The Journal of Supercomputing (2023) 79:367–388
https://doi.org/10.1007/s11227-022-04673-3

A new fast technique for pattern matching in biological sequences

Osman Ali Sadek Ibrahim[1] · Belal A. Hamed[1] · Tarek Abd El-Hafeez[1,2]

Accepted: 20 June 2022 / Published online: 10 July 2022

Abstract

At numerous phases of the computational process, pattern matching is essential. It enables users to search for specific DNA subsequences or DNA sequences in a database. In addition, some of these rapidly expanding biological databases are updated on a regular basis. Pattern searches can be improved by using high-speed pattern matching algorithms. Researchers are striving to improve solutions in numerous areas of computational bioinformatics as biological data grows exponentially. Faster algorithms with a low error rate are needed in real-world applications. As a result, this study offers two pattern matching algorithms that were created to help speed up DNA sequence pattern searches. The strategies recommended improve performance by utilizing word-level processing rather than character-level processing, which has been used in previous research studies. In terms of time cost, the proposed algorithms (EFLPM and EPAPM) increased performance by leveraging word-level processing with large pattern size. The experimental results show that the proposed methods are faster than other algorithms for short and long patterns. As a result, the EFLPM algorithm is 54% faster than the FLPM method, while the EPAPM algorithm is 39% faster than the PAPM method.

Keywords Bioinformatics · Character comparison · Pattern matching · String Matching · DNA Sequences

✉ Belal A. Hamed
belal.ahmed@mu.edu.eg

Osman Ali Sadek Ibrahim
osman.ibrahim@mu.edu.eg

Tarek Abd El-Hafeez
tarek@mu.edu.eg

[1] Department of Computer Science, Faculty of Science, Minia University, EL-Minia, Egypt

[2] Computer Science Unit, Deraya University, EL-Minia, Egypt

1 Introduction

A sequence, text, or database is scanned to discover the positions of a pattern in the text in pattern matching [1, 2]. This type of problem must be addressed for a variety of reasons, including its applications in image and signal processing, search engines, text processing, information retrieval, question–answer systems, and chemistry [3–6].

Pattern matching issue is prevalent in several areas of computational bioinformatics, such as basic local alignment search, biomarker identification, sequence alignment, proteogenomic mapping, homologous series recognition and proteogenomic mapping. In these fields, it is necessary to detect the positions of patterns in databases, as well as those of nucleotides and amino acids [7–9]. Gene analysis and DNA sequences can be used to investigate suspected illness or anomaly diagnoses in biotechnology, forensics, medicine, and agriculture research. DNA sequence analysis can be used to compare a gene to comparable genes in the same or different animals, as well as to predict its function. Another application, the functionality of a newly found DNA sequence may be predicted by comparing it to existing DNA sequences. This method has been employed in several medical investigations and applications.

Although the literature [9–18] contains many generic and specialized DNA pattern matching methods, more efficient algorithms are still needed. Because existing methods have high-computational costs [19]. So, they may not be well scalable for databases or huge DNA sequences. Unlike approximation pattern matching [20–28].

The problem of inefficiency in exact pattern matching, which entails detecting all instances of a pattern in a text, is addressed in this study. The goal of this study is to improve the efficiency of FLPM and PAPM algorithms in order to solve the computational cost problem that has plagued earlier studies [29]. Unlike the literature [20–28, 30, 31]. The proposed methods do not have preparation and matching steps. Similar intervals of the text to be matched with the pattern are detected by the matching procedure. In this work, the FLPM and PAPM algorithms' preprocessing stage will be modified to reduce comparison time. For the new approach, we calculate the number of windows, then compare them again at the matching stage.

1.1 Motivations of using pattern matching

Pattern matching is a technique for determining whether or not the elements of a pattern are present in an observed string sequence. Unlike pattern recognition, the match must usually be exact. Pattern matching sequences often include outputting the locations of a pattern inside a string sequence, outputting some component of the matched pattern, and substituting the matching pattern with another string sequence (i.e., search and replace). The pattern matching idea is used in a variety of applications. Figure 1 depicts these uses.

For pattern matching investigations, we focused on DNA sequence. To handle this kind of data, we need a better search technique. Matching patterns will help

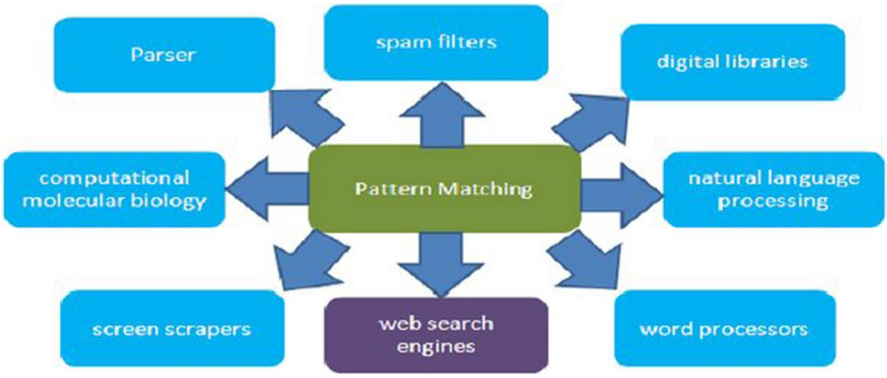

Fig. 1 Applications of pattern matching [32, 33]

you achieve the best and most appropriate result. To detect matching patterns, many algorithms have been utilized.

1.2 Algorithm's notation

MatchingAlgorithms utilize the notation shown below. Section 2 introduces a review of related work, and Sect. 3 explains the problem. The proposed algorithm is described in Sect. 4. Section 5 compares the performance and effectiveness of the PM algorithms against those of other algorithms. Finally, Sect. 6 brings the research to a conclusion and future work.

2 Related Work

Several pattern matching algorithms have been developed in order to reduce the number of comparisons done during search operations. To reduce the number of comparisons, the matching process is usually split into two parts. During the pre-processing and searching phases, the distance (shift value) by which the pattern window will move is determined. During the searching phase, this shift value is used to find the pattern in the text with as few character comparisons as possible. In this section, we will show two earlier pattern matching algorithms, the FLPM and PAPM algorithms, which have been improved by our proposed methodology.

2.1 First-Last-pattern matching (FLPM) algorithm

FLPM [29] is a Divide and Conquer Pattern Matching (DCPM) upgrade that consists of two stages, pre-processing and matching. Comparisons are the basis of FLPM. In the pre-processing stage, the text is scanned to identify windows that will be used in the matching stage later. The search will be in the range t [0… n-m] assuming that m is the pattern length and n is the text length. If the two characters

are identical, then increasing the number of windows and the algorithm compares the first character in the pattern with the first character in the text and the last character in the pattern with the last character in the text. This method is repeated throughout the pre-processing stage. During the matching step, it scans the windows to find all instances of the pattern inside the text.

