PD.TVZ.HR
POLYTECHNIC & DESIGN Vol. 14, No. 2, 2026.
Mrežna inačica: ISSN 2459-6302
Tiskana inačica: ISSN 1849-1995
DOI: 10.19279/TVZ.PD.2026-14-2-12

PREGLED ALGORITAMA ZA PODUDARANJE NIZOVA KOD INTEGRACIJE PODATAKA

OVERVIEW OF STRING-MATCHING ALGORITHMS IN DATA INTEGRATION

Aleksandar Stojanović, Silvio Plehati, Željko Kovačević

Zagreb University of Applied Sciences, 10000 Zagreb, Croatia

SAŽETAK

Učinkovita integracija podataka iz različitih izvora u dobu digitalizacije ključna je za točnost i upotrebljivost informacija, a predstavlja izazov koji algoritmi za podudaranje nizova nastoje riješiti. Kako količina podataka raste, potreba za naprednijim tehnikama podudaranja postaje sve izraženija. Ovaj pregled kategorizira algoritme u četiri glavne skupine: algoritme zasnovane na sekvenci, algoritme zasnovane na skupovima, hibridne i fonetske algoritme. Algoritmi zasnovani na sekvenci, poput Levenshteinove udaljenosti i Smith–Watermana, ističu se preciznošću na razini znakova. Metode zasnovane na skupovima, uključujući Jaccardovu i kosinusnu sličnost, učinkovite su pri radu s tokeniziranim podacima. Hibridni pristupi kombiniraju njihove prednosti i nude fleksibilnost u složenim scenarijima. Fonetski algoritmi, poput Soundexa, Metaphonea i NYSIIS-a, omogućuju podudaranje prema izgovoru te su posebno korisni kod imena i pojmova s jezičnim varijacijama. Pregled naglašava važnost stalnog razvoja svih pristupa kako bi se odgovorilo na suvremene izazove integracije podataka.

Ključne riječi: *podudaranje nizova, mjere sličnosti, integracija podataka, otkrivanje duplikata*

ABSTRACT

Efficient integration of data from different sources in the digital age is crucial for the accuracy and usability of information, and it represents a challenge that string-matching algorithms aim to address. As the volume of data grows, the need for more advanced matching techniques becomes increasingly evident. This review categorizes algorithms into four main groups: sequence-based, set-based, hybrid, and phonetic algorithms. Sequence-based algorithms, such as the Levenshtein distance and Smith–Waterman, stand out for their precision at the character level. Set-based methods, including Jaccard and cosine similarity, are effective when working with tokenized data. Hybrid approaches combine the strengths of these methods and offer flexibility in complex scenarios. Phonetic algorithms, such as Soundex, Metaphone, and NYSIIS, enable matching based on pronunciation and are particularly useful for names and terms with linguistic variations. The review highlights the importance of the continued development of all approaches to address modern challenges of data integration.

Keywords: *string matching, similarity measures, data integration, duplicate detection*

1. UVOD
1. INTRODUCTION

Today, as data is generated from countless sources, the ability to integrate and utilise this information has become the foundation for knowledge discovery and decision-making processes [1]. Data integration involves combining information from different sources and providing users with a unified and consistent view of the data [2]. A key challenge in this

CC BY NC

process is accurately matching strings that represent the same entity in different datasets, due to variations in records, typographical errors, and other inconsistencies. String matching algorithms, which assess the similarity between character strings, are fundamental tools for addressing this problem. They enable more efficient data cleaning, duplicate detection, and entity resolution [3]. These algorithms facilitate the identification of related records in databases, files, or other repositories, even when deviations naturally occur during data collection and processing. The development and evolution of these algorithms have spanned several decades, with intensified activity in the last five to ten years, parallel to the increase in data volume and the application of machine learning in data integration. This work provides an overview and comparison of existing approaches and contributes to the field through the systematisation and classification of string-matching algorithms. A particular contribution of this work is the structuring of algorithm classification into four main groups – sequence-based, set-based, hybrid, and phonetic – and the analysis of their applicability in modern areas such as duplicate detection and intelligent data integration.

