A Hybrid Length-Based Pattern Matching Algorithm for Text Searching

Victor Cornejo-Aparicio, Cesar Cuarite-Silva, Antoni Benavente-Mayta, Karim Guevara
Universidad Nacional De San Agustín De Arequipa, Arequipa, Perú

***Abstract*—This paper presents a hybrid algorithm for pattern matching in text, which combines word length preprocessing with the Knuth-Morris-Pratt (KMP) algorithm. Its performance was evaluated against KMP and Boyer-Moore (BM) in two scenarios: synthetic texts and real-world texts. In the former, classical algorithms proved more efficient due to the uniform structure of the data. However, in real-world texts, the hybrid algorithm significantly reduced search times, thanks to its ability to filter matches by length patterns before performing character-by-character comparisons. The algorithm also demonstrated flexibility in recognizing patterns with different delimiters. Among its limitations is the difficulty in detecting substrings within longer words. As future work, the incorporation of partial matching techniques and the adaptation of the approach to multilingual environments and machine learning systems are proposed. The dataset used is provided to encourage reproducibility.**

Keywords*—*Knuth-Morris-Pratt; Boyer-Moore; text search; hybrid algorithm, preprocessing; word-length patterns; test text for experiments

I. Introduction

In the field of text processing and pattern matching, algorithmic efficiency is a crucial factor for numerous applications, particularly in the search for substrings within large volumes of data. This need becomes even more relevant in areas such as data mining, information retrieval, and text analysis, where fast, accurate, and scalable search mechanisms are required.

Classical algorithms such as Knuth-Morris-Pratt (KMP) and Boyer-Moore (BM) have been widely used due to their proven mathematical efficiency. However, these algorithms exhibit limitations in certain scenarios, such as datasets with high repetitiveness or large and unbalanced alphabets. Under these conditions, their performance tends to degrade, increasing computational overhead and reducing the overall efficiency of the system.

In response to these challenges, the objective of this work is to reduce the time required to locate strings within large texts through an alternative approach that leverages the internal structure of natural language. To this end, we propose a novel substring search method based on word length patterns, which enables the use of a more compact and structured representation of the original text as the basis for searches.

The proposed solution introduces a hybrid algorithm that combines a preprocessing step—in which the sequence of word lengths is extracted—with the traditional KMP algorithm. In this way, a preliminary search is conducted on a reduced representation of the text (based on word lengths), and full validation is performed on the original content only when potential matches are detected. This strategy accelerates the pattern location process and enhances the overall performance of the algorithm.

Thus, the present research addresses a practical need in the processing of large volumes of text by proposing a method that exploits the structural properties of language to improve the speed and efficiency of searches.

The remainder of this paper is organized as follows: Section II presents related work. Section III provides a review of the relevant literature. Section IV details the proposed algorithm. Section V offers a comparative analysis of the complexity of the algorithms considered. Section VI presents the experimental results. Section VII is devoted to discussing the results, including key observations, identified limitations, and mitigation strategies. Section VIII summarizes the study's conclusions, and finally, Section IX outlines possible future directions aimed at refining the hybrid algorithm.

II. Related Works

Over the years, several studies have proposed new hybrid and compression-based approaches aimed at reducing the number of comparisons and improving the efficiency of string search, making them highly effective for large-scale applications.

SSTBMQS [1] is a hybrid algorithm that combines the best features of two existing algorithms: Tuned Boyer-Moore [2] and Quick-Skip Search [3]. Its results demonstrate superior performance in reducing character comparison attempts.

The hybrid Quick-Skip Search algorithm [3] seeks to optimize exact string matching by combining techniques from the Quick Search and Skip Search algorithms. The goal is to minimize the number of character comparisons and enhance search speed in large text volumes.

AbuSafiya [4] proposes accelerating text search in natural language through data compression. A fast compression algorithm was designed to encode each character using a single byte, thereby reducing text size and speeding up the search process. Although this technique is limited to texts in a single language, experimental results showed a significant reduction in search time.

III. Literature Review

A. Brute Force Search Algorithm

The brute force string search algorithm is one of the simplest methods for finding a substring within a larger text. It works by

comparing the target pattern with every possible substring of the text, starting from the first character and moving sequentially to the end. While its implementation is straightforward, its efficiency is low, with a worst-case time complexity of O(n * m), where, n is the length of the text and m is the length of the pattern [5]

B. Knuth-Morris-Pratt Algorithm

The Knuth-Morris-Pratt (KMP) algorithm was developed by Donald Knuth, James H. Morris, and Vaughan Pratt in 1977 [6], [7]. It builds a failure table that stores information about the prefixes and suffixes of the pattern. This table allows the algorithm to determine the optimal shift when a mismatch occurs.

Although the algorithm achieves linear time complexity in the worst case, its implementation can be complex, and it may underperform compared to other algorithms in practice [8].

KMP improves pattern matching by avoiding unnecessary comparisons. It uses a shift table to skip characters when a mismatch is found, relying on previous match information. Its time complexity is O(n + m), making it significantly more efficient than brute force search, especially in long texts [9].

C. Boyer-Moore Algorithm

The Boyer-Moore algorithm, developed by Robert S. Boyer and J. Strother Moore in 1977 [10], [11], is among the most efficient techniques for pattern searching in text strings. It compares the pattern to the text from right to left, enabling significant skips that optimize the search process. It performs particularly well on long texts and when the pattern contains infrequent characters.

In worst-case scenarios, such as when the pattern and text have many similarities or frequent characters are in unfavorable positions, its time complexity can reach O(n * m), where n is the length of the text and m is the length of the pattern.

Boyer-Moore employs two heuristics—the bad character rule and the good suffix rule—to skip over portions of the text that have already been examined. In the best case, it can achieve a time complexity of O(n/m), making it extremely fast for large-scale text processing [5].