2.2 Processor-aware-pattern-matching (PAPM) algorithm

This method differs from the FLPM algorithm in how pattern p characters and text t characters are compared PAPM [29]. It compares words composed of many characters, whereas FLPM use a character-based pattern matching method. It compares patterns using a term made up of numerous letters and is powered by the processor. PAPM algorithm compares two words at a time, with word_len = b/8 determining the length of each word, where b-bit represents the kind of processor (either 32 or 64). If b = 32, it will compare two four-letter words. The pre-processing method examines the interval t [0… n−m] for the first word of the pattern in the text. To find the word p [0…word len1], the beginning of the index is preserved in the windows array. When the PAPM algorithm searches a window, the bigger the number of characters (words) examined in the text results in fewer windows, which minimizes the time necessary for the next step

3 Research problem and the proposed solutions

The bulk of biological data has expanded significantly in recent years. Hey must be examined in a fair amount of time. This problem arises in molecular biology because amino acid or nucleotide sequences are frequently used to approximate biological molecules. Another example is the basic knowledge of species DNA sequences and the difficulty in retrieving this information by pattern matching. Moreover, identifying potential irregularities or mistakes in a DNA sequence frequently necessitates DNA sequence analysis. Pattern matching is also useful in domains such as phylogenetic and evolutionary biology. Specific DNA sub-sequences are retrieved from the genomic data of various creatures' species in these applications to better comprehend their relatedness, ancestry, and origin. Regarding this context, a pattern matching algorithm must be able to search in datasets spanning gigabytes to terabytes or more, as well as complete genomes containing 3 billion base pairs [14]. The DNA sequences, on the contrary, are extremely lengthy. As a result, the time spent in matching with the pattern is regarded as the most essential factor.

Consider the text t with length n over the alphabet $\sum$ and the pattern p with length m. A string is defined as a series of 0 or more alphabet symbols $\sum$ represents the set of all possible strings over the letter $\sum^*$. If x = abc, where a, b, and c $\in \sum^*$, then c is a substring of string x. Pattern p is stored in an infinite array p [0..m − 1], in which m > 0. The (i + 1)—st symbol of p is represented by p[i], in which $0 \le i < m$. Furthermore, p[i…j] denotes a substring of p from the (i + 1)—st symbol to the (j + 1)—st symbol of p, where $0 \le i \le j < m$.

The text is scanned in the proposed pattern matching methods to find windows of length m. In the matching step, the algorithms compare the pattern's characters one by one with those in the window to determine the overall pattern's look. The other windows are checked after a complete match or mismatch to see if they match the text.

4 Methodology

This section describes a straightforward **Enhance-First-Last Pattern Matching (EFLPM)** method and **Enhance- Processor-Aware Pattern Matching Algorithm (EPAPM)**. EFLPM is an enhancement to FLPM that combines the pre-processing and matching stages of FLPM into a single phase to minimize time complexity.

4.1 The proposed EFLPM algorithm

The FLPM pre-processing stage scans the text to highlight text windows that will be used later in the matching stage, because FLPM is based on comparisons. The windows whose first and last characters in the pattern match the first and last characters of the text in pattern size are extracted during the pre-processing stage. If the initial and last characters match, they will be added to the matrix of windows; if they don't, the pattern will be moved one letter and the process will be repeated. The procedure is then repeated throughout the paragraph. The matching stage involves comparing the extracted windows to the pattern once more. The proposed EFLPM algorithm flowchart is shown in Fig. 2.

The steps of the proposed algorithm EFLPM can be summarized as follows:

Step 1: Read the DNA sequence dataset as a fasta file.

Step 2: Initialize the counter at 0 as the initial value of the while loop counter with count 0 and This will continue till this counter hits n-m.

Step 3: Check the 1'st and last characters of the pattern, i.e., t[count] and t[count + m 1], and then compare to p [0] and p [m − 1]. The matching process begins if the results of both comparisons are equivalent.

Step 4: If the comparison yields a false result, the pattern does not exist in this section of the text since the initial and last letters did not match. However, if they match, there's a chance the pattern will match inside this text as well. Therefore, conduct the matching process immediately in this window, which is called the window.

Step 5: If this pattern and this section of the text are same, the complete pattern inside the text is identical. If there is a perfect match for the pattern in the text, increase the number of matches by one and return to the loop to finish the text.

Step 6: In the last step, if the loop is ended, returns the start index for all instances of pattern p in text t.

Figure 3 illustrates the pseudocode of the proposed algorithm EFLPM. Figures 4, 5, 6 and 7 provide a simple example of our EFLPM algorithm's pattern marching steps.

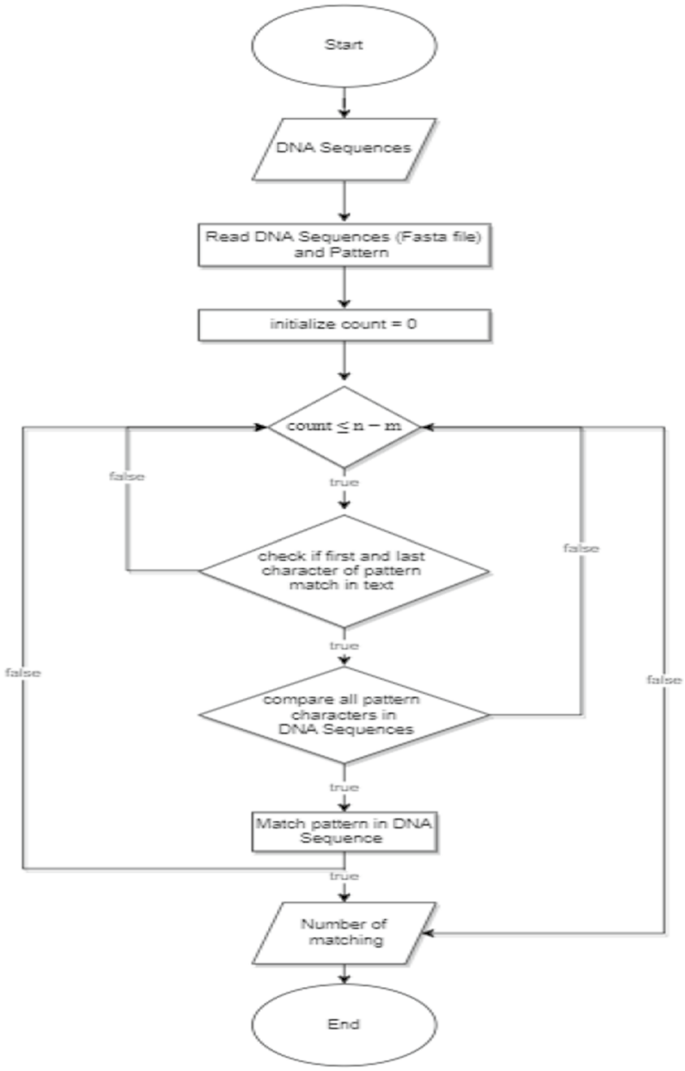

Fig. 2 A flowchart for proposed algorithm EFLPM

As an example, Figs. 4, 5, 6 and 7 shows pattern p and text t in list t[0.0.53], which is AAGCGTA in list p[0.0.6]. The algorithm searches for the first and last items of the pattern, i.e., p[0] and p[6], in the text. At the beginning of the algorithm 1, p[0] and p[6] are aligned to t[0] and t[6], respectively. As a result, the window index array contains the start index of the 1'st window, i.e. 0. Following this example, the algorithm identifies eleven other windows. if the result is true, the next step checks the pattern with this window, and increase the match counter if matching occurs, and store the first index of this window in match_index,

Input: Text t and pattern p stored in the arrays of t[0..n−1] and P p[0..m − 1], respectively, over finite alphabet ∑.

