ISSN 2085-4552

Comparison of Cosine Similarity, Rabin-Karp, and Levenshtein Distance Algorithms for Plagiarism Detection in Document

Jasman Pardede[1], Agil Yudistira[2]
[1,2] Deparment of Informatics, Intitut Teknologi Nasional (Itenas), Bandung, Indonesia
[1]jasman@itenas.ac.id, [2]agil.yudistira@outlook.com

Accepted 13 June 2025
Approved 24 June 2025

***Abstract*— Prevention and detection of plagiarism are crucial. There are several algorithms that can be used to detect plagiarism in documents, including the Cosine Similarity, Rabin-Karp, Levenshtein Distance, Hamming Distance, Euclidean Distance, Edit Distance, Jaccard Similarity, Ratcliff/Obershelp, Winnowing, Brute Force, Boyer-Moore, and Knuth Morris Pratt algorithms. Based on the literature review from previous research, three best algorithms were identified: Cosine Similarity, Rabin-Karp, and Levenshtein Distance. However, there has been no study analyzing the comparison of these three algorithms. Therefore, this study will compare the performance of each algorithm and determine the best algorithm for plagiarism detection in documents based on similarity scores and execution time. The research objects use a sample of documents consisting of titles and abstracts from Indonesian-language informatics journals. Cosine Similarity is superior to others for plagiarism detection in documents, as it produces the highest average similarity score with a relatively fast execution time. The similarity values obtained using Cosine Similarity, Rabin-Karp 4-grams, and Levenshtein Distance were found to be 48.80%, 47.13%, and 20.61%, respectively. The average execution time of Cosine Similarity, Rabin-Karp with 4-grams, and Levenshtein Distance are 0.22 s, 0.45 s, and 39.15 s, respectively.**

***Index Terms*— *comparison; cosine similarity; rabin-karp; levenshtein distance; plagiarism; similarity*.**

I. INTRODUCTION

Plagiarism constitutes a substantial issue within academic and professional contexts, as it compromises the authenticity and credibility of scholarly endeavors. Plagiarism refers to the practice of taking, utilizing, or copying the work, concepts, or discoveries of another individual, either in full or in part, without sufficient attribution or appropriate referencing, thereby misrepresenting it as one's own intellectual product or achievement [1], [2]. Prevention and detection of plagiarism are crucial, especially in the academic world [2], [3]. With the ease of information exchange through the internet, academic community comprising students and lecturers can engage in copy-paste practices that may lead to plagiarism.

According to Regulation of the Minister of Education, Culture, Research, and Technology Number 39 of 2021 regarding Academic Integrity in Producing Scientific Works, article 10 paragraph (3), Plagiarism is the act of taking and rewriting part or all of someone else's work without using one's own language, even if the source is cited correctly [2],[5]. Furthermore, in the Tridharma of Higher Education, academic community members are also obligated to conduct research and produce scientific works [5]. Therefore, it is crucial to prevent and identify acts of plagiarism [2],[3].

TABLE I. LITERATURE REVIEW

Researcher	Year	Algorithms
Alvi, F. et al.	2017	Hamming Distance
Hartanto, A. D., et al.	2019	Rabin-Karp
Süzen, N., et al.	2020	Euclidean distance, Needleman-Wunsch Distance
Wahyuningsih, T., et. al.	2021	Cosine Similarity, Jaccard Similarity
Alobed, M., et al.	2021	Cosine Similarity, Jaccard Similarity
Nalawati, R.E., & Yuntari, A.D.	2021	Ratcliff/Obershelp
Astuti, Y., & Wulandari, I.,	2022	Rabin-Karp
Hidayat, W., et al.	2022	Rabin-Karp
Al-Hagree, S., & Al-Gaphari, G.	2022	Levenshtein Distance
Nandurkar, D. A.,et al.	2023	Levenshtein Distance
Amalia, E.L., et al.	2023	Winnowing
Alfat, L., et al.	2023	Knuth Morris Pratt
Barut, Z. & Altuntas, V.	2023	Boyer-Moore
Setu, D.M., et al.	2025	Cosine Similarity
Madhan, N., et al.	2025	Manhattan Distance

One way to detect plagiarism is by using the concept of string matching to calculate the similarity between documents [1],[3],[6],[7]. To achieve the best results, the use of the best algorithm is essential for detecting plagiarism in documents. There are several algorithms that can be used for plagiarism detection in document, including Cosine Similarity [8]-[10], Jaccard Similarity [8],[9], Rabin-Karp [11]-[13], Levenshtein Distance [14],[15], Hamming Distance [16], Euclidean distance [17], Winnowing [18], Knuth Morris Pratt [19], Ratcliff/Obershelp [20], Needleman-Wunsch Distance [17], Boyer-Moore [21], Manhattan Distance [22], and others. Several previous studies have been conducted using those algorithms. A literature review of research relevant to those algorithms is presented in Table I. Among those methods, three were selected based on their consistent superiority in various aspects of plagiarism detection, as explained in the literature review.

Cosine Similarity with pre-processing is superior to Jaccard Similarity in measuring similarity because it produces the highest correlation values [8]. Cosine Similarity has a better accuracy rate compared to the Euclidean Distance algorithm for calculating similarity [9]. Although the Rabin-Karp algorithm exhibits limitations in accurately distinguishing between similar words, it demonstrates superior performance in plagiarism detection when compared to the Brute Force and Boyer-Moore algorithms [23]. Specifically, while the Brute Force algorithm excels in single-pattern searches, it remains less effective for multiple-pattern searches. Additionally, the Boyer-Moore algorithm performs efficiently by shifting the last two characters but is less effective with earlier shifts. In contrast, the Rabin-Karp algorithm effectively addresses these limitations [24].