D. Rabin-Karp Algorithm

The Rabin-Karp algorithm uses a hashing technique for pattern matching. Instead of performing direct comparisons, it computes a hash value for the pattern and for each window of the text, and checks for matches based on these hash values. While the average-case performance is O(n + m), it can degrade in the worst case due to hash collisions, especially when many substrings produce the same hash value [7], [12].

E. Aho-Corasick Algorithm

The Aho-Corasick algorithm is an efficient method for searching multiple patterns in a text simultaneously. It constructs a finite automaton that allows the simultaneous search of all patterns. During its preprocessing phase, it builds a trie (prefix tree) of the patterns, enhanced with failure pointers that manage mismatches efficiently. The algorithm runs in O(n + z + m) time, where, n is the length of the text, z is the total number of pattern occurrences found, and m is the sum of the lengths of all patterns. This makes it a robust solution for applications involving the search of multiple strings within large datasets [13].

IV. Proposed Algorithm

The proposed algorithm comprises two main components: a preprocessing phase using a length-pattern-based algorithm and a hybrid algorithm that, leveraging the preprocessing results, executes the final search for the specified pattern within the text.

A. Algorithm Based on Length Pattern (LP)

In order to design the algorithm, the following premises must first be considered

Premise 1: Texts contain information, which is represented through words, numbers, and certain symbols. These elements will be referred to as the information body, as they constitute the relevant content that needs to be located.

Premise 2: Each element of the information body has a specific length, and when combined to form information, they generate structural patterns that can be identified. Therefore, these patterns can be located within the text by analyzing the lengths of their components.

Premise 3: The substrings that form the information body are composed of sequences (words or numerical values) separated by whitespace or other non-informative characters (such as punctuation marks). When observing the text as a whole, it becomes evident that the informative sequences are delimited by non-informative ones, which enables the structure to be distinguished through length-based analysis.

In this context, both the original texts and the search patterns contain information represented by words, numbers, and symbols. All of these are processed through a preprocessing phase aimed at determining the individual length of each informative substring. Based on this information, a reduced representation of the original text is constructed—referred to as the pattern of lengths—as illustrated in Fig. 1.

In the preprocessing stage it is necessary to determine the “Function Include” in which the characters conforming the body of information are discriminated. Thus, the lengths of the information strings and those that do not constitute information are maintained.

Algorithm 1: Function Include

```
Include (character)
    alphaNum = ^[a-zA-Z0-9]
    inclusion = (character is in alphaNum)
    return (inclusion)
End
```

Based on the inclusion function, it is deduced that if a pair of terms that constitute information are separated by a character not included in 'String', such as punctuation marks, hyphens, or others, the algorithm will be able to identify them correctly. This is the case, for example, with the terms 'pre-processar' and 'pre processar', which, when subjected to preprocessing, produce the same result in terms of length pattern.

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Text	H	E	L	L	O		F	R	I	E	N	D	S	,		H	O	W		A	R	E		Y	O	U	?
Word Length	1	2	3	4	5		1	2	3	4	5	6	7			1	2	3		1	2	3		1	2	3	

Pre-Processing Text	0	5	0	7	0	3	0	3	0	3	0

Position Vector {0,6,15,19,23}

Position	0	1	2	3	4	5	6
Pattern	H	O	W		A	R	E
Word Length	1	2	3		1	2	3

Pre-Processing Pattern	0	3	0	3

Fig. 1. Preprocessing example.

Once the original text is available, and in which the search target text is to be found, the development of the search algorithm mechanism based on the length pattern (LP) proceeds as shown graphically in (Fig. 2).

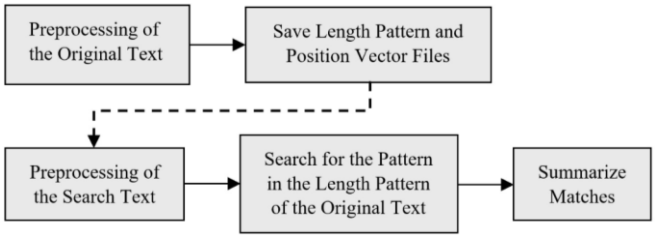

Fig. 2. Search sequence using the LP algorithm .

During the preprocessing of the original file—an initial stage of the algorithm—two text files must be generated simultaneously. The first file contains the pattern of word lengths, while the second stores a vector with the starting positions of each word that holds information. The pseudocode entitled “Preprocessor” presents the sequence of the algorithm based on the Pattern of Lengths (PL).

Algorithm 2: Preprocessor

```
Preprocessor(text):
    set txtInf = "", txtSep = "", txtIndexes = ""
    set inSep = false, inInf = false
    set x = 0, y = 0, indexAcc = 0, acc = 0
    For (char in text)
        If (char is alphanumeric) then
            If (inSep) then
                txtSep += formatToString(y, 2)
                y = 0, inSep = false
                acc += y
                indexAcc++
            End
            x++, inInf = true
        Else
            If (inInf) then
                txtInf += formatToString(x, 2)
                x = 0, inInf = false
                acc += x
                indexAcc++
            End
            y++, inSep = true
        End
        If (indexAcc == 2) then
            txtIndexes += acc
            acc = 0
        End
    End
    saveFile(txtInf, "txtLengths.inf")
    saveFile(txtIndexes, "txtIndexes.ind")
    Return 0
End
```

B. Hybrid Algorithm Based on LP and KMP (LP-KMP)

Once both the original text and the search pattern have been preprocessed, the word length patterns corresponding to each word in both texts are obtained. These patterns, extracted as text from the files generated during preprocessing, are used as input data for the KMP algorithm with the objective of locating the positions where the sequences of word lengths in the text and the pattern match. The identified match positions are stored in an index vector.