Output: The start index for all occurrences of pattern p in text t.

```
1.  count ← 0
2.  num_match ← 0
3.  WHILE count ≤ n − m DO
4.        IF t [count] = p[0] THEN
5.              IF t [count + m − 1] = p[m − 1] THEN
6.                 c ← 1
7.                 WHILE c ≤ m - 2 DO
8.                       IF p[c]  != t[count + c] THEN
9.                            Break
10.                      END-IF
11.                      c ← c  + 1
12.                END-WHILE
13.                IF c=m-1 THEN
14.                      window_index[count] ← count
15.                      num_match ← num_match+1
16.                END-IF
17.       END-IF
18.       count  ← count + 1
19. END-WHILE
```

Fig. 3 The pseudocode of the proposed algorithm EFLPM

otherwise skip this window. Consequently, the window of t[25.0.31] and the pattern are the same (Tables 1 and 2).

4.2 Proposed EPAPM algorithm

The Enhance-Processor-Aware Pattern Matching (EPAPM) algorithm, which is based on PAPM, is described in this section. The comparison of pattern p characters and text t characters differs from the FLPM method. FLPM compares words with several characters, whereas PAPM compares characters. PAPM compares two words at the same time using a CPU's processing capacity. A bit processor's registers are slightly longer, and the processor can compare data from two registers during each execution cycle. The number of processable bytes (or word length) for this processor is computed as word_len = b/8 since each byte (or character) contains eight bits. It means that, the processor may compare one word to another by reading its registers each time. A 64-bit CPU, for example, might compare four words of eight characters.

We'll apply the same strategy we did in FLPM to reduce the time complexity of the pre-processing stage and match only one process that does the same job in less time in this approach. The EPAPM Algorithm Steps can be summarized as follows:

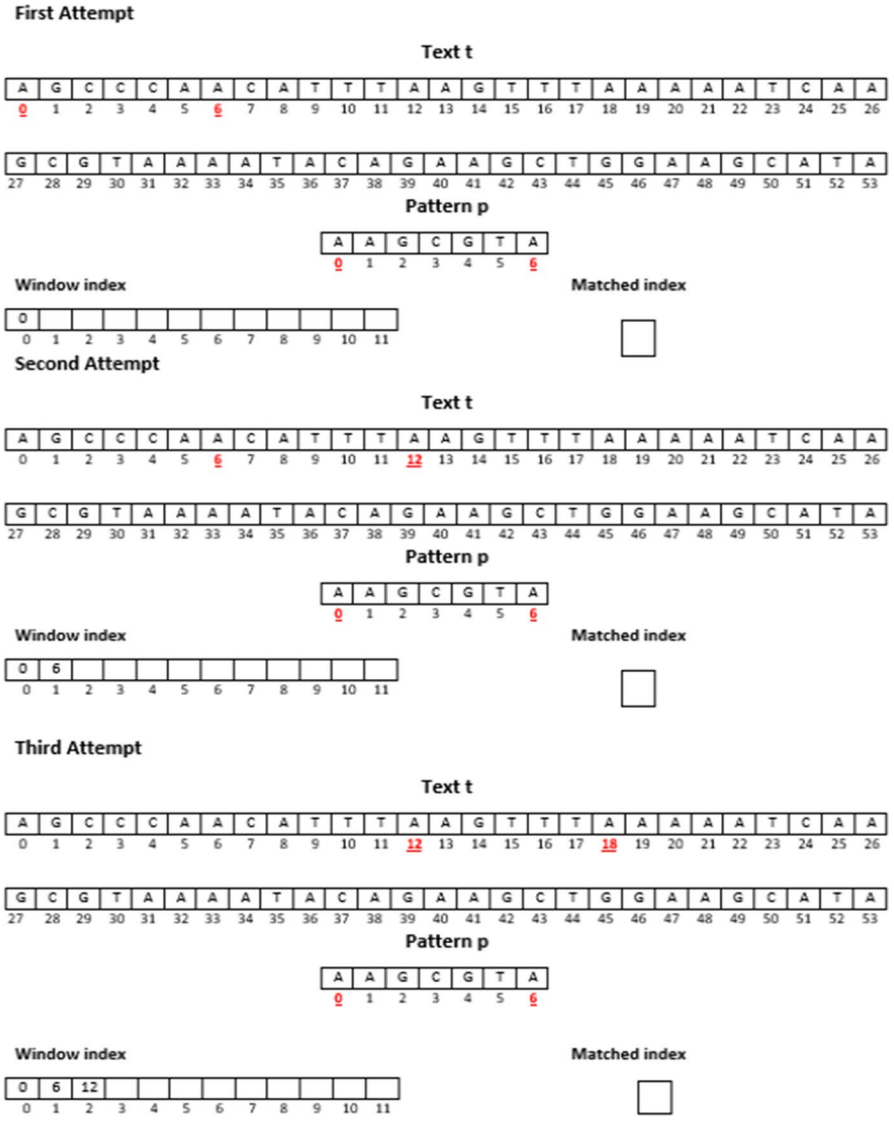

Fig. 4 The operation of the EFLPM algorithm

Step 1: Read the DNA Sequence dataset as (Fasta file)

Step 2: Initialize the counter at 0 as the initial value of the while loop counter and word_len by b/8 (Described in Section 4.2) and k by the modulus of m and word_len.

Step 3: The start index for the word comparison is determined at the start of this phase. Setting this start index ensures that the method runs successfully even if the lengths of the pattern and windows are not integer multiples of the word length.