2. PREGLED ISTRAŽIVANJA TEMELJNIH ALGORITAMA ZA PODUDARANJE NIZOVA

2. OVERVIEW OF FUNDAMENTAL RESEARCH ON STRING-MATCHING ALGORITHMS

This paper analyses fundamental algorithms developed from the 1970s to the present, with an emphasis on modern applications and variants developed in the last five years. The study of string-matching algorithms spans decades, starting with methods based on exact substring matching. Early contributions include the Knuth–Morris–Pratt algorithm (1977), which reduced search complexity [4], and the Boyer–Moore algorithm (1977), known for heuristic optimisations [5]. In data integration, where exact matches are rare due to typographical errors and variations in records, Levenshtein distance (1966) became a fundamental similarity measure [3], while Jaro–Winkler distance proved particularly effective for shorter strings and typographical errors [3]. To overcome the limitations of sequence-based measures, set-based methods like the Jaccard index were introduced [3], while hybrid approaches such as Monge–Elkan combine advantages of both [3]. According to Christen [3], similarity measurement approaches are typically classified into four main categories: sequence-based methods (character- or sequence-based) compare strings character by character and precisely identify typographical errors, making them suitable for data cleaning and duplicate detection; set-based methods (token-based) treat strings as sets of tokens and ignore order, offering simpler and more efficient estimations, especially for large data collections; hybrid methods combine sequence- and set-based approaches, providing balanced and context-sensitive assessments, particularly for heterogeneous data sources; phonetic methods compare strings by pronunciation (e.g., Soundex, NYSIIS, Double Metaphone, Fuzzy Soundex) and enable recognition of variations in names and toponyms where traditional algorithms often fail. Christen [3] emphasises that this classification is based on different similarity concepts, and each algorithm group has specific advantages depending on data type and application context. Although this paper presents classic algorithms from the 1960s and 1970s (for historical and methodological context), the analysis focuses on variants and applications developed in the last five years, treating earlier works as foundational references systematically presented by Christen. Recent trends aim to increase scalability and efficiency, applying machine learning and deep learning to tasks such as entity resolution and duplicate detection [6]. The application domain has expanded to bioinformatics (DNA sequence comparison) and cybersecurity (malware identification) [7]. Despite advances, challenges remain in handling high-dimensional and unstructured data and real-time processing. Future research focuses on developing adaptive and intelligent algorithms that automatically tune parameters according to data properties [8].

3. ALGORITMI ZA PODUDARANJE NIZOVA ZASNOVANI NA SEKVENCI

3. SEQUENCE-BASED STRING-MATCHING ALGORITHMS

Algorithms for sequence-based string matching assume that similarity between two texts can be assessed by comparing each character and its position in the sequence. Unlike methods that treat text as a set of words or tokens, these techniques record every change in the character string, allowing precise measurement of differences and reliable determination of similarity levels. This approach is especially important when even minor differences, such as typographical errors or character substitutions, can significantly affect the matching result. Algorithms that compare strings character-by-character include: *the Levenshtein*, *Jaro-Winkler*, and *Smith-Waterman algorithms* [3]. Similarity between two strings *x* and *y* is measured by a similarity function s that returns a value between *[0, 1]*, where a higher value indicates greater similarity.

The *Levenshtein distance* is an algorithm that measures similarity using the minimum number of editing operations required to convert one string into another [3]. This distance quantifies the difference between two strings by counting the minimum number of single-character edits (insertion, deletion, or substitution) needed to change one string into the other. If *d(x, y)* is the minimum number of operations required to transform string *x* into string *y*, this measure can also be converted into a similarity function *s(x, y)* as follows:

$$s(x,\ y) = 1 - \frac{d(x,\ y)}{max(length(x),\ length(y))}$$

For example, consider the words “rubac” and “ručak”. The Levenshtein distance between them is 2 (substitution of b → č and c → k). As the longer string has a length of is 5, the similarity is 0.6 (60%). The Levenshtein algorithm is limited by its high time complexity of $O(|a|\cdot|b|)$ and is impractical for long sequences. Memory usage can be reduced to $O(\min(|a|,|b|))$, but time remains the main issue.