Subsequently, the actual positions in the original text are retrieved using the previously generated position vector. At these locations, a character-by-character comparison is performed between the corresponding text substring and the full search pattern. If all characters match, a valid match is confirmed. The pseudocode labeled “Hybrid” presents the complete sequence of the PL-KMP hybrid algorithm.

Algorithm 3: Hybrid

```
Hybrid (text, pattern, lengthsTxt, lengthsPatt, indexes):
    set posMatched = 0, indFirstWord = 0
    set sizePattWords = lengthsPatt.size()
    set wordsPatt = sizePattWords/2
    set countPatternWordsFounded = 0, indexFirstPosMatched =
0
    set indMatched = KMP(lengthsTxt, lengthsPatt)
    set finalResultIndexes as (emptyList)
    For (ind in indMatched)
        posMatched = ind/2
        indFirstWord = indexes[posMatched]
        indexFirstPosMatched = indFirstWord
            For (i = 0; i < sizePattWords; i += 2)
                Set   wSize=formatNumber(lengthsPatt.substr(i,
2))
                If          (pattern.substr(indexes[i/2],wSize)==
text.substr(indFirstWord, wSize)) Then
                    countPatternWordsFounded++
                    posMatched++
                Else
                    Break
                End
            indFirstWord = indexes[posMatched]
            End
            If (countPatternWordsFounded == wordsPatt) Then
                finalResultIndexes.push(indexFirstPosMatched)
                countPatternWordsFounded = 0
            End
    End
    return finalResultIndexes
End
```

Since the algorithm makes use of the information from the structure of the text, parameters are added to the search, corresponding to the lengths and positions of the words in the texts. Below is an example of the input parameters for the hybrid algorithm based on Fig. 1: 1) text: “HELLO FRIENDS, HOW ARE YOU?”, 2) pattern: “HOW ARE”, 3) lengthsTxt: “0507030303”, 4) lengthsPatt: “0303”, 5) indexes: “0,6,15,19,23”. In this case, the alphanumeric characters constitute the information to obtain the lengths in the preprocessing phase, while the remaining characters are part of the separators.

V. Performance Comparison

A. Selection of Algorithms for Experiments

In text pattern searching, the KMP and BM algorithms are widely recognized for their efficiency and ability to process large datasets. Their design incorporates features that enable them to surpass the limitations of simpler methods, such as brute-force search.

The KMP algorithm optimizes the search process by minimizing unnecessary comparisons through the use of a shift table that stores information about previous matches. This approach maintains a time complexity of O(n+m), which is significantly more efficient than traditional brute-force search, which can reach O(n*m). This efficiency is especially valuable in situations where multiple searches on the same text are required [6].

Conversely, the BM algorithm is highly efficient due to its use of advanced heuristics, such as the bad-character rule and the good-suffix rule. These techniques allow the algorithm to skip large portions of text rather than examining each character sequentially, leading to outstanding performance, particularly when searching for long patterns within extensive texts. This algorithm can achieve a complexity of O(n/m) in its best-case, making it very attractive for applications where speed and efficiency are crucial [5].

Due to their ability to reduce processing time, optimize memory usage, and enhance search accuracy, both algorithms are widely used in applications requiring high-speed text searching, such as search engines and text analysis tools. Consequently, KMP and BM remain among the most effective algorithms for pattern searching in computer science.

In this study, KMP and BM are compared with the proposed hybrid algorithm that combines structural preprocessing based on word lengths with the KMP algorithm. This proposal emerged after observing that an approach based solely on word lengths was not sufficient. Thus, a hybrid approach was developed to improve performance, and its evaluation is presented in the following sections.

B. Complexity of Algorithms

The complexity analysis of the proposed algorithm shows that in the worst-case the complexity is O(n*m). This calculation is obtained by considering the following phases of the algorithm: During the preprocessing phase, a traversal is performed over the text and the pattern to compute the size of each word, which results in a cost of O(n + m), where 'n' is the length of the text and 'm' is the length of the pattern. Subsequently, the KMP algorithm is employed, which operates with a complexity of O(n+m). Nevertheless, in the final step, when traversing the vector that contains the indices of the substrings with the same length as the pattern and performing character-by-character comparisons between the substring and the pattern, the worst-case complexity increases to O(n*m).

By contrast, in the best-case scenario, when the pattern consists of a single word, the complexity in the final step is reduced to O(n*1), which keeps the overall complexity at O(n+m).

Table I presents a comparison of the best and worst case complexity between the algorithms used in the experiments.

TABLE I. COMPARISON OF ALGORITHMS COMPLEXITY

Algorithm	Best Case Complexity	Worst Case Complexity
KMP (Knuth-Morris-Pratt)	O(n+m) [9]	O(n+m) [9]
BM (Boyer-Moore)	O(n/m) [5][11]	O(n*m) [14]
Hybrid (LP-KMP)	O(n+m)	O(n*m)

VI. EXPERIMENTAL RESULTS

The hybrid algorithm and the comparison software for the selected algorithms were executed on a personal computer with the specifications listed in Table II. The algorithm was developed using Microsoft Visual Studio, and the C++ compiler was used for compilation and execution.

TABLE II. SYSTEM INFORMATION

Resource	Description
Processor	Intel(R) Core(TM) i5-9300H 2.4GHz (8CPUs)
Memory	8192 MB RAM
Operating System	Windows 11 Home 64-bit

To evaluate the performance of the algorithms, a set of experiments was designed and classified into two groups. The first group corresponds to artificially created scenarios [15], in which computer-generated texts were constructed using fixed-length words and controlled repetitive patterns. The second group corresponds to real-world scenarios [16], consisting of public domain classical texts.