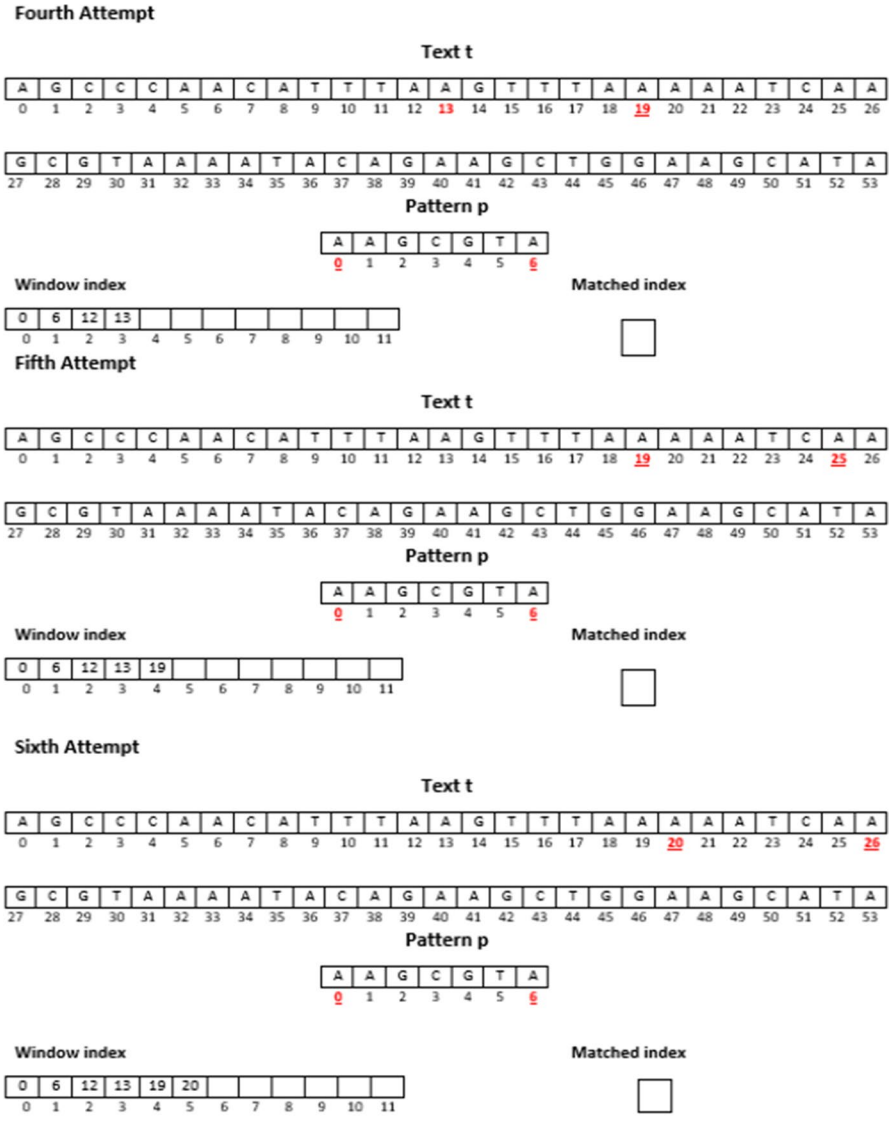

Fig. 5 The operation of the EFLPM algorithm

Step 4: This algorithm's while loop begins with count = 0 and continues until the counter reaches n-m.

Step 5: Check the two words based on word_len (a word can contain 4 or 8 characters).

Step 6: If the two words are matched, check all the patterns in the text.

Step 7: If all words of the pattern are matched in text, increase the number of matches by one and return to the loop to continue rest of text.

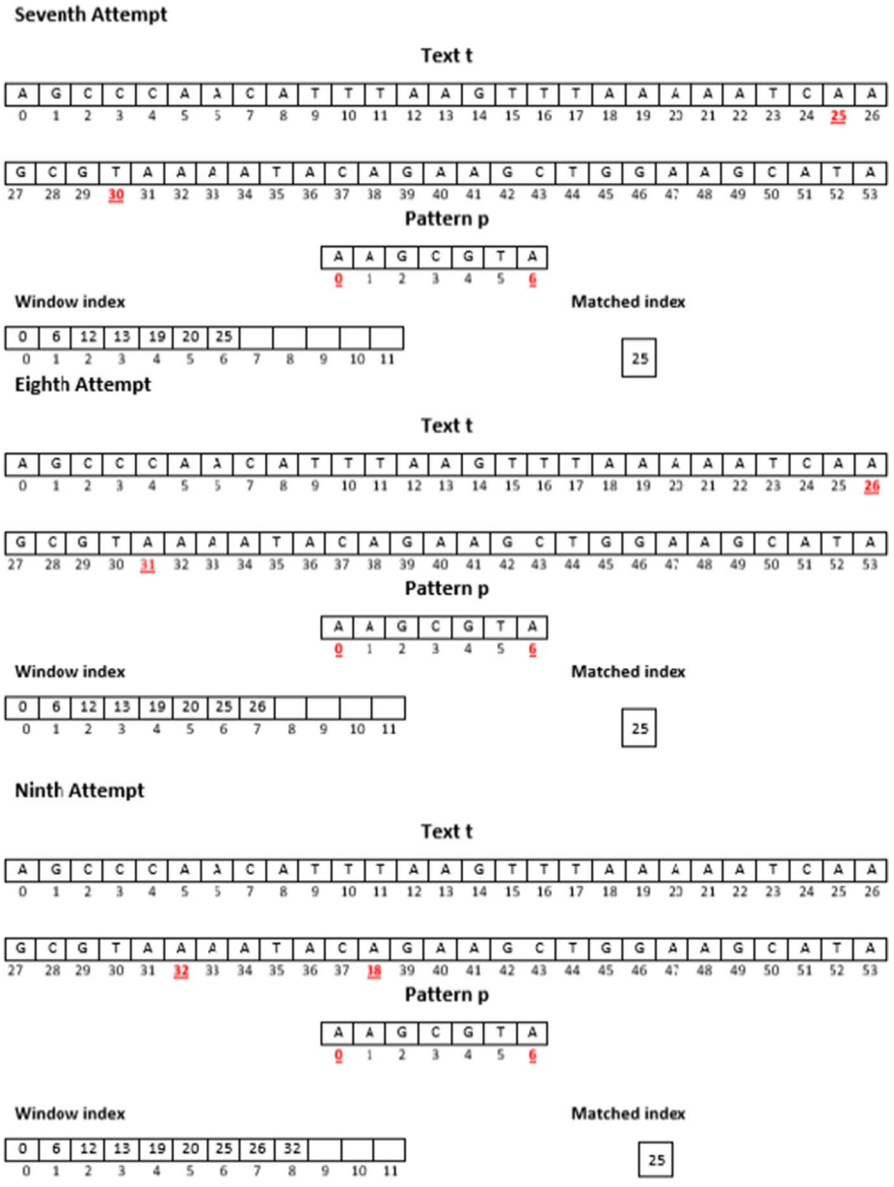

Fig. 6 The operation of the EFLPM algorithm

Step 8: In the last step, returns the start index for all instances of pattern p in text t if the loop is finished.

We illustrate the Flowchart and Pseudocode for EPAPM Algorithm in Figs. 6 and 7.

Figure 8 gives an example of using the EPAPM Algorithm run on a 32-bit processor. In this algorithm, the first word (consisting of the first 4 alphabets) of pattern