Jaro-Winkler similarity is an extension of the Jaro algorithm and, compared to Levenshtein distance, places greater emphasis on matching characters at the beginning of the string [3]. The Jaro similarity introduces the concept of pairs of matching characters within a window and penalises transpositions. The Winkler extension increases the weight of the common prefix – strings that start with the same sequence of characters are considered more similar. Jaro-Winkler more effectively recognises typical errors in names (“Marija” ↔ “Marjia”, “Đuro” ↔ “Djuro”). For strings s and t of lengths |s| and |t|, Jaro similarity calculates the number of matches *m* and the number of transpositions *t* within a window size of $max(|s|,|t|)/2 - 1$ as follows:

$$J = \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m - \frac{t}{2}}{m}\right)$$

For example, if we consider the words “*Marija*” and “*Marjia*”, the number of matching characters is *m=6*, the number of transpositions is t=1, and the Jaro similarity is J=0.944 (94.4%). The Winkler correction takes *l* as the length of the common prefix (up to 4 characters), and *p* as the scaling factor (usually 0.1), while *J* is the Jaro similarity. The Jaro-Winkler similarity is calculated in the following way $JW=J+l \cdot p \cdot (1-J)$. If we take *l=3, p=0.1,* and *J=0.944*, then the Jaro-Winkler similarity is *JW=0.961 (96.1%)*. The algorithm has a time complexity of approximately O(|a|+|b|), which makes it significantly faster than the Levenshtein distance. It is suitable for comparing a large number of short strings, such as personal names. Its effectiveness decreases for longer texts, and emphasising the common prefix can lead to bias when the strings share the same beginning but differ significantly in the remainder.

The *Smith-Waterman algorithm* goes one step further by seeking the best local alignment instead of global similarity, i.e., it recognizes similar substrings within longer strings [3]. This is particularly important when data is uneven, noisy, or when two strings only partially overlap (e.g., a short phrase within a long text). The algorithm uses dynamic programming with scoring: a match is rewarded, a mismatch and gaps are penalised, and the matrix is filled according to the rule $H[i,j]=max(0,\ H[i\text{-}1,j\text{-}1] + s(a_i, b_j),\ H[i\text{-}$

$1,j]+\text{gap}$, $H[i,j-1]+\text{gap}$) where *s(x,y)= e.g. +2* if *x=y*, otherwise *−1*, and *gap=−2*. The alignment is terminated as soon as the *alignment value* drops below zero, so the algorithm returns the best local match, not necessarily the alignment of the entire strings . For example, when comparing the words "bilježnica" and "proljetnica", the algorithm with parameters *match = +2, mismatch = –1, gap = –2* will find the local alignment *"ležnica"* ↔ *"ljetnica"* with an alignment value of 13 . Although the entire words differ significantly, the algorithm correctly highlights the strongest common substring within the longer strings. The algorithm is highly accurate because it finds the optimal local alignment between strings, making it especially useful in bioinformatics and with data that has partial overlaps. However, its time complexity O(|a|·|b|) and substantial memory requirements make it computationally expensive, so it is mainly used in practice for shorter strings.

Algorithms based on character sequences enable precise measurement of string similarity by considering every change at the character level. They provide a flexible framework applicable to tasks ranging from data cleaning to bioinformatics, and future developments include improving efficiency and exploring new application areas (see [6] and [8]). Combining these with other similarity measures can result in a holistic approach to sequence matching, tailored to the requirements of modern data integration. This review highlights key aspects of sequential methods and lays the foundation for considering set-based and hybrid algorithms presented in the following chapters.