In the artificial scenarios, text files were generated with a predefined number of repeated words, all of the same length. Within these texts, specific words designated as occurrences were inserted, serving as search patterns. In these experiments, the BM, KMP, and the proposed hybrid algorithm PL-KMP were evaluated.

The tests were conducted under controlled conditions, measuring the number of CPU cycles required by each algorithm to locate the occurrences. The results corresponding to this first group of experiments are presented in Tables III, IV, and V, and are graphically illustrated in Fig. 3 to Fig. 14.

TABLE III. TEXT SCENARIOS CREATED WITH 1-CHARACTER WORDS

Number of Words	Occurrences	BM	KMP	LP-KMP
5000	1	496404	1645344	6054392
5000	3	618372	1838636	7911064
5000	5	511716	1661372	7691332
10000	1	1623122	5440384	19853538
10000	3	1620814	5462448	20295200
10000	5	1641032	5427442	20015822
15000	1	2430706	8101176	29889788
15000	3	2428924	8222824	29758070
15000	5	2431012	8243022	29865682
20000	1	1894232	6497074	23494778
20000	3	1886316	6342556	23382132
20000	5	1896082	6346574	23916868

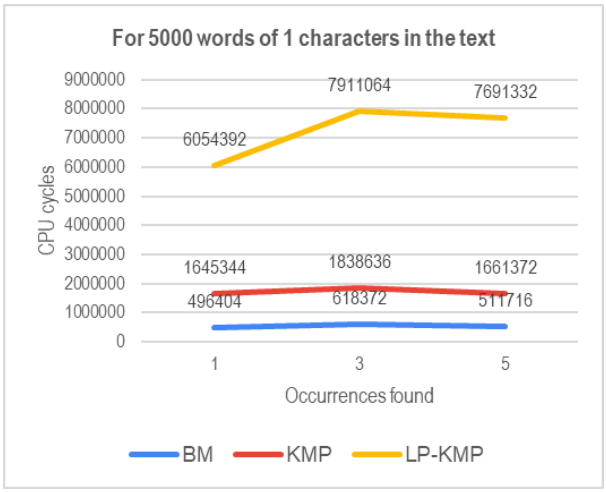

Fig. 3. Comparison of searches in texts of 5000 words of 1 character each word.

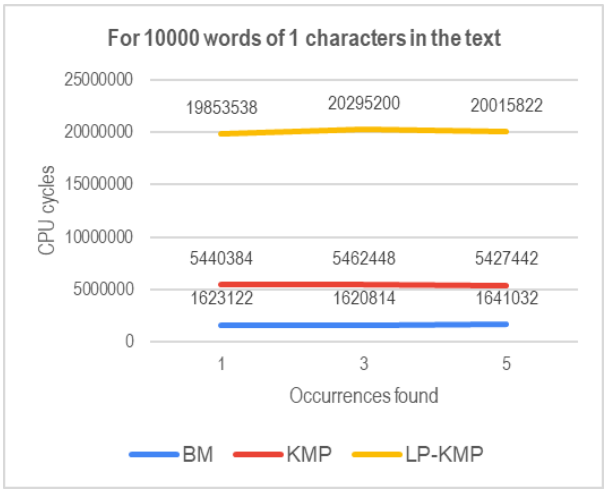

Fig. 4. Comparison of searches in texts of 10000 words of 1 character each word.

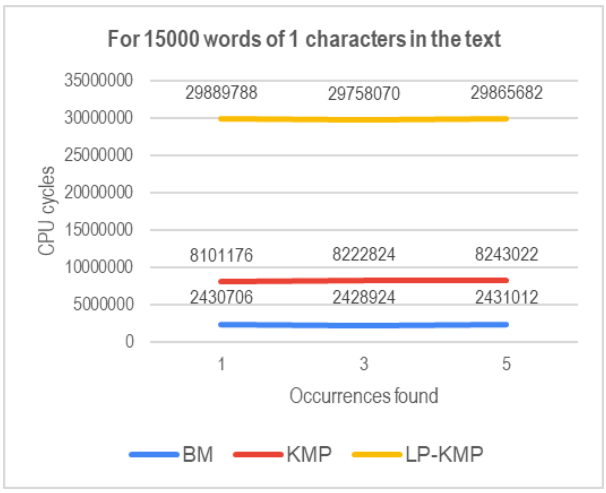

Fig. 5. Comparison of searches in texts of 15000 words of 1 character each word.

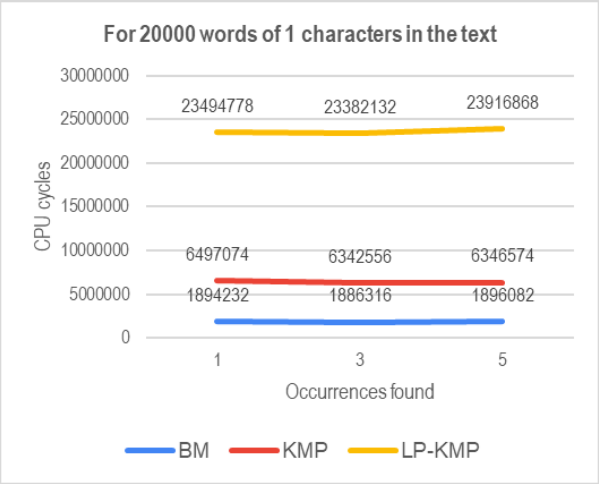

Fig. 6. Comparison of searches in texts of 20000 words of 1 character each word.

