Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

Research Article

Enhanced Hybrid Algorithm for E-AbdulRazzaq and Fast Online Hybrid Matching Algorithms for Exact String Matching

[1] Hayder kamil AL-Jazayiri
Informatics Institute for Postgraduate Studies
Iraqi Commission for Computers and Informatics
ms202130653@iips.edu.iq

[2] Atheer Akram AbdulRazzaq
Business Informatics College, University of Information Technology and Communications,
Baghdad, Iraq
athproof@uoitc.edu.iq

ARTICLEINFO
Article History
Received: 18/12/2023
Accepted: 01/02/2024
Published: 01/06/2024

ABSTRACT
Algorithms for string matching are considered one of the most extensively researched topics in the field of computer science due to their substantial role in various applications, such as information retrieval, editing, security, firewalls, and biological applications. String matching involves examining the optimal alignment by comparing the characters in the pattern and the text. Over the past two decades, it has gained considerable attention due to technological advancements. The need to address string-matching problems has also emerged because of its wide-ranging applications. This study presents the E-ARFO hybrid string-matching algorithm, which combines the best features of two original algorithms, namely, E-AbdulRazzaq and fast online hybrid matching. Compared with other algorithms, the proposed method demonstrates outstanding performance in terms of the number of attempts and character comparisons conducted across multiple databases, including DNA and protein sequences. Results indicate that irrespective of the number of attempts or character comparisons made, E-ARFO consistently ranks first for short and lengthy patterns in most databases. Results also reveal reduced runtimes and competitive character comparisons. Moreover, results underscore the potential effect of E_ARFO on computational biology, offering a new paradigm for precision and efficiency in string matching.

Keywords: ***Exact String-Matching Algorithm, E-ARFO Algorithm, DNA sequence, Protein sequence, Pattern.***

1. INTRODUCTION

String matching is a basic method that identifies all occurrences of a certain pattern, designated as “PATTERN,” in a long string or text called “TEXT.” This computer procedure seeks to find and count all instances when the pattern matches text substrings [1],[2],[3]. In many computer science applications, string matching, which is the act of comparing two finite-length strings and determining their best line, is critical [4]. It has numerous applications, including computational biology [5], intrusion detection systems, operating systems [6], data retrieval, AI, web search engines [7], signal processing, and picture analysis [8], [9]. In addition to its use in reference systems, string matching has many other important applications, including error correction, text processing, speech and pattern recognition, bibliographic search, question–answer applications, DNA pattern matching, protein sequence analysis, and dictionary and knowledge base construction [10], [11]. Recently, a crucial issue that has emerged is the doubling of the amount of string-match able data contained within these databases every two years. Consequently, the demand for efficient string-matching algorithms that can effectively handle the expanded memory size and powerful modern machines is increasing [12], [13]. Meeting this demand is the primary objective pursued by the proposed algorithm. Moreover, the implementation of string-matching techniques must be executed with precision to enhance application efficiency. The efficacy of a string-matching algorithm relies on three crucial factors: the number of attempts, character comparisons, and runtime [14]. The remainder of this paper is carefully organized to help understand the research. Section 2 provides a thorough explanation of string-matching ideas, while Section 3 meticulously

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

formulates the sophisticated hybrid approach. Section 4 details the experimental design and evaluation methodology used. Section 5 describes the implementation details and environmental factors affecting our study, illuminating its practicality. Section 6 discusses and analyzes the results and findings. This section examines algorithm details, runtime, number of attempts, and character comparisons. Section 7 summarizes the findings and conclusions.

2. RELATED WORK

In 1977, the Boyer–Moore algorithm was devised as a proficient algorithm for matching strings that notably amplified the velocity in the pursuit of text. This accomplishment is attained by traversing through text commencing from the far rightmost position and disregarding any character that does not conform to the specified pattern. The algorithm employs two tables, namely, a bad character table and a good suffix table, to establish the utmost feasible shift during each comparison stage. It performs exceedingly well with extensive patterns. The Boyer–Moore algorithm manifests exceptional performance in real-world scenarios, with the average time complexity estimated to be O(n/m). Moreover, it is highly effective for concise and lengthy patterns [15], [16].

The Raita algorithm was developed in 1992, with the objective of identifying the initial occurrence of a pattern within a document. This method is an illustration of a pattern-matching algorithm that is influenced by the use of a bad character table by the Boyer–Moore algorithm, specifically the bmBc table. It compares characters in the text window with those in the pattern at various distances. This algorithm consists of two main stages [17], [18]:

Phase 1. The bmBc table is constructed to enable the algorithm to determine the optimal shifts for each character during the matching process.