4. ALGORITMI ZA PODUDARANJE NIZOVA ZASNOVANI NA SKUPOVIMA ELEMENATA

4. ALGORITHMS FOR STRING MATCHING BASED ON SETS OF ELEMENTS

Algorithms for string matching based on sets of elements represent an important approach to measuring similarity, particularly in data integration and duplicate detection tasks. Unlike algorithms that focus on character order, these methods treat strings as sets of tokens, words, characters, or n-grams and assess similarity using set operations such as intersection, union, or difference. We will present the Jaccard, Sørensen-Dice, and cosine similarity algorithms [3].

The *Jaccard algorithm* measures the similarity between two sets containing strings [3]. It is calculated as the ratio of the intersection to the union of the two sets. Unlike algorithms that compare strings based on character order, the Jaccard approach treats the string as a set of tokens, words, or n-grams, where order is not important. The Jaccard index as a similarity measure takes values between 0 and 1, where 0 means the sets have no common elements, and 1 indicates that the sets are identical. It is calculated as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are sets of elements from two sequences, |A∩B| denotes the number of elements in the intersection of the two sets, and the number of elements in their union. For example, if we take the words "*bilježnica*" and "*proljetnica*" and break them into bigrams, the intersection consists of 5 elements {lj, je, ni, ic, ca} while the union consists of 14 elements {bi, il, lj, je, už, žn, ni, ic, ca, pr, ro, ol, et, tn}. The Jaccard index is approximately 0.357, i.e., about 35.7% similarity at the bigram level. Although the whole words are different, the algorithm clearly highlights the common part. The Jaccard index stands out for its simplicity and efficiency and is particularly suitable for comparing sequences in data integration and duplicate detection tasks. Its time complexity is *O(|A|+|B|)*, as it is based on calculating the intersection and union of two sets, and memory requirements are relatively low, making it applicable to larger datasets.

The *Sørensen–Dice algorithm* is based on the similarity coefficient of two sets which, unlike the Jaccard index, gives double weight to the size of their intersection relative to the sum of the number of elements in both sets [3]. This makes it particularly suitable for duplicate detection tasks and natural language processing, where emphasising overlaps is important. The Sørensen–Dice similarity coefficient is calculated as follows:

$$D(A,B) = \frac{2\,|A \cap B|}{|A| + |B|}$$

where A and B are sets of elements from two sequences, and |A∩B| denotes the number of elements in the intersection of the two sets. For example, if we take the words "*bilježnica*" and "*proljetnica*" and break them into bigrams, the intersection consists of 5 elements {lj, je, ni, ic, ca}. Each set contains nine elements, and the Sørensen–Dice coefficient is approximately 0.526, indicating 52.6% similarity. The time complexity of the Sørensen–Dice algorithm is $O(|A|+|B|)$, as all elements of both sets must be processed for the calculation, while memory requirements are low. Therefore, it is suitable for larger datasets and, like the Jaccard index, gives the best results when comparing sequences represented as tokens or n-grams.

Cosine similarity measures the similarity between two sequences by representing them as vectors in multidimensional space, and calculating the cosine of the angle between them [3]. This measure is particularly useful when sequences are broken down into tokens or n-grams and their frequencies are recorded, as similarity then reflects how closely their vectors are aligned. Cosine similarity is calculated as the cosine of the angle between two vectors representing the sequences. The smaller the angle (i.e. the cosine is closer to 1), the more similar the sequences. If the vectors are completely perpendicular (cosine = 0), the sequences are considered completely different. Cosine similarity is calculated as follows:

$$\mathrm{sim}(A,B) = \cos(\theta) = \frac{A \cdot B}{|A| \cdot |B|}$$

where A and B are sets of elements from two sequences. For example, if we take the words "*bilježnica*" and "*miljenica*" and break them into bigrams, the number of elements in each set is $|A|=9, |B|=8$. The common bigrams are $|A \cap B| = il,lj,je,ni,ic,ca \rightarrow 6$ {lj, je, ni, ic, ca} so $\|v_A\|=\sqrt{9}=3$, $\|v_B\|=\sqrt{8}$, $v_A \cdot v_B=6$, and when these values are inserted into the above expression the cosine similarity is approximately 0.707 which gives 70.7% similarity. Cosine similarity has time complexity $O(|A|+|B|)$ and moderate memory requirements, and due to efficient comparison, it is often used in search and recommendation systems.

Set-based similarity focus on emphasize the composition of sequences rather than their order, making them particularly useful for data integration and duplicate detection. Algorithms such as the Jaccard index, Sørensen–Dice coefficient and cosine similarity have proven effective in document analysis, product feature matching and other practical applications. Their advantages are simplicity and low computational requirements, enabling work with larger datasets. Future developments aim to incorporate the frequency and importance of elements and to combine set-based and sequential approaches, thus creating hybrid similarity measures (according to [6], [8]) adapted to more complex tasks.

5. HIBRIDNI ALGORITMI KOJI KOMBINIRAJU SEKVENCIJSKI PRISTUP I PRISTUP ZASNOVAN NA SKUPOVIMA

5. HYBRID ALGORITHMS COMBINING SEQUENCE-BASED AND SET-BASED APPROACHES

Hybrid algorithms that combine the sequential approach and the set-based approach aim to merge the advantages of both worlds. This combines the precision of algorithms based on character order with the flexibility of measures that treat strings as sets of elements. Hybrid approaches often achieve a better balance between accuracy and computational efficiency, and are used in text analysis, information retrieval, and recommendation systems. Next, we will describe the Monge–Elkan, SoftTFIDF, and Hybrid Jaccard algorithms [3].

The *Monge–Elkan algorithm* combines set-based algorithms (where strings are viewed as sets of tokens, such as words) and sequential algorithms (similarity between tokens is calculated using another measure, such as Levenshtein or Jaro–Winkler). Instead of simply comparing the intersection of tokens (as in Jaccard), Monge–Elkan calculates the pairwise similarity of each token from one string with the most similar token

in the other string. The result is the average of these maximum values. If *S* and *T* are two sets of tokens, and "sim" (s,t) is a similarity measure between tokens (e.g., Jaro–Winkler), Monge–Elkan is calculated as:

$$ME(S,T) = \frac{1}{|S|}\sum_{s\in S}\max_{t\in T}\mathrm{sim}(s,t)$$

The symmetric version is often calculated in reverse and the average of ME(S,T) and ME(T,S) is taken. In the example of the strings "*bilježnica*" and "*miljenica*", where each string consists of one token, the results obtained are *ME(A,B)*=0.896 and *ME(B,A)*=0.896, so the symmetric version is equal. The Monge–Elkan algorithm is computationally more expensive than simple set-based measures because for each token from one string, the most similar token from the other string must be found using the chosen sequential similarity measure (e.g., Levenshtein or Jaro–Winkler). The time complexity is $O(|S|\cdot|T|)$, with quadratic growth in the number of tokens and further increase for more complex metrics such as Levenshtein. Memory requirements are low, but execution time remains the main limiting factor for large datasets.