TABLE IV. TEXT SCENARIOS CREATED WITH 3-CHARACTER WORDS

Number of Words	Occurrences	BM	KMP	LP-KMP
5000	1	981200	1098608	6277770
5000	3	975608	1101816	6146920
5000	5	955380	1066634	6137210
10000	1	1956372	2184216	15324210
10000	3	2134800	2192866	12661116
10000	5	1955818	2294792	12539332
15000	1	2912518	3269166	31429612
15000	3	4846424	5439562	31367776
15000	5	4842008	5490228	31812352
20000	1	3859580	4224038	25805696
20000	3	3874884	4329296	24525460
20000	5	3773204	4225842	24736656

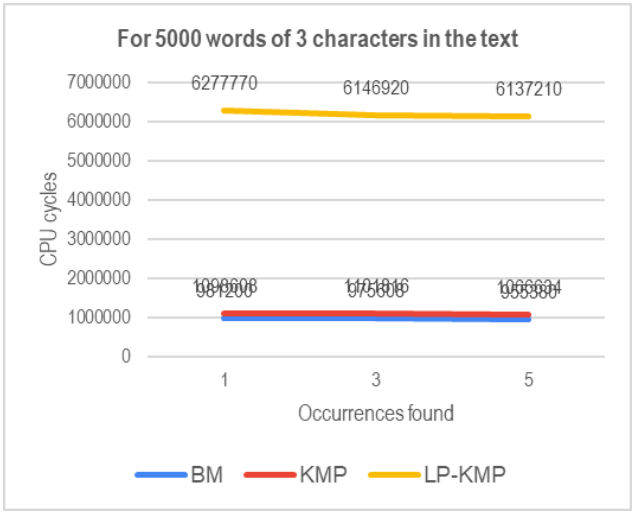

Fig. 7. Comparison of searches in texts of 5000 words of 3 characters each word.

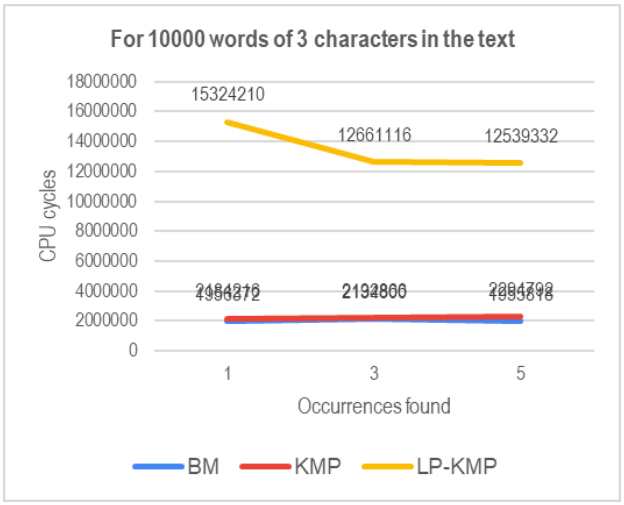

Fig. 8. Comparison of searches in texts of 10000 words of 3 characters each word.

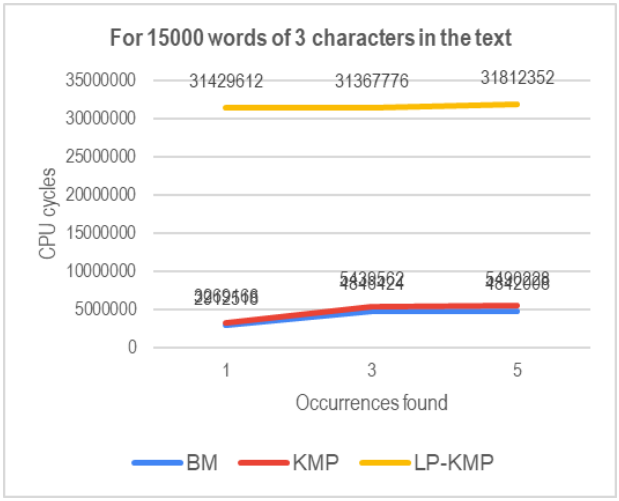

Fig. 9. Comparison of searches in texts of 15000 words of 3 characters each word.

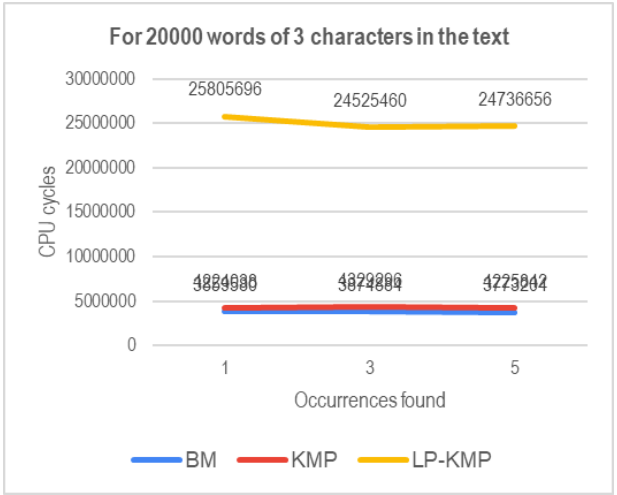

Fig. 10. Comparison of searches in texts of 20000 words of 3 characters each word.

