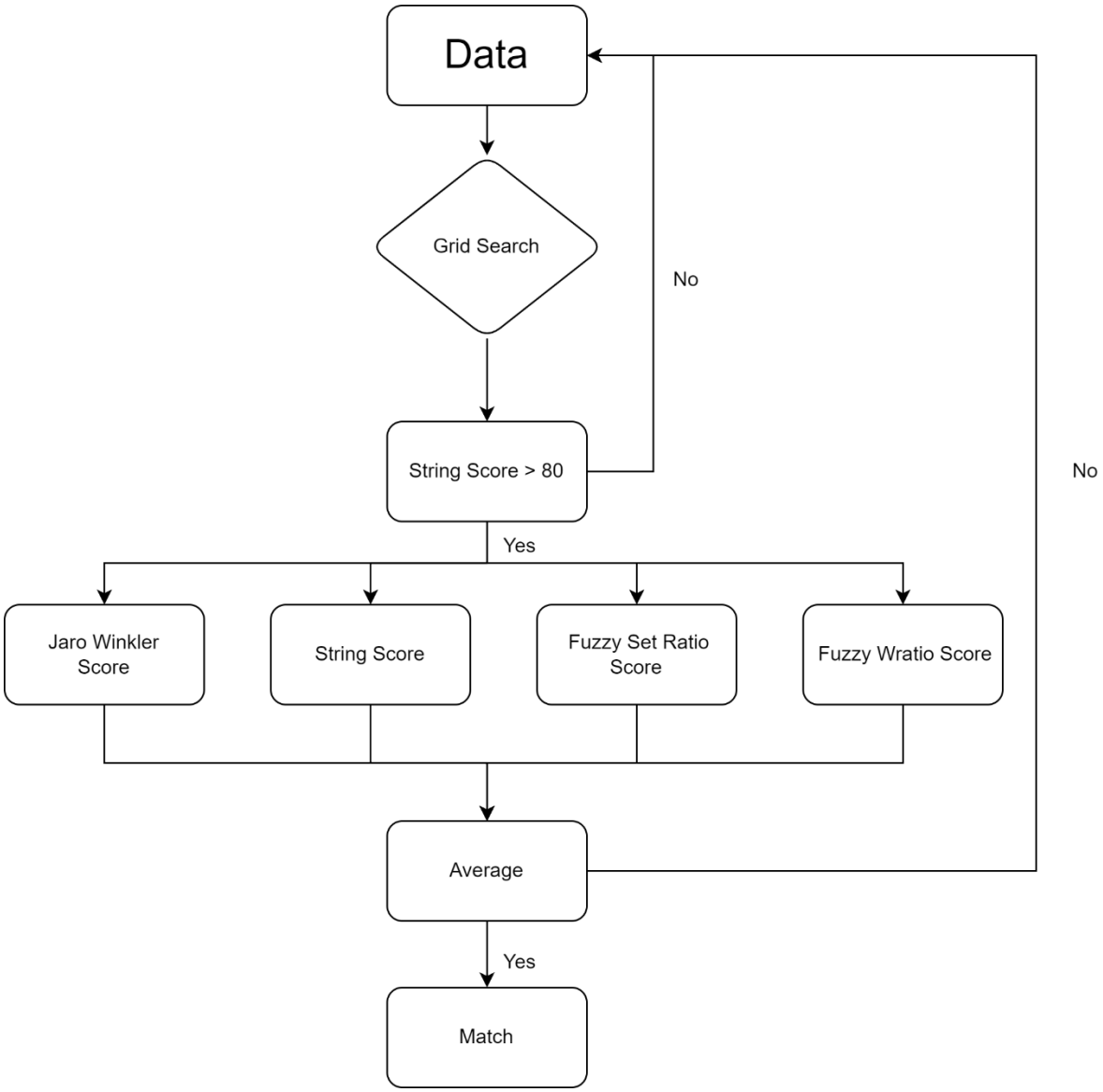




### 3.2 Detection of Multiplexed Addresses in Live System with Similarity Rate

In the logistics sector, operations are conducted on a continuous 24/7 basis. In such a dynamic system, real-time data entries are made into the databases of operational systems. The newly registered addresses have a high potential to corrupt the cleaned database. For this reason, the similarity score is produced by using String similarity algorithms for newly uploaded addresses. As a result of this similarity score, if an address in the database matches with a high score, these addresses become merged. These process steps are as in Figure 2.