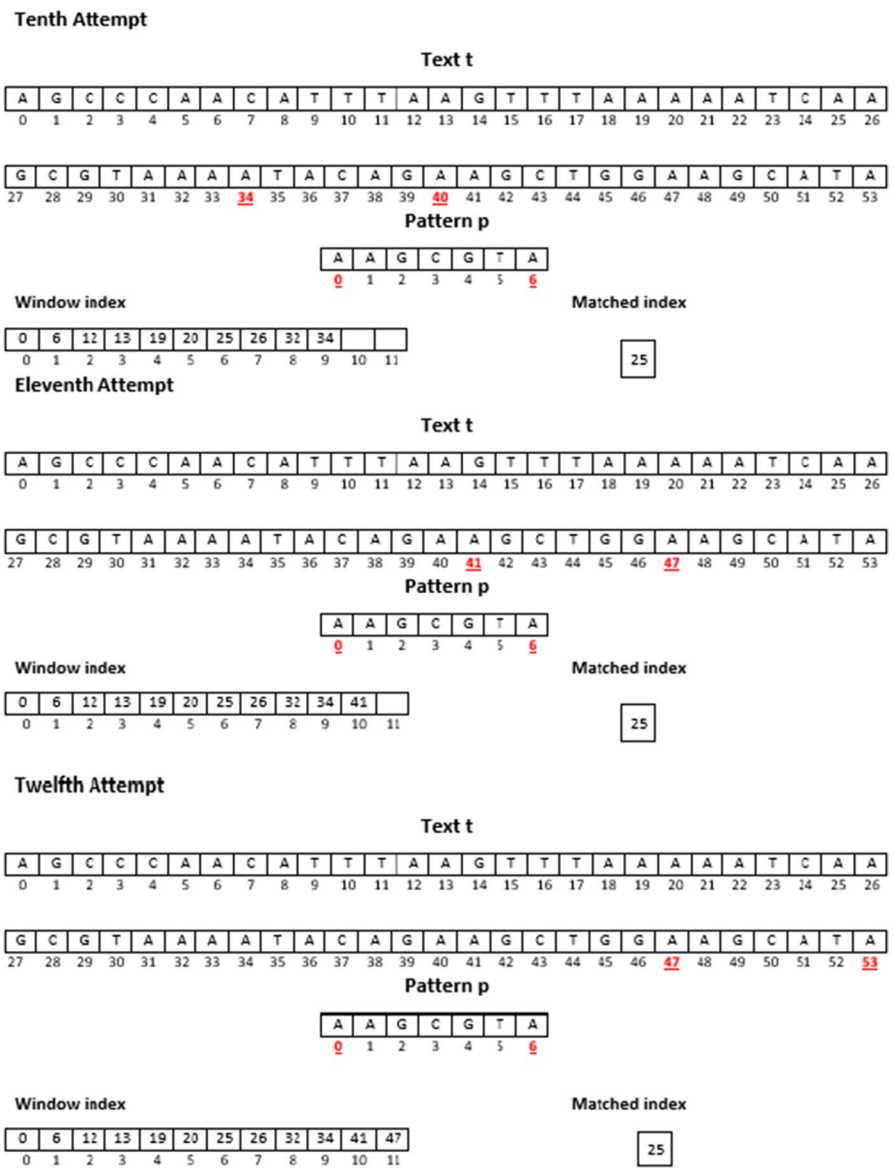

Fig. 7 The operation of the EFLPM algorithm

Table 1 Notations used for algorithms

Notation	Used for
T	The text (string)
P	The pattern (string)
N	The length of the text
M	The length of the pattern (string)

Table 2 Pattern matching algorithms comparisons

No	Algorithm	Time complexity (pre-processing)	Time complexity (Matching)	Comparison order	Year
1	BM [34–38]	O(m + σ)	O(m × n)	Right to Left	1977
2	Horspool [36, 39]	O(m + σ)	O(m × n)	Rightmost character then leftmost character then moves forward	1980
3	Karp-Rabin [34, 38]	O(m)	O(m + n)	From left to right	1987
4	Zhu-Takaoka [40]	$O(m+\sigma^2)$ ztBc $O(m^2)$ bmGs		right to left	1987
5	d-BM [41]	O(n)	O(m × n)	Right to Left	2004
6	BF [42]	N/A	O(m × n)	any order	2004
7	KMP [22, 36]	O(m)	O(n + m)	left to right	2012
8	FLPM [29]	O(n)	O(m × n)	specific order	2020
9	PAPM [29]	O(n)	O(m × n)	specific order	2020

p is searched in text t. the window_index array is composed of three start indexes of the found windows, i.e., 25, 40 and 47. For this example, it should be noticed that the EFLPM method identifies 12 start indexes as potential intervals or windows in this case. As a result, EPAPM decreases the number of recognised windows. Because the remainder of pattern length over word length is 3 in the matching stage, the start index for matching is also 3. As a result, the second word of the pattern that corresponds to the second word of windows is CGTA. After this phase is completed, only one of the two windows (i.e., t[25.0.31]) is matched with the pattern (Figs. 9 and 10).

5 Experiential results

In this part, the performance of the suggested algorithms (EFLPM and EPAPM) is compared to that of the Boyer-Moore (BM), Horspool, Karp-Rabin, Zhu-Takaoka, d-BM, KMP, FLPM, and PAPM algorithms. The computer environment used to execute many simulated algorithms has the following specifications: Windows 10 Home 64 bit, 8 GB RAM, Intel(R) Core(TM) i7-7500U CPU 2.70 GHZ.

The word length for the EPAPM and EFLPM algorithms was considered eight bytes due to the use of a 64-bit machine. The Python programming language was used to run the simulation. Each experiment involved searching the reference for ten patterns and reporting the average of the results. Because all tests used the HRG dataset, which is published in [43], each file had over 12 million characters, this time overhead was eliminated throughout the simulation. Table 3 shows the results of the performance evaluation of the simulated algorithms in terms of time cost. The rest of this section will give the findings of pattern matching algorithm comparisons with DNA sequences larger than 12 million characters and varied pattern sizes.

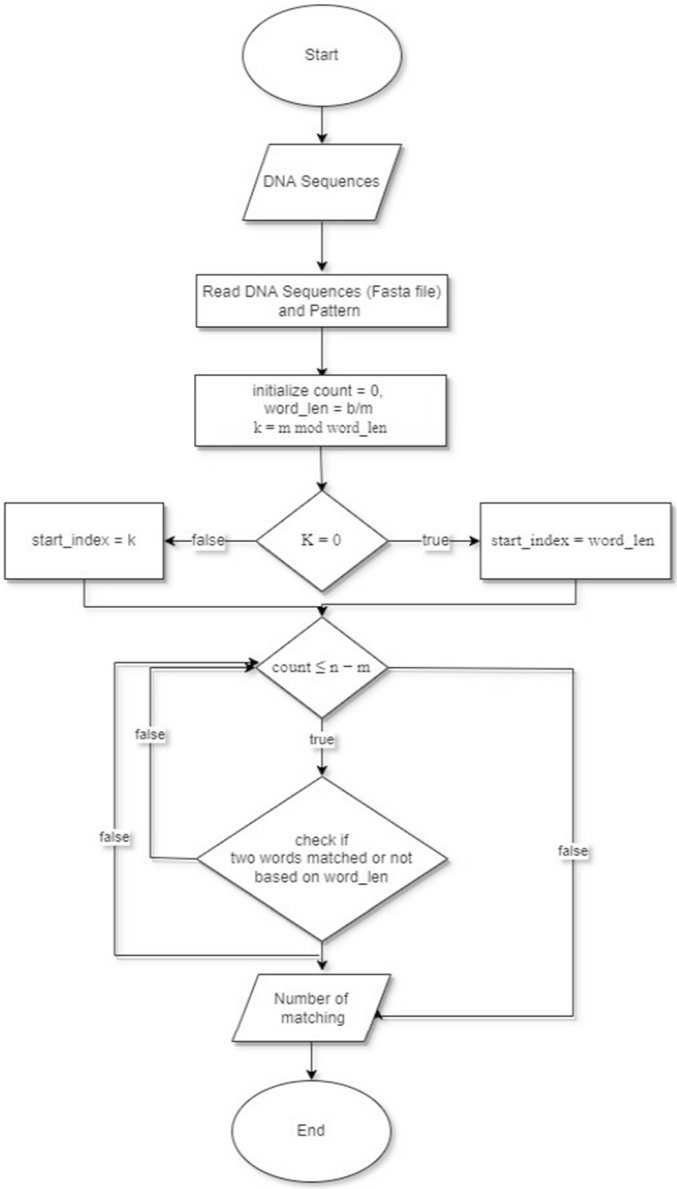

Fig. 8 A flowchart for proposed algorithm EPAPM

Table 3 compares and contrasts the algorithms created in this study with how patterns are matched to DNA sequences using various pattern matching algorithms. Table 3 compares the time it takes to complete the matching process using various methods from previous studies to the proposed algorithms. EFLPM and EPAPM are effective pattern matching algorithms that reduce time when compared to other