The Rabin-Karp algorithm can be used for searching long pattern strings but has a longer execution time compared to Ratcliff/Obershelp. Similarity values are highly influenced by the sentence structure in Rabin-Karp, whereas it has no impact at all on Ratcliff/Obershelp [11]. From the testing of Rabin-Karp with ***k***-grams ranging from 2 to 10, the one with the highest accuracy is ***3***-gram [13]. The Rabin-Karp algorithm is better than the Winnowing for detecting plagiarism because it produces higher similarity scores. Therefore, for this research, the Rabin-Karp algorithm is proposed for detecting plagiarism.

The Levenshtein Distance algorithm, which checks each character one by one, produces perfect results on simple short sentences but does not perform well on long documents with irregular positions [25]. The Levenshtein Distance algorithm is better than the Knuth Morris Pratt algorithm because it has higher speed and accuracy and can minimize errors when searching for data [26]. The Levenshtein Distance algorithm is superior and more efficient compared to Ratcliff/Obershelp for detecting plagiarism [27].

Based on the literature review, there are three best algorithms: Cosine Similarity, Rabin-Karp, and Levenshtein Distance. These algorithms have their respective strengths and weaknesses, but there has been no research analyzing the comparison of these three algorithms for detecting plagiarism in documents. Therefore, a comparative analysis among these algorithms is needed to determine the best algorithm by measuring the level of similarity for detecting plagiarism in documents and execution time as a parameter to assess the speed of each algorithm.

Therefore, the purpose of this research is to compare the similarity scores and execution times to know the performance of each algorithm and determine the best algorithm for plagiarism detection in documents based on similarity scores and execution times. This study is a novelty compared to previous research, as there has been no study comparing the Cosine Similarity, Rabin-Karp, and Levenshtein Distance algorithms for plagiarism detection in documents.

II. METHODOLOGY

The research method used is a comparative analysis, where this study will compare performance of Cosine Similarity, Rabin-Karp, and Levenshtein Distance algorithms for detecting plagiarism in documents. The research begins with data collections, followed by comparison document collections processing. After that, testing is conducted using the Cosine Similarity, Rabin-Karp, and Levenshtein Distance algorithms. Once the testing is completed, an analysis will be performed to determine the best algorithm based on similarity scores and execution time. The block diagram of this research can be seen in Fig. 1.

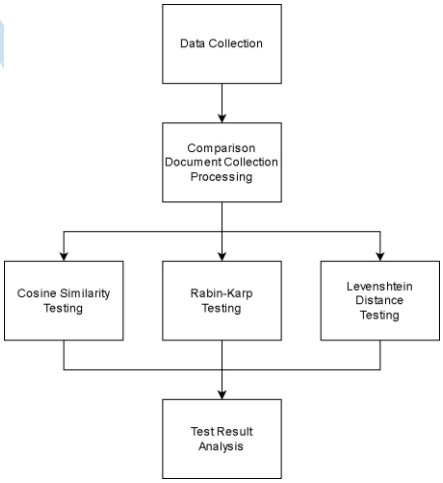

Fig. 1. Block diagram system

A. Data Collections Technique

The data collections technique used for testing in this research is by sampling documents consisting of titles and abstracts from Indonesian-language journals. The chosen journals are in the field of informatics, particularly those with high citation numbers and

publication years ranging from 2018 to 2023. This is to obtain data that is relevant to the research objectives and of good quality. The process of collecting sample documents is conducted through Google Scholar using the keywords {"Information Technology," "Artificial Intelligence," "Machine Learning," "K-Means Clustering," and "Naive Bayes"}. Then, the titles and abstracts from each journal will be extracted and saved in *.txt* file format to facilitate data pre-processing.

The total sample documents in this research amount to 440 documents. The 440 documents were then divided into two groups: 400 documents were used as the comparison document collections (reference corpus) without specific labeling, which serve as the base dataset for similarity comparison, while 40 documents were designated as test documents. The test documents were selected randomly from the collected sample to evaluate the performance of the plagiarism detection algorithms. Since all documents are from the same domain and collected via the same keywords, this setup simulates real-world scenarios of document similarity detection without explicit labeling of plagiarism cases.

B. Data Pre-processing

Pre-processing is performed as part of data cleansing to simplify and standardize the text, allowing it to be processed more effectively in the main process [28]. The pre-processing conducted in this research consists of only two stages, namely Case Folding and Filtering.

The case folding stage is performed to eliminate differences in letter case, thus converting the entire text to lowercase. The filtering stage is carried out to enhance accuracy and speed by focusing on more significant words. In this study, the filtering process does not remove stop words because, based on the adopted definition of plagiarism, paraphrasing is considered important. Therefore, only punctuation marks and special characters will be removed. The flowchart of the data pre-processing can be seen in Fig. 2.

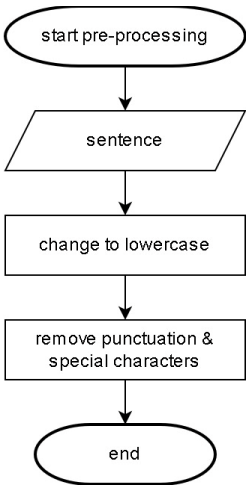

Fig. 2. Flowchart of data pre-processing

C. Comparison Document Collections Processing

In the testing process, the test documents will be compared with the collections of comparison documents, while the text content in each document must undergo parsing or text segmentation based on sentences and pre-processing. The purpose of processing the collections of comparison documents is to avoid repetitive parsing and pre-processing of the document collections every time a test is conducted.