*Figure 2: Block diagram of the string similarity combiner algorithm.*




The data used in our test set consists entirely of the addresses utilized in transportation operations. These datasets are entirely specific to Turkey and are present in our database. Matching scores between each address in the address pool and other addresses are calculated. Addresses with the highest matching scores are paired.

It is very possible that the database contains duplicated address data on an actively using system. For this reason, it is necessary to check the addresses that will enter the database and import them into the database in a controlled manner. At specific intervals, newly added addresses are clustered, and after applying a certain control process, matching addresses are added to the database. These control steps, as seen in Figure 2, involve the use of grid search

and similarity algorithms. During the grid search stage, city and district breakdowns are performed to provide a list of potentially similar addresses. Then, the addresses in this list that achieve a score above the threshold value determined in the String Score algorithm are retained. Subsequently, the average of the scores from other similarity algorithms is calculated. Finally, the final scores and matched addresses are displayed on the screen. At this stage, the recommended addresses can be viewed by company personnel. Addresses approved by personnel are added to the active address pool, while those not approved are placed in the passive address pool.

## 4. Results

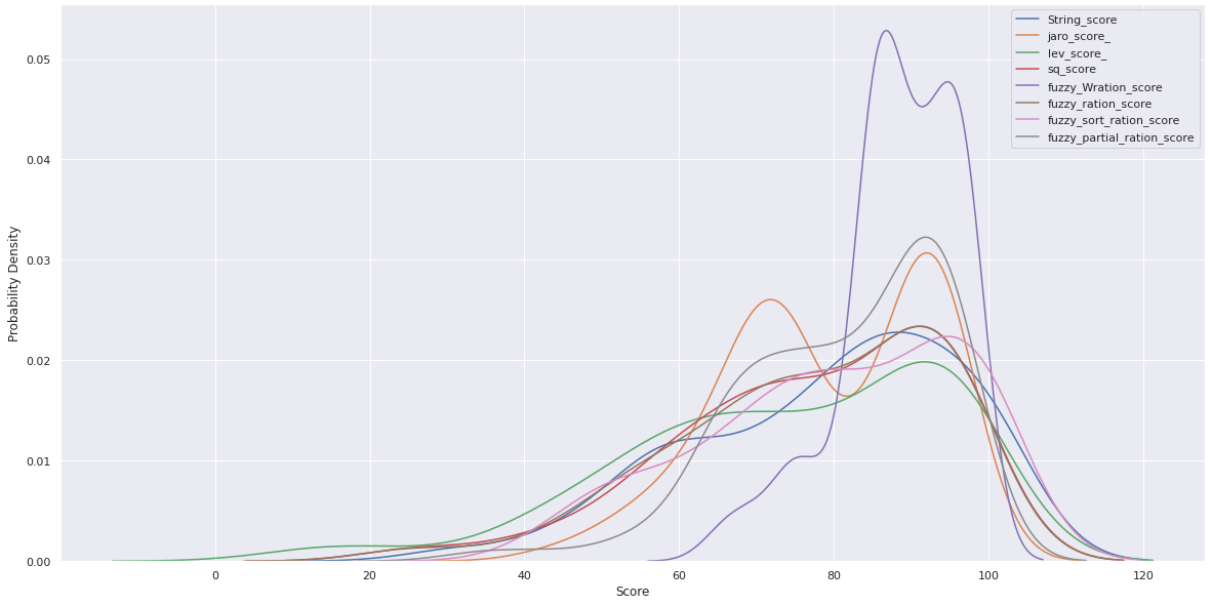
Some algorithms performed well for the used dataset, while others performed poorly. In order to achieve more accurate results, the algorithms that were most suitable for our dataset were selected. In order to minimize line-based errors in the final algorithm, the average of all selected algorithms was taken.