Input: Text t and pattern p are stored in the arrays of t[0..n−1] and Pattern p[0..m − 1], respectively, over a finite alphabet ∑.
Output: The beginning index for all occurrences of pattern p in text t.

```
count ← 0
word_len ← b / 8
num_match ← 0
k ← m mod word_len
IF k = 0 THEN
start_index ← word_len
ELSE
start_index ← k
END-IF
WHILE count ≤ n − m DO
    IF t [count…count+word_len] = p[0…word_len-1] THEN
        c ←start_index
        s = count
        WHILE c ≤ m–word_len DO
            IF p[c… c+word_len]  != t[s + c… s +c + word_len] THEN
                Break
            END-IF
            c ← c  + word_len
            IF c == m THEN
                window_index[num_match] ← s
                num_match ← num_match +1
            END-IF
        END-WHILE
    END-IF
    count  ← count + 1
END-WHILE
```

Fig. 9 Pseudocode for EPAPM algorithm

techniques. This demonstrates that the pattern matching time optimization strategy we utilised was successful. EFLPM is the most effective and efficient method in terms of minimising the time required for various pattern sizes, according to the table, and it outperforms other algorithms, particularly FLPM.

Furthermore, EPAPM yielded better outcomes than PAPM, which includes a pre-processing and matching phase. This shows that they are both superior in all pattern scaling matches. In a short length of time, this will solve the problem of matching large patterns with some algorithms and small patterns with others. This is especially advantageous for the expansion of biological data, which is constantly

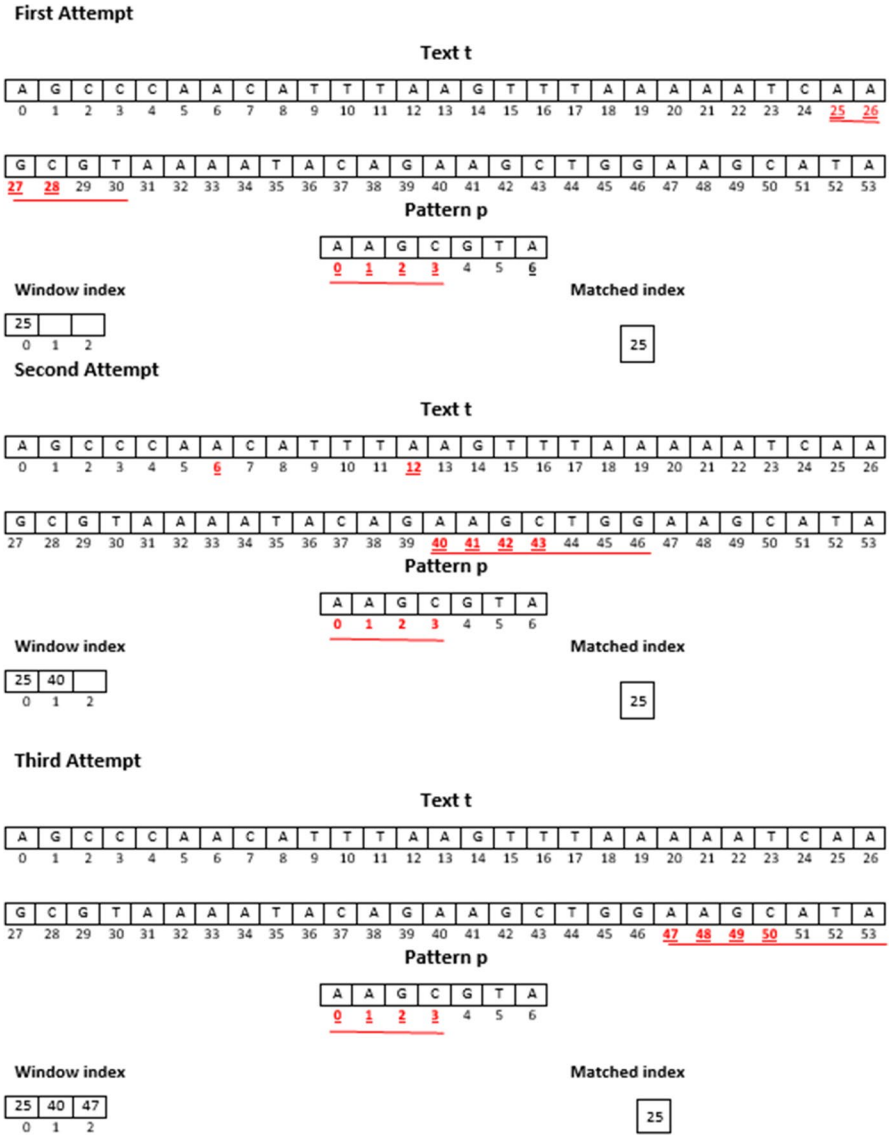

Fig. 10 The operation of the EPAPM algorithm

increasing in volume. The results of the experiments show that the suggested algorithms EFLPM and EPAPM surpass the other simulated algorithms in terms of time cost. This improvement is primarily due to a decrease in the number of identified windows.

The optimized methods have an arithmetic cost of O (n2) for implementing the algorithm in a single process (matching), whereas the FLPM and PAPM algorithms have an arithmetic cost of O (n) for pre-processing and O (n2) for matching. None

Table 3 Pattern matching algorithms time complexity

No	Algorithm	Result time(Sec)							Average time (Sec)	
		Pattern size								
		4	47	344	550	10,000	100,000	1,000,000	Detection rate#	Average rate
1	BM	5.43	3.8	7.44	3.86	3.62	1.55	3.88	100%	4.23
2	Horspool	6.09	3.73	3.95	5.03	4.1	3.33	9.66	100%	5.13
3	Karp-Rabin	7.99	8.73	9.91	9.1	9.69	9.47	8.64	100%	9.08
4	Zhu-Takaoka	6.07	3.41	2.92	2.24	2.12	2.33	3.48	100%	3.22
5	d-BM	8.28	3.72	6.94	4.35	4.7	1.83	3.85	100%	4.81
6	BF	11.67	11.3	11.94	13.1	11.23	12.43	5.5	100%	11.02
7	KMP	5.73	6.97	6.38	7.27	6.12	5.91	9.79	100%	6.88
8	FLPM	4	3.75	3.85	4.16	4.78	4.82	5.27	100%	4.38
9	PAPM	5.92	6.85	6.5	7.83	5.76	6.94	5.62	100%	6.49
10	Proposed EFLPM	2.02	2.01	2.76	2.18	1.22	1.92	1.89	100%	2.00
11	Proposed EPAPM	3.88	4.21	4.05	3.97	4.01	4.04	3.37	100%	3.93

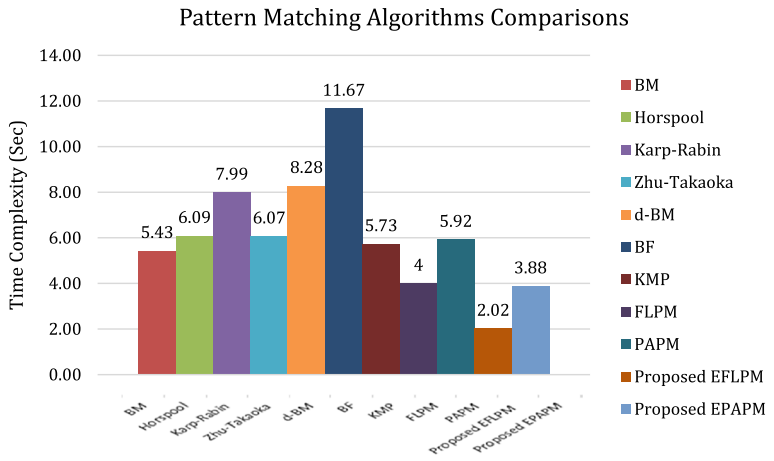

Fig. 11 Pattern matching algorithms comparisons with pattern Length = 4

of the strategies are ineffective or inaccurate, and none of them failed to find the pattern. As a result, the comparison's most important factor was time complexity. All approaches are 100% perfect efficient and accurate, and none of them failed to discover the pattern. As a result, the comparison's primary criterion was time complexity.