TABLE V. TEXT SCENARIOS CREATED WITH 5-CHARACTER WORDS

Number of Words	Occurrences	BM	KMP	LP-KMP
5000	1	1464074	992586	8265800
5000	3	1481530	993654	10028736
5000	5	1448260	972152	6263314
10000	1	2842428	1927880	12150638
10000	3	2857908	1926674	12423264
10000	5	3181448	1945964	12134070
15000	1	4253702	2861360	18489042
15000	3	4269750	2865512	18541728
15000	5	4317254	2948970	18319860
20000	1	5652428	3815354	24354276
20000	3	5677966	3804260	24807472
20000	5	5674680	3814230	34425994

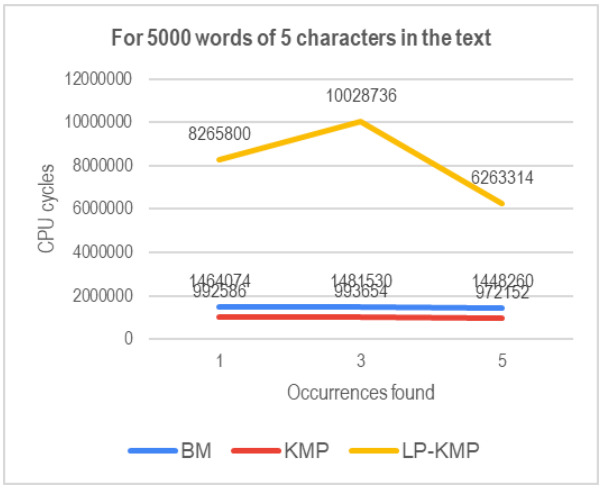

Fig. 11. Comparison of searches in texts of 5000 words of 5 characters each word.

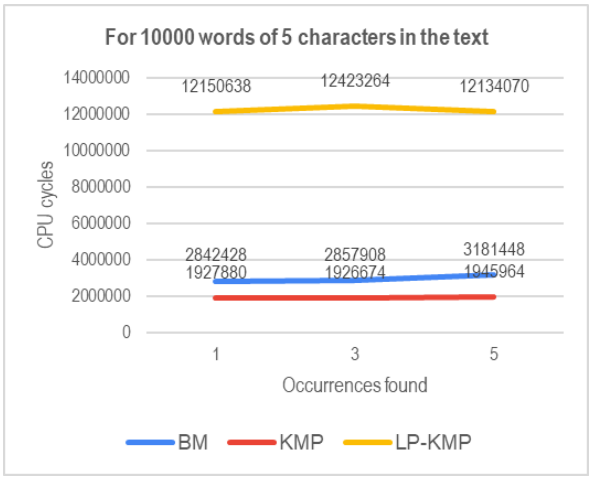

Fig. 12. Comparison of searches in texts of 10000 words of 5 characters each word.

A significant difference is evident in the approach of the proposed hybrid PL-KMP algorithm compared to the classical BM and KMP algorithms. In the evaluated scenarios, the hybrid algorithm does not outperform BM and KMP, primarily because its search mechanism performs between three to six times more operations for each detected match, thereby increasing the computational load in these cases.

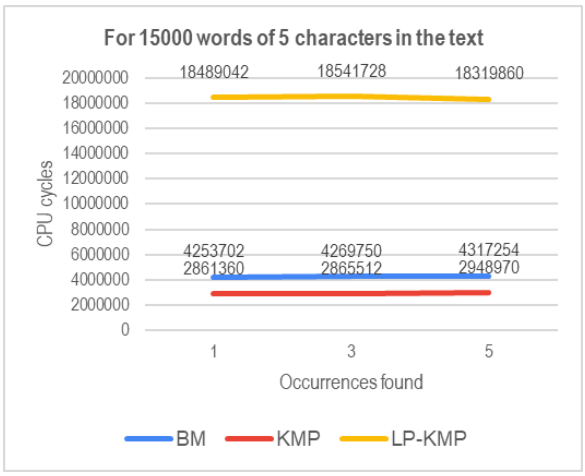

Fig. 13. Comparison of searches in texts of 15000 words of 5 characters each word.

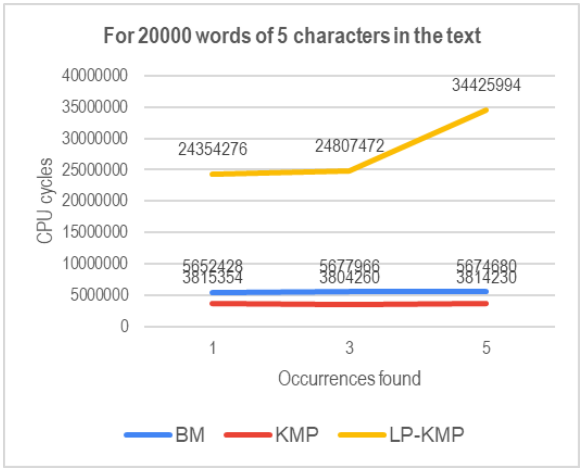

Fig. 14. Comparison of searches in texts of 20000 words of 5 characters each word.

In the second group of experiments, tests were conducted using five real-world texts corresponding to classic short stories. In these cases, the BM, KMP, and PL-KMP algorithms were again applied to search for patterns composed of one (1), three (3), and five (5) consecutive words. The results are presented in Tables VI and VII and are graphically illustrated in Fig. 15 to Fig. 17.

The results obtained in real-world scenarios highlight the true potential of the proposed hybrid algorithm. Compared to the KMP algorithm, which demonstrated the best performance in the most demanding synthetic scenarios, the hybrid algorithm managed to reduce the search time by more than half. This improvement is attributed to its preprocessing strategy based on length patterns, which enables the identification of structural matches prior to character-by-character comparison. By limiting comparisons only to cases where there is a preliminary match in word lengths, the total number of operations required is significantly reduced.