*Table 1: P-values of the scores.*

| | **Score** | **String score** | **Jaro score** | **Le-venshtein Score** | **sq_score** | **Fuzzy set ration score** | **Fuzzy Wration score** | **Fuzzy ration score** |
|---|---|---|---|---|---|---|---|---|
| p-values | 0.000033 | 0.000007 | 0.000003 | 0.000012 | 0.000001 | 0.000013 | 0.000035 | 0.000003 |

Our research utilized the Shapiro-Wilk test to determine the distribution of the results, which indicated that the dataset did not follow a normal distribution. The test results and their p-values for all algorithms, as presented in Table 1, reveal that all values are below 0.05, confirming that the scores deviate from a normal distribution. Consequently, due to the lack of normality, the parametric ANOVA test was deemed unsuitable; the non-parametric Kruskal-Wallis H test was chosen as an alternative.

*Figure 3: Distribution of the scores.*




The Kruskal-Wallis H test was utilized to determine statistical differences among the algorithms evaluated in the dataset. Algorithms unsuitable for the problem were identified based on the distribution of their scores, as depicted in Figure 3. The application of the H test to these algorithms resulted in high p-values, which led to the acceptance of the null hypothesis, as indicated in Table 2. This suggested the absence of any statistical difference among the algorithms considered unsuitable.

*Table 2: P-values of the algorithms.*

| | **Selected Algorithms Scores** | **Excluded Algorithm Scores** |
|---|---|---|
| p-values | $8.52 \times 10^{-7}$ | 0.13186 |

The Kruskal-Wallis H test, when applied to algorithms considered more suitable for the problem, yielded significantly low p-values, necessitating the rejection of the null hypothesis. These results suggest statistically significant differences in performance among the chosen algorithms. Such findings represent a crucial step towards the identification of algorithms that can provide more effective solutions for the problem.

After all the above-mentioned methodologies are done and the script has run, the “Score” is generated as the final score to consider whether the address texts indicate the same address or not. For calculating the “Score”, the used formula is:

$$Score = \frac{(String\ Score + Jaro\ Score + Fuzzy\ Set\ Ratio\ Score + Fuzzy\ Wration\ Score)}{4} \quad (5)$$

To observe the result and the various cases in terms of each scoring models, all models are listed below in a table and the performance of all can be seen.