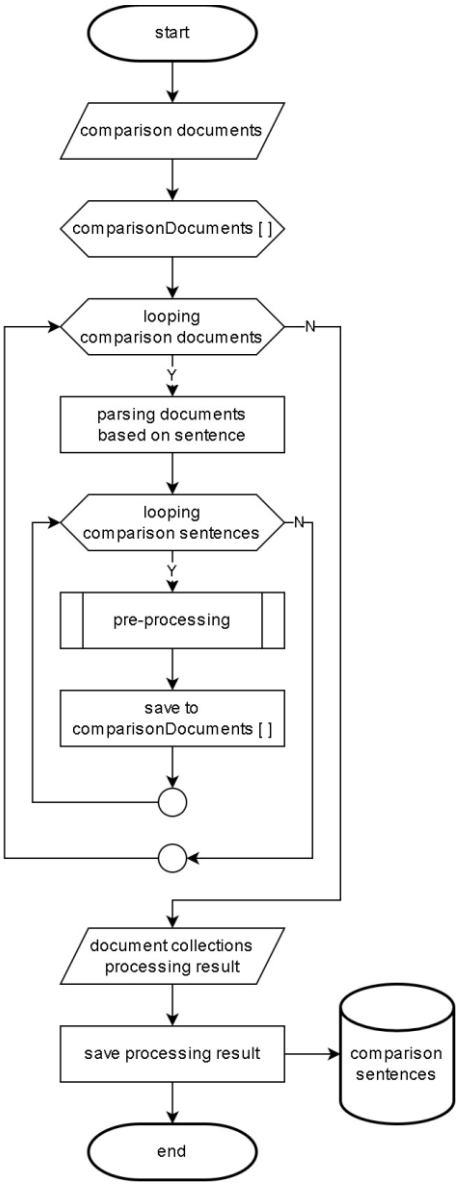

Fig. 3. Flowchart of comparison document collections processing

The system will read the folder containing the collections of comparison documents. Each document will undergo parsing to separate the text based on sentences, and each sentence will undergo pre-processing. Then, the processed sentences will be stored in the variable "comparisonDocuments" with the document index and its corresponding sentence. Once all the sentences in the first document are processed, the system will proceed to the next document until all documents in the folder are processed. After that, the

results of the comparison document collections processing will be stored in the comparison sentence database, allowing access for each testing session. The flowchart of the comparison document collections processing can be seen in Fig. 3.

D. Algorithm Testing Technique

The purpose of this testing is to obtain the similarity scores for each test sentence with all the sentences in the comparison document collections. Then, the highest similarity score for each test sentence will be determined to identify which comparison document sentence it is most similar to and at which sentence position. The average similarity score for each test sentence will be calculated to determine the similarity value between the test document and the entire collections of comparison documents.

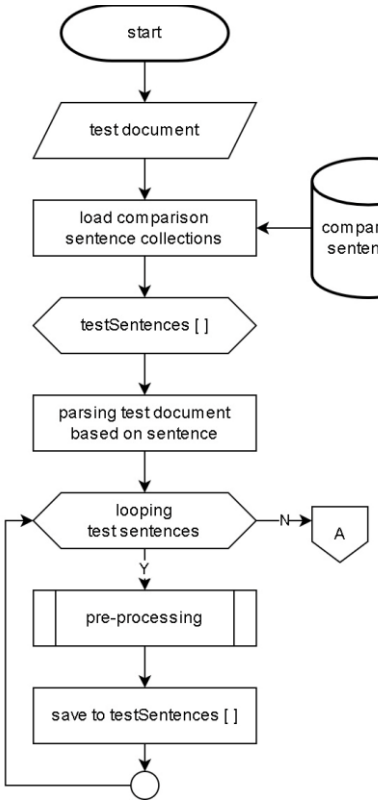

Fig. 4. Flowchart of algorithm testing

The similarity scores between the test documents generated by each algorithm will be compared to determine which algorithm produces the highest similarity score. Additionally, the execution time of each algorithm will also be measured to determine which algorithm is the fastest in its usage. The testing phase will be conducted using 40 test documents. Each algorithm will be applied to the same test documents. Analyze each of the results. Comparing the performance of each algorithm in detecting plagiarism.

In this study, testing will be conducted one by one in sequential order for each algorithm, namely: (1) Cosine Similarity, (2) Levenshtein Distance, (3) Rabin-Karp 2-gram, (4) Rabin-Karp 4-gram, and (5) Rabin-Karp 6-gram. The selection of these *k*-gram values is based on previous studies that have also used similar *k*-gram values [13], to observe the impact of *k*-gram size on Rabin-Karp algorithm performance. Therefore, one document will not be tested using all algorithms simultaneously. This means that one test document will be repeatedly used in five different tests. As a result, out of 40 test documents, a total of 200 tests will be conducted. The testing flowcharts can be seen in Fig. 4 to Fig. 6.