Despite the fact that time is an important part of the pattern matching process, we noticed that all of the algorithms used focus on efficiency without considering the time spent in the process. The speed of pattern finding in biological data was the emphasis of this work. As shown in Table 3, as compared to the prior algorithms, the speed of pattern detection increases dramatically. When this ratio was calculated, it was discovered that EFLPM had a 54% faster pattern detection speed than FLPM and EPAPM had a 39% faster pattern detection speed than PAPM. As a result, the proposed algorithms are faster and more efficient than traditional algorithms.

Figure 11 shows the Pattern Matching Algorithms time complexity with the smallest pattern possible containing 4 characters only and search in DNA have sequence size more than 12 million characters, we show that EFLPM and EPAPM are the least time-consuming algorithms.

Figure 12 shows the Pattern Matching Algorithms time complexity with the smallest pattern possible containing 10,000 characters only and search in DNA have sequence size more than 12 million characters, we also show that EFLPM is the least time-consuming algorithm.

Figure 13 shows the Pattern Matching Algorithms time complexity with the smallest pattern possible containing 1,000,000 characters only and search in DNA have sequence size more than 12 million characters, we also show that EFLPM and EPAPM are the least time-consuming algorithms with large pattern size.

Finally, Fig. 14 shows the overall time cost of all algorithms. As seen in this diagram, EPAPM and EFLPM use word processing to significantly reduce the time required to complete pattern matching.

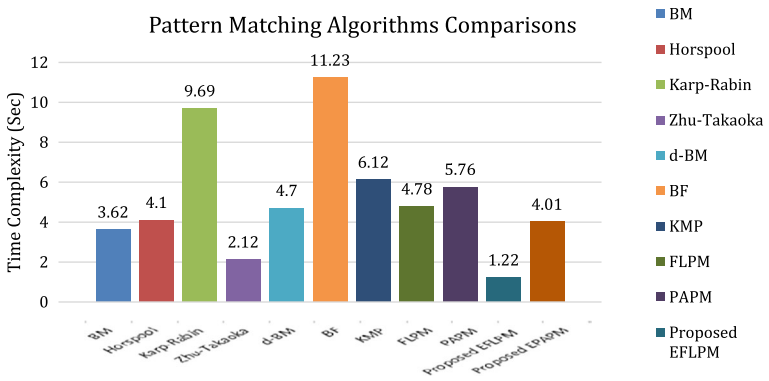

Fig. 12 Pattern matching algorithms comparisons with pattern Length = 10,000

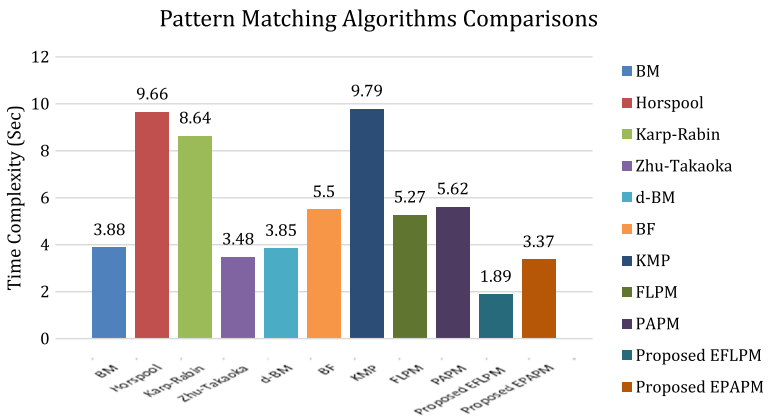

Fig. 13 Pattern matching algorithms comparisons with pattern Length = 1,000,000

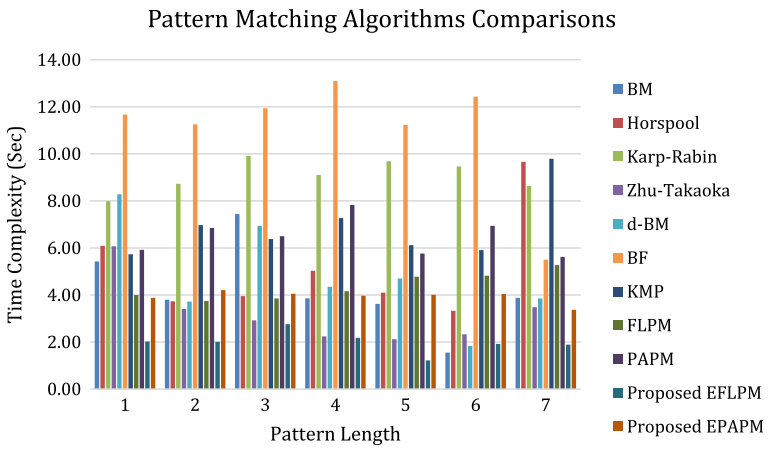

Fig. 14 Pattern matching algorithms comparisons

6 Discussion

In the Real-time world, problems need a quick algorithm with minimum error. Pattern matching is used in a wide range of applications, Pattern matching algorithms have many applications that cover a wide range including Pattern recognition, information retrieval, text processing, and DNA sequence analysis. Pattern matching will help to explore the right and appropriate result. There are many algorithms used to find pattern matching, we focused on DNA Sequences. Nowadays there are many algorithms are used for Pattern matching results. But we found that all the algorithms used focus on efficiency without looking at the time used in the pattern matching process, while the time taken is an effective factor in the matching process. Also, these algorithms consist of two stages, which increases the time spent. And because the time taken is a very influential factor now, especially since we are in the world of speed, we have worked to reduce the time spent with efficiency as well. This paper introduces two improved pattern matching algorithms specifically formulated to speed up searches on large DNA sequences. We also plan to create an efficient approach on real parallel processors in future research to reduce the amount of comparisons and attempts. and employ various techniques, such as machine learning and deep learning, to reduce the time and number of comparisons.

The effect of different factors, such as text size, RAM, and IDE, can be determined on string matching algorithms by designing a factorial model based on factorial design. The trend has already started with a new algorithm for DNS sequence matching by using an MPI technique. Another future direction is to implement string-matches algorithms in GPUs and FPGAs.

Arabic is the second most spoken language in the world after English. In Arabic, connected and unconnected words exist, which take considerable bytes and processing time. Development of multilingual exact matching algorithms with suitable encoding techniques is a promising and interesting future work. Analysis of memory requirements of existing string matching algorithms on heap during runtime is an interesting topic.