The *SoftTFIDF algorithm* extends traditional TF–IDF by introducing a similarity threshold when comparing tokens [3]. Unlike classical TF–IDF, which recognises only identical terms, SoftTFIDF allows partial matching of similar terms (e.g., due to typos or morphological variations) if their similarity exceeds the threshold θ, thus better reflecting real cases of input inaccuracies or different word forms. If *S* and *T* are sets of tokens from two strings or documents, w(x) is the TF–IDF weight of token *x*, sim(s,t) is the similarity measure between tokens (e.g., Jaro–Winkler), and the threshold θ above which similarity is recognised, SoftTFIDF is calculated as follows:

$$\mathrm{SoftTFIDF}(S,T) = \sum_{s\in S} w(s)\cdot \max_{\substack{t\in T \\ sim(s,t)\geq\theta}} (sim(s,t)\cdot w(t))$$

Using the example of the strings "*bilježnica*" and "*miljenica*", where each string constitutes one token, the SoftTFIDF result is 0.896, or 89.6% similarity. This demonstrates that the algorithm effectively recognises a high degree of similarity between the two words despite spelling differences. Unlike classical TF–IDF, SoftTFIDF includes a similarity threshold when comparing sets of terms, allowing recognition of words that are similar but not identical. SoftTFIDF improves matching accuracy in cases of minor variations, typographical errors, or input errors, making it valuable in data integration, duplicate detection, and text retrieval. The algorithm is computationally more demanding than purely set-based measures, with a complexity of $O(|S|\cdot|T|\cdot c_{sim})$. Although scaling can be improved with heuristics and indexing, the basic complexity remains quadratic, while memory requirements are moderate.

The *Hybrid Jaccard algorithm* extends the traditional Jaccard index [3]. It combines the sequential and set-based approaches by using similarity measure between elements, rather than requiring exact matches. In other words, the intersection and union of sets are determined based on sequential similarity, not solely on identity. The Hybrid Jaccard is calculated as follows:

$$HJ(A,B) = \frac{\sum_{s\in A}\max_{t\in B} sim(s,t)}{|A|+|B|-\sum_{s\in A}\max_{t\in B} sim(s,t)}$$

In the example A={"bilježnica"} and B={"miljenica"}, with Jaro–Winkler similarity ≈0.896, the Hybrid Jaccard is ≈0.812 (81.2%). The algorithm considers both set intersection and the sequential similarity of elements, combining the simplicity of set-based similarity with the precision of sequential measures. It is suitable for data integration and duplicate detection, with a time complexity of $O(|A|\cdot|B|\cdot c_{sim})$ and low memory requirements.

Hybrid similarity measures combine the advantages of sequential and set-based measures, enabling more accurate and context-sensitive evaluations and offering solutions to complex data integration challenges. They are particularly useful when working with diverse sources, and future development is aimed at greater computational efficiency and the application of machine learning (according to, [6], [8]) for dynamic adaptation of the approach. This chapter highlights their key role and the importance of further research in the field of sequence matching algorithms.

6. ALGORITMI ZA FONETSKO PODUDARANJE NIZOVA NA TEMELJU IZGOVORA

6. ALGORITHMS FOR PHONETIC STRING MATCHING BASED ON PRONUNCIATION

Algorithms for phonetic string matching based on pronunciation compare strings by how they sound, rather than solely by their spelling. This approach is particularly useful in data integration involving personal names, place names, or other textual data where phonetic variations are common due to dialects, language differences, or spelling errors. By assessing similarity at the pronunciation level, phonetic algorithms enable more accurate record linkage and duplicate detection in situations where classical string matching methods might fail. Here we will present the Soundex, Metaphone, and NYSIIS algorithms [3].

The Soundex algorithm is one of the earliest and most well-known phonetic algorithms for encoding strings, especially personal names, according to their pronunciation [3]. The algorithm assigns each word a code consisting of the initial letter and three digits, with vowels generally ignored and consonants grouped into classes based on sound similarity. In this way, different spellings of the same or similar name (e.g., "Šime" and "Sime") receive the same or very similar code, allowing easy recognition of phonetic matches in databases. Soundex can be represented as $Soundex(w)=L\,d_1d_2d_3$ where w is the input word, L is the first letter of the word, and are the first three digits obtained by replacing consonants with numerical codes. If there are not enough digits, zeros are added. The algorithm keeps the first letter, and the other letters are coded according to the rules (1 = B, F, P, V; 2 = C, G, J, K, Q, S, X, Z; 3 = D, T; 4 = L; 5 = M, N; 6 = R), while vowels and H, W, Y are ignored. For example: "Dario" → D600, "Mario" → M600; both names share the numeric part (600) and differ only in the initial letter, so Soundex considers them similar but not identical. The algorithm is linear and very fast, suitable for large name databases, but often produces false positives due to collisions.