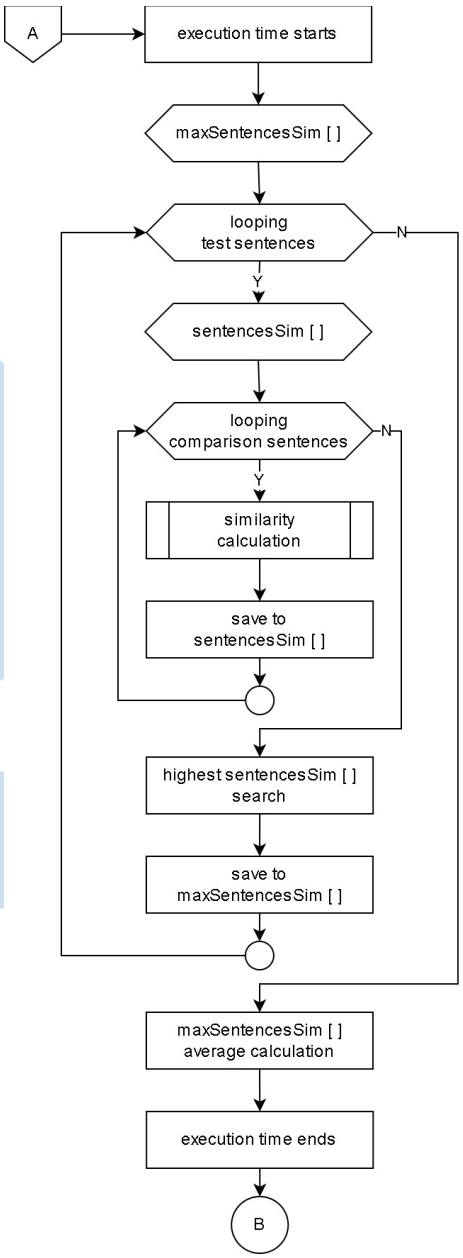

Fig. 5. Flowchart of algorithm testing (continued)

The testing begins by uploading the test documents. First, the system will load the comparison sentence collections from the database. Then, the test document will be parsed based on sentences, and each test sentence will undergo pre-processing and will be saved in the variable "testSentences". This is done to facilitate the execution time calculation, as the time will be

measured only during the algorithm testing process. Once all test sentences have completed pre-processing, the next stage will proceed to the main testing process. The flowchart of the main testing process can be seen in Fig. 5.

At this stage, the execution time calculation begins. Each test sentence will be compared with all the comparison sentences. The first test sentence will be compared with the first comparison sentence to calculate the similarity. As the testing is conducted one by one for each algorithm, at this stage of similarity calculation, one algorithm function will be called based on the selected algorithm testing.

The similarity calculation results will be stored in the "sentencesSimilarity" variable along with the index of the test sentence, the comparison document, and the comparison sentence. This stage will be repeated continuously until the first test sentence is compared with all the comparison sentences. After that, the highest similarity of the first test sentence will be determined and saved in the variable "maxSentencesSimilarity". This is intended to determine which test sentence has the highest similarity with which comparison sentence in which comparison document. If the first test sentence has been compared with all the comparison sentences and the index with the highest similarity value is obtained, the process will be repeated for the second test sentence until all the test sentences are compared with all the comparison sentences.

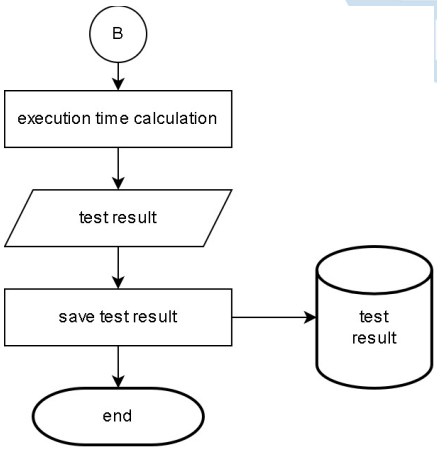

Fig. 6. Flowchart of algorithm testing (continued)

Afterward, the similarity score for the test document will be determined by calculating the average of the highest similarity scores from all test sentences. This stage is the end of the main testing, thus the execution time calculation is completed at this stage. The process proceeds to the final stage, which can be seen in Fig. 6.

The final stage is the execution time calculation, which is obtained by subtracting the end execution time from the start execution time. After that, all test results will be saved to the test result database. These tests are conducted on each algorithm.

E. Cosine Similarity Algorithm

The working principle of the Cosine Similarity algorithm is to measure the proximity between two vectors by calculating their dot product and then dividing it by the Euclidean distance between the two vectors for normalization. Equation (1) represents the formula for Cosine Similarity used to calculate the similarity score [9],[29].

$$S_{(A,B)} = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2}\sqrt{\sum_{i=1}^{n} {B_i}^2}} \times 100 \quad (1)$$

S represents the similarity score, ***A*** and ***B*** are vector representations of two documents being compared, where each vector represents the term frequency (TF) values of words in the documents, ***n*** is the number of vectors, and the ***i*** is the vector order.

F. Rabin-Karp Algorithm

The working principle of the Rabin-Karp algorithm is string searching based on patterns (substrings) and comparing the hash values of each pattern [23]. First, the text will be divided into grams based on the value of k to obtain substrings. Then, the substrings will be converted into hash values, and the hash values between two texts will be compared. If both hash values are the same, the comparison will be done once again on the characters of the substring to ensure that the hash value represents the same substring. If they are not the same, the substring will be shifted to the right [13]. Afterward, the similarity score is calculated using the Dice Similarity Coefficient equation. Equation (2) represents the formula used in the Rabin-Karp algorithm to calculate the hash value.

$$H_{(p)} = \sum_{i=1}^{k} ascii(p_i)\ b^{(k-1)} \quad (2)$$

H represents the hash value, ***p*** is the pattern, ***k*** is the value of ***k***-gram, ***b*** is the base or radix value (a prime number), and ***i*** is the character order in the pattern. Equation (3) is the Dice Similarity Coefficient formula used for calculating the similarity in the Rabin-Karp algorithm [12].

$$S_{(A,B)} = \frac{2 \times |H_A \cap H_B|}{|H_A| + |H_B|} \quad (3)$$

S represents the similarity score, ***A*** and ***B*** are the texts being compared, and ***H*** is the hash. Example calculations using the Rabin-Karp algorithm can be found in [11],[23].