*Table 3: Results of the address text comparison.*

| | Addess Text | Reference Text | Score | String score | Jaro score | Le-venshtein Score | sq_score | Fuzzy set ratio n score | Fuzzy Wration score | Fuzzy ration score |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | YAKUPLAR KOYU YAKUPLAR 5. SK. PK:14030 MERKEZ BOLU | YAKUPLAR KOYU YAKUPLAR 5. SK. PK: 14030 MERKEZ BOLU | 98.517 | 100 | 96.07 | 97 | 97.09 | 100 | 98 | 97 |
| 2 | KIRAN MH. 110. SK. PK:55090 ILKADIM SAMSUN | KIRAN MH. 110. SK. PK: 55090 ILKADIM SAMSUN | 98.214 | 100 | 94.86 | 97 | 96.55 | 100 | 98 | 97 |
| 3 | KOROGLU MH. IKMAL SK. NO:24 PK:14200 MERKEZ BOLU | KOROGLU MH. IKMAL SK. NO:24 PK: 14200 MERKEZ BOLU | 97.720 | 100 | 93.88 | 96 | 96.00 | 100 | 97 | 96 |
| 4 | AKTAS MH. PK:14200 MERKEZ BOLU | AKTAS MH. PK: 14200 MERKEZ BOLU | 97.462 | 100 | 92.85 | 97 | 95.24 | 100 | 97 | 95 |
| 5 | YUKARISOKU MH. ARCELIK SK. PK:14020 MERKEZ BOLU | YUKARISOKU MH. ARCELIK SK. PK: 14300 MERKEZ BOLU | 94.108 | 87.5 | 94.93 | 95 | 94.85 | 98 | 96 | 95 |
| 6 | KASUSTU KOYU DEVLET KARAYOLU CD. NO:64 PK:61250 YOMRA TRABZON | KASUSTU MH. DEVLET KARAYOLU CD. NO:64 PK: 61250 YOMRA TRABZON | 92.524 | 90.909091 | 90.19 | 92 | 91.20 | 97 | 92 | 91 |
| 7 | DURBINAR MH. INONUCD. NO:5 PK:61300 AKCAABAT TRABZON | DURBINAR MH. INONU CD. NO:5 PK: 61300 AKCAABAT TRABZON | 92.173 | 84.444444 | 93.25 | 95 | 95.41 | 95 | 96 | 95 |
| 8 | SAMSUN IL SAGLIK MUDURLUGU ADALET MH. PK:55060 ILKADIM SAMSUN | SAMSUN IL SAGLIK MUDURLUGU , ADALET MH. SADE SK. PK: 55060 ILKADIM SAMSUN | 91.197 | 91.666667 | 78.12 | 85 | 89.05 | 100 | 95 | 89 |
| 9 | OZDEMIRCI KOYU PK:61300 AKCAABAT TRABZON | OZDEMIRCI MH. PK: 61300 AKCAABAT TRABZON | 90.536 | 83.333333 | 91.81 | 94 | 87.80 | 96 | 91 | 88 |
| 10 | SANLIURFA EGITIM VE ARASTIRMA HASTANESI ASYA MH. PK:63200 EYYUBIYE SANLIURFA | SANLIURFA EGITIM VE ARASTIRMA HASTANESI , ASYA MH. SANLIURFA EGITIM VE ARASTIRMA HASTANESI PK: 63200 EYYUBIYE SANLIURFA | 89.191 | 100 | 70.77 | 54 | 76.77 | 100 | 86 | 77 |
| 11 | CUMHURIYET MH. BULENT ECEVIT CD. NO:19 PK:17400 CAN CANAKKALE | TOSHIBA - ARSLAN ELEKTRIK , CUMHURIYET MH. BULENT | 86.416 | 86.666667 | 73.00 | 67 | 78.71 | 100 | 86 | 79 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ECEVIT CD. NO:19 PK: 17400 CAN CANAKKALE | | | | | | | | |
| 12 | A101 MALATYA BOLGE MUDURLUGU 1. OSB MH. PK:44900 YESILYURT MALATYA | MALATYA MERKEZ 1. ORGANIZE SANAYI BOLGE MUDURLUGU , 1. OSB MH. PK: 44900 YESILYURT MALATYA | 85.678 | 84.74026 | 72.97 | 67 | 77.99 | 95 | 90 | 78 |
| 13 | BEDESTENLIOGLUOSB OSB. PK:60100 MERKEZ TOKAT | BEDESTENLIOGLU OSB MH. PK: 60100 MERKEZ TOKAT | 84.277 | 61.90476 | 91.21 | 93 | 90.11 | 93 | 91 | 90 |
| 14 | IZZET BAYSAL KUCUK SANAYI SITESI VAKIFGECITVEREN KOYU PK:14030 MERKEZ BOLU | IZZET BAYSAL KUCUK SANAYI SITESI B BLOK , VAKIFGECITVEREN KOYU (MERKEZ MH.) IZZET BAYSAL KUCUK SANAYI SITESI SK. NO:17/E B BLOK, DAIRE:E PK: 14030 MERKEZ BOLU | 81.668 | 78.571429 | 62.10 | 11 | 62.45 | 100 | 86 | 62 |
| 15 | OGULBEY KOYU RECEP TAYYIP ERDOGAN BLV. PK:63050 HALILIYE SANLIURFA | BAMYASUYU MH. RECEP TAYYIP ERDOGAN BLV. PK: 63040 HALILIYE SANLIURFA | 78.917 | 70 | 77.67 | 84 | 83.82 | 85 | 83 | 82 |
| 16 | AHMET REMZI YUREGIR MH. ERDAL ACET CD. NO:1 PK:01130 SEYHAN ADANA | ESAS 01 BURDA AVM , AHMET REMZI YUREGIR MH. TURHAN CEMAL BERIKER BLV. NO:1 PK: 01130 SEYHAN | 77.700 | 66.239316 | 70.56 | 57 | 69.88 | 88 | 86 | 70 |
| 17 | KONAK MH. SILOPI BLV. CIZRE GUCLUKONAK YOLU YAFES CD.CIZRE SILOPI YOLU SIRNAK/1. SK. PK:73200 CIZRE SIRNAK | YAFES MH. MEM U ZIN BLV. PK: 73200 CIZRE SIRNAK | 72.918 | 60 | 58.67 | 29 | 40.00 | 87 | 86 | 45 |
| 18 | TUMPA ELEKTRIK MALZEMELERI TIC. VE PAZ. A.S. YESILOBA MH. PK:01100 SEYHAN ADANA | ONUR MH. PK: 01100 SEYHAN ADANA | 71.754 | 60.897436 | 49.12 | 46 | 48.21 | 91 | 86 | 50 |
| 19 | BORUSAN LOJISTIK YESILYALI KOYU PK:61900 ARSIN TRABZON | ONURLAR , YESILYALI MH. 11. CD. PK: 61900 ARSIN TRABZON | 70.353 | 59.027778 | 68.38 | 73 | 64.29 | 79 | 75 | 68 |
| 20 | CARREFOURSA SUPER DURUSEHIR DEREBAHCE MH. PK:55060 ILKADIM SAMSUN | GUZELDERE MH. ANKARA BLV. PK: 55060 ILKADIM SAMSUN | 66.134 | 59.027778 | 67.51 | 64 | 58.12 | 71 | 67 | 56 |
| 21 | AGIRTAS KOYU BIM KM SANLIURFA GAZIANTEP YOLU BLV. ADA 135 16 PK:63800 SURUC SANLIURFA | IPEKYOL MH. IPEKYOLU BLV. PK: 63050 HALILIYE SANLIURFA | 58.628 | 32.083333 | 63.43 | 46 | 26.95 | 53 | 86 | 27 |