The *Metaphone algorithm*, along with its improved version *Double Metaphone*, was developed to address the limitations of Soundex and provide more accurate phonetic encoding [3]. Rather than relying on a simple table, it employs more complex rules that account for cover a wider range of phonetic variations. Double Metaphone further supports the encoding of non-English sounds and generates both a primary and a secondary code for alternative pronunciations. Thus, Metaphone and its variants achieve higher accuracy in recognising phonetically similar words. The codes do not have a fixed length but instead reflect the phonetic structure of the word. The procedure can be represented as a function $Metaphone(w)=C(w)$ where w is the input word and $C(w)$ is a string of phonetic symbols obtained by applying a set of phonetic rules. For the Double Metaphone algorithm, the output is an ordered pair $DoubleMetaphone(w)=(C_p(w),C_s(w))$, where $C_p(w)$ is the primary code and $C_s(w)$ is the secondary code for an alternative pronunciation. For the words "Mario" and "Mara", Metaphone and Double Metaphone produce the same code "MR", treating them as phonetically identical, even though they differ in speech. This illustrates limitation, as it generalises too broadly and can result in false positive matches. Both algorithms have time complexity $O(|w|)$ and space complexity $O(1)$, with Double Metaphone additionally generating a primary and a secondary code, but without a significant increase in computational cost.

NYSIIS (*New York State Identification and Intelligence System*) was developed to achieve greater accuracy in name matching than Soundex [3]. Instead of short and coarse codes, it generates longer and more phonetically accurate codes by applying more detailed transformation rules, thereby reducing false positive matches. It is particularly effective for Spanish and Italian names but can also be applied to other languages with notable phonetic variations. Compared to Soundex and Metaphone, NYSIIS lies in the middle, and is simple to implement, providing higher phonetic precision. Its codes do not have a fixed length and are based on rules that normalise vowels, group consonants, and remove final variations. The process can be expressed as a function $NYSIIS(w)=C(w)$, where w is the input word and $C(w)$ is the phonetic code obtained through algorithmic transformation rules. For

example, the words “Pero” and “Pere” both yield the code “PER” when processed by the NYSIIS algorithm, demonstrating its advantage in recognising morphological variations (in final vowels) and treating them as phonetically identical. The NYSIIS algorithm has a time complexity of $O(|w|)$ as it processes the word character by character applying a fixed set of rules, and its space complexity is $O(1)$ since it uses only a limited number of auxiliary variables.

Phonetic similarity measures focus on sound rather than spelling, complementing traditional methods and recognising variations in names, toponyms, and other data. Algorithms such as Soundex, Metaphone, and NYSIIS are simple and effective, and further advances in linguistics and machine learning may improve their accuracy (according to [6], [8]) and extend their application to other languages. As international data diversity increases, phonetic measures will play an increasingly important role in accurate data integration.

7. ZAKLJUČAK
7. CONCLUSION

Research on string matching algorithms, including sequence-based, set-based, hybrid, and phonetic similarity measures, highlights the diversity and complexity of techniques available for data integration tasks. These methods enable the recognition of typos, spelling variations, and phonetic deviations while effectively identifying duplicates. Sequence-based methods offer precision, while set-based methods provide efficiency for large datasets. Hybrid methods combine these advantages, and phonetic methods add the dimension of pronunciation, which is crucial for names and terms. Examples demonstrate show that no single algorithm is universal; the method choice of method is adapted to the data type and integration task. Integrating these approaches into data management systems has significantly improved data quality and accessibility across fields from bioinformatics to public administration. According to the authors, future development trends include applications with machine learning and artificial intelligence to enhance accuracy, scalability, and adaptability, particularly in global and heterogeneous data ecosystems. Ongoing innovation in these algorithms will remain critical for successful and reliable data integration in the digital age.