G. Levenshtein Distance Algorithm

The working principle of the Levenshtein Distance algorithm is by creating a matrix to calculate the number of operations required to change one character using addition, deletion, or substitution operations. This calculation is done by comparing each character [14].

However, based on the research conducted [18], distance calculations can be done by directly calculating the word differences without creating a matrix, making the steps in the distance calculation simpler and can improve execution time. Equation (4)

represents the formula used for calculating the Levenshtein distance value [15].

$$D_{(A,B)} = \sum_{i=1}^{n} d(A_i, B_i) \quad (4)$$

D is the Levenshtein distance, ***A*** and ***B*** are the texts being compared, ***d*** is the comparison value (if equal = 0, if different = 1), ***n*** is the maximum number of texts between ***A*** and ***B***, and the ***i*** is the word order in the text. Equation (5) represents the Levenshtein Distance formula for calculating the similarity.

$$S_{(A,B)} = \left(1 - \frac{D_{(A,B)}}{n}\right) \times 100 \quad (5)$$

S is the similarity value, ***D*** is the Levenshtein distance, ***A*** and ***B*** are the texts being compared, and ***n*** is the maximum number of texts between ***A*** and ***B***. Examples of calculations using the Levenshtein Distance algorithm can be found in [18].

H. Test Result Analysis Technique

The data analysis technique used in this study is a comparative analysis, where the test results of each algorithm will be presented in the same table, allowing the performance of each algorithm to be directly compared. Additionally, the average similarity values and execution times of each algorithm will be presented in the form of bar charts.

III. Result And Discussion

A. Experimental Result

After comparison document collections processing is completed, out of the 400 documents used, a total of 3,471 comparison sentences were obtained. Therefore, in this study, each test sentence will be compared and its similarity calculated with the 3,471 comparison sentences. Table II shows the test results on the same 10 test documents using Cosine Similarity, Levenshtein Distance, and Rabin-Karp algorithms

TABLE II. Experimental Result

No	Document	Cosine Similarity		Levenshtein Distance		Rabin-Karp		
		Similarity (%)	Time (s)	Similarity (%)	Time (s)	k	Similarity (%)	Time (s)
1	2020-403 TI	49.63	0.50	15.29	0.23	2 4 6	89.71 45.69 29.28	24.74 51.95 62.37
2	2021-10 ML	53.81	0.36	22.88	0.17	2 4 6	89.77 52.12 33.16	19.00 40.07 48.06
3	2021-14 KM	47.09	0.23	14.94	0.12	2 4 6	89.00 47.33 29.52	13.07 26.15 32.10
4	2021-84 NB	53.01	0.52	18.76	0.22	2 4 6	86.14 44.63 30.72	22.66 41.39 49.02
5	2021-86 TI	41.86	0.40	17.82	0.18	2 4 6	84.90 42.28 27.04	17.47 31.98 38.40
6	2021-105 NB	45.77	0.54	18.34	0.26	2 4 6	84.46 41.80 27.40	24.42 44.66 52.56
7	2021-145 NB	53.18	0.46	25.23	0.22	2 4 6	84.06 46.41 31.79	19.40 35.35 42.10
8	2021-155 NB	48.89	0.50	21.47	0.24	2 4 6	83.82 45.07 28.40	20.72 38.51 45.63
9	2021-170 NB	48.30	0.53	26.33	0.24	2 4 6	83.36 48.50 33.97	22.41 40.35 48.68
10	2021-198 TI	38.53	0.61	10.61	0.29	2 4 6	82.96 39.44 20.03	29.48 56.42 68.06

The test result using the Cosine Similarity algorithm resulted in an average similarity of 48.80% and an execution time of 0.45 seconds. The test result using the Levenshtein Distance algorithm resulted in an average similarity of 20.61% and an execution time of 0.22 seconds. The test result using the Rabin-Karp algorithm with 2-grams resulted in an average similarity of 83.76% and an execution time of 19.83 seconds. With 4-grams, the average similarity was 45.96% and the execution time was 37.71 seconds. Meanwhile, with 6-grams, the average similarity was 30.76% and the execution time was 45.70 seconds. The comparison of

average similarity values and execution times of each algorithm is presented in Table III.

TABLE III. COMPARISON OF SIMILARITY AND EXECUTION TIME

Cosine Similarity	Levenshtein Distance	Rabin-Karp		
		2	4	6
48.80 %	20.61 %	86.21 %	47.13 %	31.40 %
0.45 s	0.22 s	20.74 s	39.15 s	47.52 s

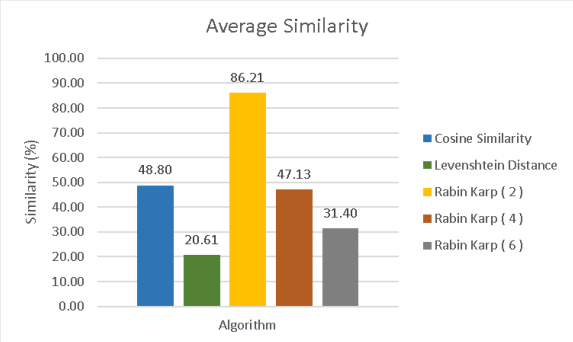

Fig. 7. Comparison diagram of similarity

To facilitate the analysis process, the test results are also presented in the form of diagrams. The diagram comparing the average similarity values can be seen in Fig. 7, and the diagram comparing the average execution time can be seen in Fig. 8. Based on the test results conducted on the same test document, it shows that the Rabin-Karp method with 2-grams has a greater similarity. This is because the test document used is an abstract document that has a word length of between 150 and 350 words. When compared to the time requirements, Rabin-Karp with 2-grams is faster than 3-grams and 4-grams. This is because Rabin-Karp 2-grams only has 2 characters, resulting in the time required to calculate each hash much faster than 3-grams and 4-grams.