*Source: The company's own data.*

After calculating all the scores by using different methods mentioned in methodology, addresses are indexed in descending orders in terms of the "Score" and in this way, especially by looking at lowest scores, the performance of each method can be observed clearly. In different situations, different solutions get high score. For example, at index 10, when the coming address texts is short and is included in reference text (reference text is longer and repeat itself), Levenshtein get 54 and sq (Sequence) scores generate 76.77 score while Fuzzy set ration give 100 as a score. Because Fuzzy consider the repeated text and look whether the coming address is in reference address or not. Fuzzy gives high score but, since it needs to be reliable, model needs to give stable and accurate score, Therefore the algorithm is written, and the formula given at the beginning of this section is found by taking average of 4 scores and applied in this project.

## 5. Conclusion

In summary, effective address management is essential in the globalizing environment of the logistics industry. Having unwanted data in the address pool is very important for an industry such as the logistics industry, where geographical information is very critical. Multiple addresses or incorrect addresses cause financial losses and inefficiency for the company. In addition, this situation is likely to cause deviations and even errors in potential projects to be carried out with these addresses. In this study, a solution consisting of two basic steps: cleaning the address pool and preventing contamination is proposed. Since there cannot be a single correct result when establishing the general structure, the algorithms that will work most appropriately in our own dataset were determined, and the results of more than one text similarity algorithm were evaluated in order to increase the reliability of the general structure. After reviewing versatile similarity algorithms such as Levenshtein, FuzzyWuzzy, Jaro-Winkler, Hamming Distance, String Score, and Sequence Matcher, in the next stage, the algorithms suitable for the established structure were selected and combined appropriately.

The proposed solution promises to solve the problem in two important stages: cleaning the address pool and preventing it from being contaminated again. It also increases the efficiency of artificial intelligence and machine learning models, which can be used especially in tasks such as optimization, which can significantly reduce the company's cost and improve operational quality. This efficiency leads to cost savings and reduced environmental impact. Maintaining a clean address database is crucial to streamline logistics and supply chain management in a technology-driven world, offering a path to a more efficient, cost-effective, and sustainable future.