TABLE VI. EXPERIMENTS CREATED FOR REAL SCENARIOS

Experiment	Test Text		Number of Words	Number of characters	Words to search	Occurrences
1	1	Cinderella.txt	249	1465	"Cinderella"	10
2	2	Little Red Riding Hood.txt	262	1457	"little"	3
3	3	The Adventures of Pinocchio.txt	371	1997	"Pinocchio"	10
4	4	The Three Little Pigs.txt	408	2058	"built"	3
5	5	The Ugly Duckling.txt	284	1608	"ugly"	5
6	1	Cinderella.txt	249	1465	"for the ball"	1
7	2	Little Red Riding Hood.txt	262	1457	"talk to strangers"	1
8	3	The Adventures of Pinocchio.txt	371	1997	"magical place where"	1
9	4	The Three Little Pigs.txt	408	2058	"a house of"	1
10	5	The Ugly Duckling.txt	284	1608	"he found shelter"	1
11	1	Cinderella.txt	249	1465	"she turned a pumpkin into"	1
12	2	Little Red Riding Hood.txt	262	1457	"she walked through the forest"	1
13	3	The Adventures of Pinocchio.txt	371	1997	"who had been looking for"	1
14	4	The Three Little Pigs.txt	408	2058	"and rolled down the hill"	1
15	5	The Ugly Duckling.txt	284	1608	"and he saw a group"	1

TABLE VII. COMPARISON OF ALGORITHMS IN REAL SCENARIOS

Experiment	Test Text		CPU Cycles		
			BM	KMP	LP-KMP
1	1	Cinderella.txt	215396	173092	127972
2	2	Little Red Riding Hood.txt	140676	91996	63636
3	3	The Adventures of Pinocchio.txt	158826	122350	89916
4	4	The Three Little Pigs.txt	227556	125174	108230
5	5	The Ugly Duckling.txt	185114	116140	110520
6	1	Cinderella.txt	280302	165592	74070
7	2	Little Red Riding Hood.txt	184168	177350	75862
8	3	The Adventures of Pinocchio.txt	203524	201518	92962
9	4	The Three Little Pigs.txt	304024	204744	85164
10	5	The Ugly Duckling.txt	199110	172962	61580
11	1	Cinderella.txt	236340	201708	75332
12	2	Little Red Riding Hood.txt	203252	171854	136352
13	3	The Adventures of Pinocchio.txt	181358	179920	90966
14	4	The Three Little Pigs.txt	201598	210324	100404
15	5	The Ugly Duckling.txt	183504	173764	67588

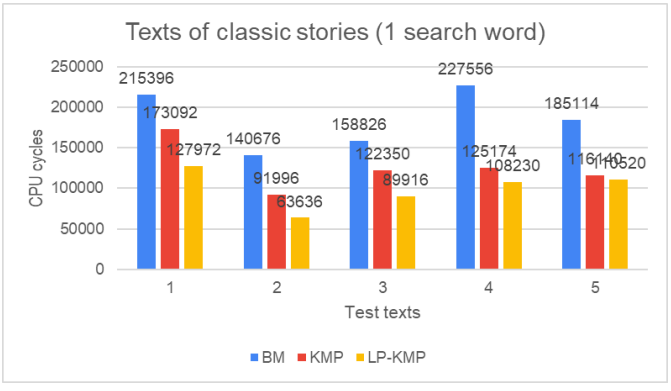

Fig. 15. Search for one (1) word in classic story texts.

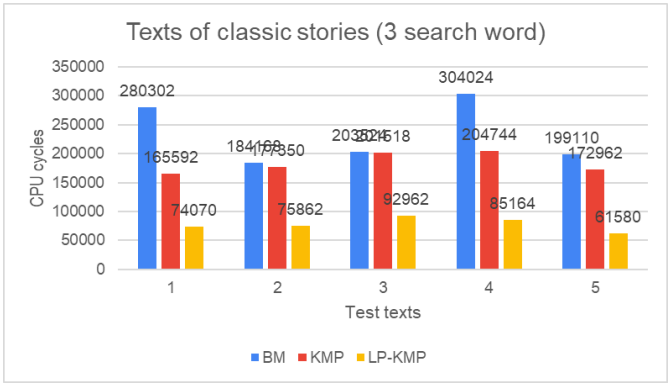

Fig. 16. Search for three (3) words in classic story texts.

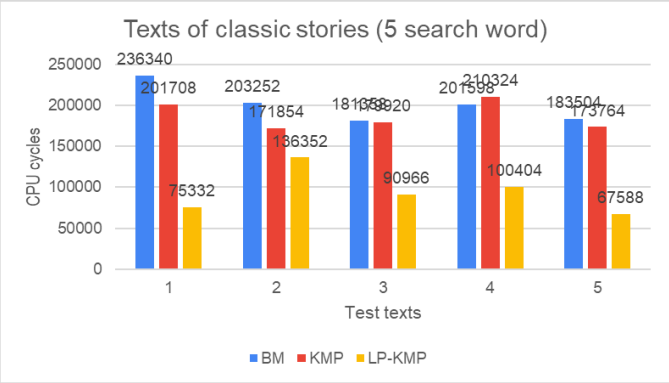

Fig. 17. Search for five (5) words in classic story texts.

VII. Discussion

The experimental results reveal a differentiated behavior of the hybrid PL-KMP algorithm compared to the classical BM and KMP algorithms, depending on the type of text processed.

In synthetic texts—designed to represent highly repetitive scenarios with uniform-length patterns—the traditional algorithms demonstrated better performance. This is because they are optimized for direct character comparisons, allowing them to maintain a lower computational load in predictable contexts. In such cases, PL-KMP exhibited a higher number of operations, due to its strategy of validating matches only after identifying length-based pattern matches.

In contrast, in real-world texts—such as classic tales with natural linguistic structures and variable word lengths—the hybrid algorithm showed a substantial improvement in search time. This advantage is attributed to its preprocessing mechanism, which efficiently filters potential candidates before initiating character-by-character comparison. This strategy reduces unnecessary operations and improves performance in less structured contexts.

A notable feature of the algorithm is its flexibility in detecting text patterns even when they are separated by various delimiters (spaces, hyphens, or punctuation marks), which is not considered by classical methods based solely on exact matches.