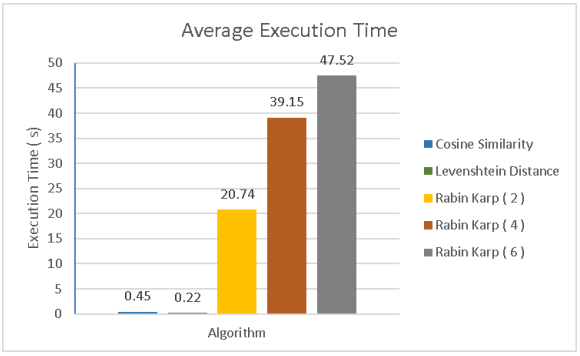

Fig. 8. Comparison diagram of execution time

Cosine Similarity demonstrated the second-best performance following Rabin-Karp 2-gram, with lower computational time. Although the time required for Cosine Similarity is less than that of Rabin-Karp 2-gram—attributable to Cosine Similarity's use of vector space models for computing document similarity rather than substring rolling hash—the time requirement for Cosine Similarity is greater than that of Levenshtein Distance. This is due to the fact that Levenshtein Distance compares two relatively short strings, whereas Cosine Similarity involves comparing two high-dimensional vectors representing entire documents.

B. Discussion

In Cosine Similarity algorithm, the text is represented as a vector, and the difference between words is represented as angles. This approach only considers the frequency of same words, without taking into account the order of these words. Therefore, the similarity score is only influenced by the frequency of shared words, while the word arrangement does not affect the similarity score at all.

In the Rabin-Karp algorithm, this approach involves comparing patterns (substrings). The pattern will be formed based on the value of ***k***-gram, which is influenced by the word order. Therefore, the similarity score is influenced by ***k***-gram and word arrangement. The smaller the value of ***k***-gram, the more patterns will be formed, potentially resulting in higher similarity.

This study adds to the previous research [11] that besides sentence arrangement, the similarity score in the Rabin-Karp algorithm is also influenced by the word arrangement. Responding to the study [12], it is observed that the same sentence with different word order can yield different similarity scores. Thus, if the order is altered during comparison, the similarity score may also change.

In the Levenshtein Distance algorithm, the distance calculation is performed by comparing characters. However, in this study, it has been successfully implemented by computing the distance based on word comparisons. This research addresses the issues in the previous study [26] since the Levenshtein Distance algorithm can perform well on long documents with unstructured sentence positions. This approach compares words at the same position, so even if two sentences have all the same words, the resulting similarity score will be 0% if all their positions are different. Therefore, the similarity score is highly influenced by word order. The more same words are found in the same order, the higher the resulting similarity score will be.

In each algorithm, the execution time is influenced by the number of steps each algorithm has to perform. The more steps that need to be performed, the longer the execution time required. In the Rabin-Karp algorithm, the execution time is also influenced by the value of k-gram, the larger the value of ***k***-gram, the more steps need to be performed.

In the Levenshtein Distance algorithm, the distance calculation method used [25], which directly calculates the word differences using Equation (4), can improve the execution time compared to creating a matrix. This

also makes the Levenshtein Distance algorithm faster than Cosine Similarity.

The test results indicate that the Rabin-Karp algorithm with 2-grams produces the highest similarity score with an average of 83.76%. However, based on the research conducted by [11], [13] on testing the Rabin-Karp algorithm with ***k***-grams from 1 to 10, it was found that ***k***-gram 3 achieved the highest accuracy because it produced very close similarity scores on each test data compared to other ***k***-grams. Hence, 4-gram is more suitable for plagiarism detection. Therefore, in this study, to compare the performance of the Rabin-Karp algorithm with other algorithms, the data used will be the test results with 4-gram. Therefore, the Cosine Similarity algorithm is superior in detecting similarity with an average similarity score of 48.80%, followed by the Rabin-Karp Algorithm with 4-grams at an average of 47.13%, and the Levenshtein Distance algorithm with an average of 20.61%. In terms of execution time, the Levenshtein Distance algorithm is superior as it shows the fastest performance with an average execution time of 0.22 seconds, followed by the Cosine Similarity algorithm with an average of 0.45 seconds, and the Rabin-Karp algorithm with 4-grams with an average of 39.15 seconds.

The same test sentence can yield the highest similarity score with different sentences in the comparison document, depending on the algorithm used in the testing. Thus, out of the 40 test documents, the Cosine Similarity algorithm produces the highest average similarity score in 26 documents, while the Rabin-Karp algorithm with 4-grams yields the highest average similarity score in 14 documents. Conversely, the Levenshtein Distance algorithm consistently produces lower average similarity scores compared to the other algorithms in all test documents.

Based on the performance of each algorithm, the selection of an algorithm for detecting plagiarism in documents can be based on the specific needs and objectives of the application. The Cosine Similarity algorithm is superior if the main priority is the similarity level of documents, without considering the word arrangement, and with relatively fast execution time. The Levenshtein Distance algorithm may be a more suitable choice if the application requires faster execution time while considering the word order. Meanwhile, the Rabin-Karp algorithm with 4-grams could be a better option for applications that emphasize high similarity levels while considering the word arrangement, despite longer execution time considerations.