8. REFERENCE
8. REFERENCES

[1.] A. Stojanović, Ž. Kovačević, „Algoritmi i strukture podataka“, Tehničko veleučilište u Zagrebu, 2024, ISBN: 978-953-8444-12-8

[2.] A. Halevy, A. Rajaraman, and J. Ordille, „Data integration: the teenage years“, in Proceedings of the 32nd International Conference on Very Large Data Bases, 2006, pp. 9–16. [Online]. Available: https://dl.acm.org/doi/10.5555/1182635.1164130

[3.] P. Christen, „Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection“, Springer, 2012, doi: 10.1007/978-3-642-31164-2

[4.] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt, „Fast Pattern Matching in Strings“, SIAM Journal on Computing, vol. 6, no. 2, pp. 323–350, 1977, doi: 10.1137/0206024

[5.] R. S. Boyer and J. S. Moore, „A fast string searching algorithm“, Commun. ACM, vol. 20, no. 10, pp. 762–772, Oct. 1977, doi: 10.1145/359842.35985

[6.] C. Stippel, C. Pratellesi, B. Schwendinger, and R. Hoch, „A deep learning approach for event-driven duplicate detection“, in 2025 8th International Conference on Information and Computer Technologies (ICICT), Hawaii-Hilo, HI, USA, 2025, pp. 151–156. doi: 10.1109/ICICT64582.2025.00030

[7.] V. Naidu and A. Narayanan, „Using different substitution matrices in a string-matching technique for identifying viral polymorphic malware variants“, in 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 2903–2910. doi: 10.1109/CEC.2016.7744156

[8.] U. Brunner and K. Stockinger, „Entity Matching on Unstructured Data: An Active Learning Approach“, in 2019 6th Swiss Conference on Data Science (SDS), 2019, pp. 97–102. doi: 10.1109/SDS.2019.00006

AUTORI · *AUTHORS*



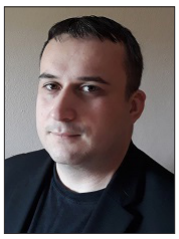
• **Aleksandar Stojanović** - Nepromijenjena biografija nalazi u časopisu P&D, Svezak. 11, Br. 4 iz 2023. godine.

Korespondencija · *Correspondence*
aleksandar.stojanovic@tvz.hr



• **Silvio Plehati** - Predavač je na Tehničkom veleučilištu u Zagrebu gdje sudjeluje u nastavi iz kolegija programskog inženjerstva, objektno-orijentiranog programiranja i naprednog programiranja. Na Tehničkom veleučilištu u Zagrebu 2003. godine završio je preddiplomski studij Informatike, 2010. godine diplomski studij Informatike, a na Grafičkom fakultetu Sveučilišta u Zagrebu završio je 2023. godine diplomski studij Grafičke tehnologije. Radi u struci od 2003. godine. Izbor u zvanje predavača stječe 2023. godine. Trenutno pohađa doktorski studij na Grafičkom fakultetu Sveučilišta u Zagrebu. Objavljuje sa koautorima stručne i znanstvene radove povezane sa grafičkim i programskim inženjerstvom, programiranjem i projektiranjem grafičkih zaštita. Područja interesa su: programiranje u C jezicima, grafički programski jezici, 2D i 3D grafika, 3D tisak, mikro (embedded) elektronika i senzori.

Korespondencija · *Correspondence*
silvio.plehati@tvz.hr



• **Željko Kovačević** - Biografija autora nalazi se na web stranicama https://nastava.tvz.hr/zkovacevic/.

Korespondencija · *Correspondence*
zeljko.kovacevic@tvz.hr