Nonetheless, the proposed algorithm presents certain limitations. One of them is its inability to detect substrings embedded within longer words, since the algorithm relies on identifying length patterns before executing textual comparisons. This limitation represents an opportunity for improvement in future versions of the algorithm, which could incorporate more flexible internal validation mechanisms.

Another significant limitation is the absence of a standardized corpus for evaluating pattern matching algorithms in text. Unlike other areas of natural language processing, there are no widely accepted benchmark datasets for this purpose. Furthermore, many related studies do not publish their datasets, making it difficult to perform objective comparisons between different approaches.

To contribute to reproducibility and foster further research in this area, the datasets used in this study—both the artificial scenarios and the real-world texts—have been made publicly available [15], [16]. This initiative aims to establish a fairer and more standardized basis for comparing future text search algorithms.

VIII. Conclusion

In this research, a hybrid text search algorithm has been developed and evaluated, which combines a structural preprocessing based on length patterns with the KMP algorithm. The experimental results demonstrate that, in scenarios with real texts and diverse linguistic structures, the hybrid algorithm outperforms traditional methods in terms of efficiency, thanks to its ability to reduce the number of comparisons through prior filtering.

In contrast, in highly structured synthetic scenarios, traditional algorithms like KMP and BM showed superior performance, as they are optimized for direct comparisons in uniform patterns.

The main advantages of the proposed algorithm include its high search speed in real texts and its flexibility to identify textual patterns even when they are separated by non-alphanumeric characters —such as hyphens, spaces, or punctuation marks. This feature allows for the equivalent identification of patterns such as “aaa-bbb-ccc,” “aaa bbb ccc,” or “aaa.bbb.ccc,” as long as the delimiters have been defined as excluded by the inclusion function.

However, a significant limitation of the algorithm is its inability to identify substrings contained within longer words, as the matching process is initially based on the sequence of lengths. This aspect represents an opportunity for improvement in future versions that incorporate more flexible search techniques after the initial filtering.

IX. Future Work

As part of future work, the development of an improved version of the hybrid algorithm is proposed, which would enable the detection of patterns embedded within compact strings. This would overcome the current limitation of relying exclusively on exact matches in length sequences. A possible solution is the incorporation of partial search techniques applied to candidate fragments once preliminary structural similarities have been identified.

Additionally, the adaptation of the algorithm to multilingual environments is considered, which would open up possibilities for its application in more diverse contexts such as information retrieval systems, search engines, or large-scale text analysis.

Finally, it is proposed to investigate the integration of the algorithm with machine learning techniques, with the aim of dynamically optimizing preprocessing criteria based on the linguistic characteristics of the text. This integration would enable greater adaptability and efficiency in real-world applications.

REFERENCES

[1] Naser M. A. S, Al-Dabbagh, S. S. M, and N. H. Barnouti. Fast hybrid string matching algorithm based on the quick-skip and tuned boyer-moore algorithms. International Journal of Advanced Computer Science and Applications, 8(6):117–127, 2017.S

[2] A. Hume and D. Sunday(1991). "Fast String Searching". Software: Practice and Experience, 21(11), 1221-1248.

[3] Naser, Mustafa Abdul Sahib, and Mohammed Faiz Aboalmaaly. "QuickSkip search hybrid algorithm for the exact string matching problem." International Journal of Computer Theory and Engineering 4.2 (2012): 259.

[4] AbuSafiya, M. (2021). Speeding up Natural Language Text Search using Compression. International Journal of Advanced Computer Science and Applications (IJACSA), 12(4).

[5] R. S. Boyer y J. S. Moore, "A fast string searching algorithm," IEEE Transactions on Computers, vol. 20, pp. 962-970, 1971.

[6] D. Knuth, J, Morris, and V. Pratt, "Fast pattern matching in strings," SIAM journal on Computing, vol. 6 No.2, pp. 323-350, 1977

[7] Z. Barut and V. Altuntaş, "Applied Comparison of String Matching Algorithms", GBAD, vol. 12, no. 1, pp. 76–85, 2023

[8] Ziviani, N. (2007). Diseño de algoritmos con implementaciones en Pascal y C. (J. Adiego, Trad.) Madrid, España: Thomson.

[9] Paulson, L. C., "Knuth–Morris–Pratt String Search", 2025.

[10] Y. S. Purwanto, M. F. Rifai, H. Jatnika, and G. A. Ardelia, "Information Retrieval in Text-Based Document using Boyer Moore Algorithm," JATISI (Jurnal Teknik Informatika dan Sistem Informasi), vol. 9, no. 2, pp. 1308–1316, Jun. 2022.

[11] Lecroq, T., "A fast implementation of the good-suffix array for the Boyer-Moore string matching algorithm", Art. no. arXiv:2402.16469, 2024.

[12] M. O. Rabin y R. Karp, "Efficient randomized pattern-matching algorithms," IBM Journal of Research and Development, vol. 31, no. 2, pp. 256-267, 1987.

[13] S A. V. Aho y M. J. Corasick, "Efficient string matching: an aid to bibliographic search," Communications of the ACM, vol. 18, no. 6, pp. 333-340, 1975

[14] Benavides, K. R. (19 de 11 de 2014). Algoritmos de Búsqueda en Texto. Obtenido de http://www.kramirez.net/wp-content/uploads/2012/02/Algoritmos-de-Busqueda-Secuencial-de-Texto.pdf

[15] Test texts - created scenarios, https://drive.google.com/drive/folders/1bn6xnIkZTWDAn223ppp7qhwMywOXBREh?usp=drive_link

[16] Test texts - real-life scenarios, https://drive.google.com/drive/folders/1q1TGX2FE9lgHBzVMOf24U_nxgLFZZP-Z?usp=drive_link