IV. CONCLUSIONS

Based on the experimental results conducted, it was obtained that the best performance for detecting plagiarism was Cosine Similarity, followed by Rabin-Karp 4-gram and Levenshtein Distance. The plagiarism detection performance of Cosine Similarity, Rabin-Karp 4-gram, and Levenshtein Distance are 48.80%, 47.13%, and 20.61%, respectively. In contrast, the best time requirements are Levenshtein Distance, Cosine Similarity, and Rabin-Karp 4-gram, with time requirements of 0.22s, 0.45s, and 39.15s, respectively.

Plagiarism detection performance can be affected by the length of the words in the document, the arrangement of the words, and the number of the same words tested. Cosine Similarity in calculating the similarity between documents uses a vector space, while Rabin-Karp uses a substring rolling hash influenced by *k*-grams. Levenshtein compares two strings with a relatively short length while Cosine Similarity compares two vectors with a length that represents the entire document.

The selection of an algorithm for detecting plagiarism in documents should be guided by the specific requirements and objectives of the application. The Cosine Similarity algorithm is preferable when the primary focus is on the overall similarity between documents, regardless of word order, and when a relatively fast execution time is desired. In contrast, the Levenshtein Distance algorithm may be more appropriate for applications that prioritize both execution speed and consideration of word order. Meanwhile, the Rabin-Karp algorithm using may be the optimal choice for applications that emphasize achieving high similarity detection with attention to word order, even though it generally requires longer processing time.

REFERENCES

[1] Chowdhury, H.A., & Bhattacharyya, D. K. (2018). Plagiarism: Taxonomy, Tools and Detection Techniques. arXiv:1801.06323, Information Retrieval, doi: 10.48550/arXiv.1801.06323

[2] Velmurugan, V.S. (2024). Types and Definitions of Plagiarism: An Overview. Global Research Journal of Social Sciences and Management, vol. 2 (1), ISSN: 2583-858X, doi: 10.55306/GRJSSM.2024.2102.

[3] Bhavana, M., Rao, K. S., Koduru, G. K., Vatsal, K. V. K., Parvathi, S., & Sravani, M. (2024). Plagiarism Detection and Similarity Checking Program using Machine Learning and String Matching Algorithm. 2024 9th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2024, pp. 2032-2036, doi: 10.1109/ICCES63552.2024.10859471

[4] Perkins, M., Gezgin, U.B. & Roe, J. (2020). Reducing plagiarism through academic misconduct education. Int J Educ Integr, vol. 16 (3), doi: 10.1007/s40979-020-00052-8

[5] Republic of Indonesia, "Regulation of the Minister of Education, Culture, Research, and Technology of the Republic of Indonesia Number 39 of 2021 concerning Academic Integrity in Producing Scientific Works." Jakarta, 2021.

[6] Kumar, P., Gupta, M. K., Rao, C. R. S., Bhavsingh, M., & Srilakshmi, M. (2023). A Comparative Analysis of Collaborative Filtering Similarity Measurements for Recommendation Systems. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3s), 184–192, doi: 10.17762/ijritcc.v11i3s.6180.

[7] Swathi, M., & Selvi, C. (2022). An Improved Similarity Measure Based on Collaborative Filtering for Sparsity Problem in Recommender Systems. In: Bhateja, V., Khin Wee, L., Lin, J.CW., Satapathy, S.C., Rajesh, T.M. (eds) Data Engineering and Intelligent Computing. Lecture Notes in Networks and Systems, vol 446. Springer, Singapore, doi: 10.1007/978-981-19-1559-8_5.

[8] Wahyuningsih, T., Henderi, H., & Winarno, W. (2021). Text Mining an Automatic Short Answer Grading (ASAG), Comparison of Three Methods of Cosine Similarity, Jaccard Similarity and Dice's Coefficient. Journal of Applied Data Sciences, 2(2), doi: 10.47738/jads.v2i2.31.

[9] Alobed, M., Altrad, A. M. M., & Bakar, Z. B. A. (2021). A Comparative Analysis of Euclidean, Jaccard and Cosine Similarity Measure and Arabic Wordnet for Automated Arabic Essay Scoring," 2021 Fifth International Conference on Information Retrieval and Knowledge Management (CAMP), Kuala Lumpur, Malaysia, 2021, pp. 70-74, doi: 10.1109/CAMP51653.2021.9498119.

[10] Setu, D.M., Islam, T., Erfan, M., Dey, S.K., Asif, M.R.A., & Samsuddoha, M. (2025). A comprehensive strategy for identifying plagiarism in academic submissions. J. Umm Al-Qura Univ. Eng. Archit. 16, 310–325, doi: 10.1007/s43995-025-00108-1.

[11] Hartanto, A. D., Syaputra, A., & Pristyanto, Y. (2019). Best Parameter Selection Of Rabin-Karp Algorithm In Detecting Document Similarity. 2019 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2019, pp. 457-461, doi: 10.1109/ICOIACT46704.2019.8938458.

[12] Astuti, Y., & Wulandari, I., . (2022). An arrangement of the number of K-grams in the performance of Rabin Karp algorithm in text adjustment. Indonesian Journal of Electrical Engineering and Computer Science (IJEECS), p-ISSN: 2502-4752, vol. 26 (3), pp. 1388-1394, doi: 10.11591/ijeecs.v26.i3.pp1388-1394.

[13] Hidayat, W., Utami, E., & Sunyoto, A. (2022). Selection of the Best K-Gram Value on Modified Rabin Karp Algorithm. IJCCS (Indonesian Journal of Computing and Cybernetics Systems), vol. 16 (1), ISSN: 2460-7258, pp. 11-22, doi: 10.22146/ijccs.63686.