7 Conclusion and future work

In this paper, we presented two fast pattern matching algorithms which are EFLPM and EPAPM. The EFLPM and EPAPM algorithms are introduced in this study. The FLPM approach is a character-based pattern matching method, similar to previous studies, but the EPAPM method is a word processing method. The proposed algorithms outperform other simulated algorithms in terms of time cost, according to the outcomes of this study's experiments. Therefore, we noted that the accelerated time increased by 54% for EFLPM and 39% for the EPAPM algorithm. So, the proposed algorithms are quite applicable for pattern matching in biological sequences. This improvement is mostly due to the reduction in

the number of detected windows and the consolidation of the pre-processing and matching steps into a single step. The presentation of a parallel version of current procedures, as well as the use of deep learning techniques in this field, will be the focus of future research. Furthermore, while this study focuses on algorithms that allow for exact pattern matching, future studies could focus on methods that allow for approximate pattern matching.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Data availability The data that support the findings of this study are available in https://github.com/belalahmedhamed/Bioinformatics-Dataset. These data were derived from the following resources available in the public domain of "The National Center for Biotechnology Information advances science and health by providing access to biomedical and genomic information" . https://www.ncbi.nlm.nih.gov/guide/dna-rna/

Declarations

Conflict of interest The authors declare that there is no conflict of interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Montanari P et al (2016) Pattern similarity search in genomic sequences. IEEE Trans Knowl Data Eng 28(11):3053–3067
2. Abrishami V et al (2013) A pattern matching approach to the automatic selection of particles from low-contrast electron micrographs. Bioinformatics 29(19):2460–2468
3. Faro S, Lecroq T (2013) The exact online string matching problem: a review of the most recent results. ACM Comput Surveys (CSUR) 45(2):1–42
4. Tahir M, Sardaraz M, Ikram AA (2017) EPMA: Efficient pattern matching algorithm for DNA sequences. Expert Syst Appl 80:162–170
5. Hakak SI et al (2019) Exact string matching algorithms: survey, issues, and future research directions. IEEE Access 7:69614–69637
6. Sazvar M, Naghibzadeh M and Saadati N (2012) Quick-MLCS: a new algorithm for the multiple longest common subsequence problem. in Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering.
7. Gudur VY, Acharyya A (2018) Hardware-software codesign based accelerated and reconfigurable methodology for string matching in computational bioinformatics applications. IEEE/ACM Trans Comput Biol Bioinf 17(4):1198–1210
8. Amit M et al (2014) Local exact pattern matching for non-fixed RNA structures. IEEE/ACM Trans Comput Biol Bioinf 11(1):219–230

9. Amit M et al. (2012) Local exact pattern matching for non-fixed RNA structures. In: Annual Symposium on Combinatorial Pattern Matching. Springer.
10. Cantone D, Faro S, Pavone A (2019) Linear and efficient string matching algorithms based on weak factor recognition. J Exp Algorithmics (JEA) 24:1–20
11. Deng F, Wang L, Liu X (2015) An efficient algorithm for the blocked pattern matching problem. Bioinform 31(4):532–538
12. Ryu C, Park K (2018) Improved pattern-scan-order algorithms for string matching. J Discret Algorithms 49:27–36
13. Li Z, Yan M, Zhou M (2010) Synthesis of structurally simple supervisors enforcing generalized mutual exclusion constraints in Petri nets. IEEE Trans Syst, Man, Cyber, Part C (Appl Rev) 40(3):330–340
14. Srikantha A et al (2010) A fast algorithm for exact sequence search in biological sequences using polyphase decomposition. Bioinformatics 26(18):i414–i419
15. Hakak S et al (2018) A new split based searching for exact pattern matching for natural texts. PLoS ONE 13(7):e0200912
16. Kim H, Choi K-I (2016) A pipelined non-deterministic finite automaton-based string matching scheme using merged state transitions in an FPGA. PLoS ONE 11(10):e0163535
17. Lee C-L, Lin Y-S, Chen Y-C (2015) A hybrid CPU/GPU pattern-matching algorithm for deep packet inspection. PLoS ONE 10(10):e0139301
18. Otto C et al (2014) ExpaRNA-P: simultaneous exact pattern matching and folding of RNAs. BMC Bioinform 15(1):1–14
19. Al-Ssulami AM, Mathkour H (2017) Faster string matching based on hashing and bit-parallelism. Inf Process Lett 123:51–55
20. Policriti A, Prezza N (2015) Fast randomized approximate string matching with succinct hash data structures. BMC Bioinform 16(9):1–8
21. Ayad LA, Pissis SP, Retha A (2016) libFLASM: a software library for fixed-length approximate string matching. BMC Bioinform 17(1):1–12
22. Knuth DE, Morris J, James H, Pratt VR (1977) Fast pattern matching in strings. SIAM J Comput. 6(2):323–350
23. Raju SV, Reddy K, Rao CS (2018) Parallel string matching with linear array, butterfly and divide and conquer models. Ann Data Sci 5(2):181–207
24. Boyer RS, Moore JS (1977) A fast string searching algorithm. Commun ACM 20(10):762–772
25. Apostolico A, Giancarlo R (1986) The Boyer–Moore–Galil string searching strategies revisited. SIAM J Comput 15(1):98–105
26. Caragiuli K Language processing techniques for searching on Transparenzportal Hamburg.
27. Li H, Durbin R (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics 26(5):589–595
28. Cormen, T.H., et al., *Introduction to algorithms*. 2009: MIT press.
29. Neamatollahi P, Hadi M, Naghibzadeh M (2020) Simple and efficient pattern matching algorithms for biological sequences. IEEE Access 8:23838–23846
30. Murugan A, Punitha K (2021) An efficient DNA sequence compression using small sequence pattern matching. Int J Comput Sci Network Security 21(8):281–287
31. Punitha K. and Murugan A (2021) Pattern Matching Compression Algorithm for DNA Sequences. In: Proceedings of International Conference on Sustainable Expert Systems. Springer.
32. Diwate MRB, Alaspurkar SJ (2013) Study of different algorithms for pattern matching. Int J Adv Res Comput Sci Softw Eng 3(3):1–18
33. Ahmad MK (2014) An enhanced Boyer-Moore algorithm. Middle East University.
34. Liao Y-C (2015) A survey of software-based string matching algorithms for forensic analysis.
35. Shibata Y et al. (2000) A boyer—moore type algorithm for compressed pattern matching. In: Annual Symposium on Combinatorial Pattern Matching. Springer.
36. Singla N, Garg D (2012) String matching algorithms and their applicability in various applications. Int J Soft Comput Eng 1(6):218–222
37. Fainstein J (2005) The application of pattern matching algorithms in bioinformatics: Southern Connecticut State University.
38. Crochemore M and Lecroq TJsfp (2008) A fast implementation of the Boyer-Moore string matching algorithm.
39. Horspool RN (1980) Practical fast searching in strings. Softw Pract Exp 10(6):501–506

40. Padmaveni K, Aravindhar DJ (2021) Improved skip algorithm for single pattern searching. inventive communication and computational technologies. Springer, pp 255–267
41. Chen L, Lu S and Ram J (2004) Compressed pattern matching in dna sequences. In: Proceedings 2004 IEEE Computational Systems Bioinformatics Conference, 2004. CSB 2004. 2004. IEEE.
42. Charras C and Lecroq T (2004) Handbook of exact string matching algorithms. Citeseer
43. Information., N.C.f.B., [Online]. Available: https://www.ncbi.nlm.nih.gov/guide/dna-rna/, 2021