In future work, the selected algorithms will be combined with machine learning clustering algorithms. Thus, the mathematical algorithms would be supported by machine learning, and this system would be smarter than the current architecture.

**Acknowledgment**

This paper is the output of the 'Address Singularization and Address Management' project. This project emerged as a requirement of route optimization. Thus, the accuracy rate in addresses increased, and operational efficiency was achieved.

## References

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003, August). A Comparison of String Distance Metrics for Name-Matching Tasks. In IIWeb (Vol. 3, pp. 73-78).Chen, W. K. (1993). *Linear Networks and Systems*, Belmont, CA: Wadsworth, pp. 123-135.

Comber, S., & Arribas-Bel, D. (2019). Machine Learning Innovations in address matching: A practical comparison of word2vec and crfs. *Transactions in GIS*, *23*(2), 334–348. https://doi.org/10.1111/tgis.12522

Giap, Y.C., Johari, M., Indrawan, P., & Dedih. (2019). Implementation of the Levenshtein Distance Method and Similarities in Checking the Equal Content of the Document Text. *International Journal of Recent Technology and Engineering*, 7(6S5), 38. https://www.ijrte.org/wp-content/uploads/papers/v7i6s5/F10070476S519.pdf.

Guermazi, Y., Sellami, S., & Boucelma, O. (2020). Address validation in transportation and Logistics: A machine learning based entity matching approach. *ECML PKDD 2020 Workshops*, 320–334. https://doi.org/10.1007/978-3-030-65965-3_21

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal*, 29(2), 147-160.

Hosseinnia Shavaki, F., & Ebrahimi Ghahnavieh, A. (2023). Applications of deep learning into supply chain management: a systematic literature review and a framework for future research. *Artificial Intelligence Review*, 56(5), 4447-4489.

Hussain, A. (2012). Textual Similarity. Bachelor Thesis. *Kongens Lyngby: University of Denmark.*

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406), 414-420.

Jarrous, A., & Pinkas, B. (2009). Secure hamming distance based computation and its applications. *In Applied Cryptography and Network Security: 7th International Conference, ACNS* 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings 7 (pp. 107-124). Springer Berlin Heidelberg.

Koumarelas, I., Kroschk, A., Mosley, C., & Naumann, F. (2018). Experience: Enhancing address matching with geocoding and similarity measure selection. *Journal of Data and Information Quality*, 10(2), 1–16. https://doi.org/10.1145/3232852

Levenshtein, V.I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 10, 707-710.

Lhoussain, A. S., Hicham, G., & Abdellah, Y. O. U. S. F. I. (2015). Adaptating the levenshtein distance to contextual spelling correction. *International Journal of Computer Science and Applications*, 12(1), 127-133.

Mangnes, B. (2005). The use of Levenshtein distance in computer forensics (Master's Thesis), Department of Computer Science and Media Technology, Gjøvik University College. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=abcc79845ed7bed1e97cad2fb27ab4c1adb94b99

Seatgeek. (n.d.). GitHub - seatgeek/thefuzz: Fuzzy String Matching in Python. *GitHub.* https://github.com/seatgeek/thefuzz

Tinaliah, T., & Elizabeth, T. (2018). Perbandingan Hasil Deteksi Plagiarisme Dokumen dengan Metode Jaro-Winkler Distance dan Metode Latent Semantic Analysis. *Jurnal Teknologi dan Sistem Komputer*, 6(1), 7-12.

Wang, Y., Qin, J., & Wang, W. (2017, October). Efficient approximate entity matching using jaro-winkler distance. *In International conference on web information systems engineering* (pp. 231-239). Cham: Springer International Publishing.

Westgate, M. J. (2019). revtools: Tools to support evidence synthesis. *R package version 0.4*, 1.

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage.

Yildirim, V., Yomralioglu, T., Nisanci, R., & Inan, H. (2014). Turkish street addressing system and geocoding challenges. *Proceedings of the Institution of Civil Engineers - Municipal Engineer*, 167(2), 99–107. https://doi.org/10.1680/muen.13.00008