[14] Al-Hagree, S., & Al-Gaphari, G. (2022). Arabic Sentiment Analysis on Mobile Applications Using Levenshtein Distance Algorithm and Naive Bayes. 2022 2nd International Conference on Emerging Smart Technologies and Applications (eSmarTA), Ibb, Yemen, 2022, pp. 1-6, doi: 10.1109/eSmarTA56775.2022.9935492.

[15] Nandurkar, D. A., Ujjainkar, P., Miglani, B., & Kanojiya, A. (2023). Plagiarism Checker & Link Advisor using concepts of Levenshtein Distance Algorithm with Google Query Search - An Approach. 2023 1st International Conference on Advanced Innovations in Smart Cities (ICAISC), Jeddah, Saudi Arabia, 2023, pp. 1-6, doi: 10.1109/ICAISC56366.2023.10085404.

[16] Alvi, F., Stevenson, M., & Clough, P. (2017). Plagiarism Detection in Texts Obfuscated with Homoglyphs. ECIR 2017: Advances in Information Retrieval. 39th European Conference on Information Retrieval, 08-13 Apr 2017, Aberdeen, Scotland. Lecture Notes in Computer Science. Springer, Cham , pp. 669-675. ISBN 978-3-319-56608-5.

[17] Süzen, N., Gorban, A.N., Levesley, J., & Mirkes, E.M. (2020). Automatic short answer grading and feedback using text mining methods. Procedia Computer Science, vol 169, 2020, pp. 726-743,ISSN 1877-0509, doi: 10.1016/j.procs.2020.02.171

[18] Amalia, E.L., Lestari, V. A., Wijayaningrum, V.N., & Ridla, A. A. (2023). Automatic essay assessment in e-learning using winnowing algorithm. Indonesian Journal of Electrical Engineering and Computer Science, vol. 29 (1), pp. 572-582, ISSN: 2502-4752, doi: 10.11591/ijeecs.v29.i1.pp572-582.

[19] Alfat, L., Faisal, F. M., Negara, K. P. S., Munggaran, M. R., & Ihsan. (2023). Implementing Knuth-Morris-Pratt Algorithm in Detecting The Plagiarism of Document. 2023 10th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, Indonesia, 2023, pp. 54-58, doi: 10.1109/ICITACEE58587.2023.10276453.

[20] Nalawati, R.E., & Yuntari, A.D. (2021). Ratcliff/Obershelp Algorithm as An Automatic Assessment on E-Learning. 2021 4th International Conference of Computer and Informatics Engineering (IC2IE), Depok, Indonesia, 2021, pp. 244-248, doi: 10.1109/IC2IE53219.2021.9649217.

[21] Barut, Z. & Altuntas, V. (2023). Applied Comparison of String Matching Algorithms. Gaziosmanpasa Journal of Scientific Research, vol. 12 (1), ISSN: 2146-8168.

[22] Madhan, N., Dheva Rajan, S., Jain, M. (2025). Directing Natural Language Processing Text Similarity Challenges in Social Media with AI Techniques. In: Goar, V., Kuri, M., Kumar, R., Senjyu, T. (eds) Advances in Information Communication Technology and Computing. AICTC 2024. Lecture Notes in Networks and Systems, vol 1075. Springer, Singapore, doi: 10.1007/978-981-97-6106-7_26.

[23] Leonardo, B. & Hansun, S. (2017). Text Documents Plagiarism Detection using Rabin-Karp and Jaro-Winkler Distance Algorithms. Indonesian Journal of Electrical Engineering and Computer Science, vol. 5 (2), pp. 462- 471, doi: 10.11591/ijeecs.v5.i2.pp462-471

[24] Khan, Z.A. & Pateriya, R.K. (2012). Multiple Pattern String Matching Methodologies: A Comparative Analysis. International Journal of Scientific and Research Publications, vol. 2 (7), ISSN 2250-3153.

[25] Coates, P. & Breitinger, F. (2023). Identifying document similarity using a fast estimation of the Levenshtein Distance based on compression and signatures. arXiv: Information Retrieval, doi: 10.48550/arXiv.2307.11496.

[26] Rao, P.J., Rao, K.N., Gokuruboyina, S., & Neeraja, K.N. (2024). An Efficient Methodology for Identifying the Similarity Between Languages with Levenshtein Distance. In: Kumar, A., Mozar, S. (eds) Proceedings of the 6th International Conference on Communications and Cyber Physical Engineering. ICCCE 2024. Lecture Notes in Electrical Engineering, vol. 1096. Springer, Singapore, doi: 10.1007/978-981-99-7137-4_15.

[27] Kalbaliyev, E., Rustamov, S. (2021). Text Similarity Detection Using Machine Learning Algorithms with Character-Based Similarity Measures. In: Biele, C., Kacprzyk, J., Owsiński, J.W., Romanowski, A., Sikorski, M. (eds) Digital Interaction and Machine Intelligence. MIDI 2020. Advances in Intelligent Systems and Computing, vol 1376. Springer, Cham, doi: 10.1007/978-3-030-74728-2_2.

[28] Finansyah, A. Y. W., Afiahayati, & Sutanto, V.M. (2022). Performance Comparison of Similarity Measure Algorithm as Data Preprocessing Stage: Text Normalization in Bahasa Indonesia. Scientific Journal of Informatics, vol. 9 (1), ISSN 2407-7658, pp. 1-7, doi: 10.15294/sji.v9i1.30052.

[29] Usino, W., Prabuwono, A.S., Allehaibi, K.H.S., Bramantoro, A., Hasniaty A, & Amaldi, W. (2019). Document Similarity Detection using K-Means and Cosine Distance. (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 10 (2), pp. 165-170, doi: 10.14569/IJACSA.2019.0100222