Phase 2. This stage is also known as the research phase. The last element of the pattern is compared with the last element of the window. Upon finding a match, the algorithm proceeds to check the first element of the pattern text. If a match is found, it then examines the middle element of the pattern text. If these conditions are met, then it proceeds to check the remaining elements.

The utilization of the bmBc component of the bad character table in the Boyer–Moore algorithm enhances the performance of the Raita algorithm, resulting in a more effective approach for accurate string matching. By employing this method, the matching process can be expedited, leading to a reduction in the time required for character comparison. [19].

An improved hybrid algorithm of the Raita method (i.e., SSABS) was created in 2004. The first step is a preprocessing phase in which the bad character shift table's quick search bad character (qsBc) is constructed, and the second is a search phase in which occurrences of the pattern are located within the text. During the search phase, the qsBc table is crucial to calculating the pattern shift [20]. The search is initiated by focusing on the last character visible in the text pane. The algorithm looks at the window and the pattern's final element. After checking the pattern against the window, it advances to the first matching element. If these two pieces are a match, then it moves on to the next ones. If these two characters are a match, then the algorithm continues to check the next characters from right to left to see if any further correlation exists. If a perfect match is found, then the algorithm determines the presence of a pattern. The algorithm has been tested, and the results obtained are consistently better than those of the Raita algorithm and quick search algorithm.

Although both algorithms share a common preprocessing phase, the ABSBMH algorithm, introduced in 2017, is a vast improvement over its predecessor by using the qsBc approach in the shift operation while maintaining compatibility with the original's preprocessing steps. A novel approach is presented, however, during the search phase of ABSBMH [21]. To achieve similar search efficiency gains, the processing phase in ABSBMH is similar to that of SSABS, with qsBc serving as the basis for dealing with character shifts. When doing a search, ABSBMH uses a one-of-a-kind method. During the research phase of the method, the final element and the next-to-last element are investigated. When these two items are compared and a match is made, the first item is investigated further. If a match is verified, the algorithm continues with left-to-right matching of the remaining items. The ABSBMH algorithm exhibits a lower number of character comparisons in comparison with the outcomes of alternative algorithms while demonstrating minimal disparity in the results of multiple trials. This characteristic can be attributed to the preprocessing phase of the hybrid algorithm, which is founded on the SSABS algorithm, the same preprocessing phase employed by the quick search algorithm. Consequently, given these factors, a slight divergence

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

is observed in the total number of attempts between the results of the ABSBMH algorithm and other algorithms. This discrepancy arises from the utilization of distinct pattern lengths, which are randomly selected from the databases.

The FLPM algorithm was proposed in 2020. The preprocessing part of this algorithm requires searching for repeats of the first and last characters between the text and the pattern; therefore, it can be thought of as a relatively sluggish algorithm. The number of possible window matches between the pattern and the text is calculated using this search. The method checks the remaining elements against the pattern throughout the search phase. Moreover, it is dependent on the number of windows discovered; if a discrepancy is found, then the algorithm moves on to investigate another window [22], [23]. FLPM finds matching windows based on the text pattern's beginning and ending character locations during preprocessing. These windows' remaining components are meticulously compared during the search phase to locate matches. Window count affects algorithm iteration. If a mismatch occurs, then the software checks the next window. FLPM can assess matching windows by employing the initial and final characters in this particular design. The algorithm exhibits satisfactory performance in terms of the number of attempts made; however, it incurs a considerable computational cost and necessitates numerous character comparisons. In situations where the pattern and text features are shared, this method guarantees precise match detection.

The purpose of our research is to improve the E-AbdulRazza and FOHM algorithms to improve string-matching performance. This paper explains our algorithm methodology, results, and discussion, explaining the effectiveness of our enhanced hybridization of algorithms in different applications.

3. METHODOLOGY

This paper describes an improved hybrid model incorporating E-AbdulRazzaq mixed with FOHM. Subtle changes include modifying these two algorithms to produce a hybrid algorithm that is characterized by speed and accuracy by combining the good features of the two algorithms above while conducting a comparison test with other algorithms with the same specific relevant performance indicators that serve as a measurement tool (average running time, number of comparisons, and number of attempts). The FOHM algorithm consists of two stages. Creating the qsBc table is part of the preprocessing phase. In the search phase, the pattern is divided into three components, and the three components are analyzed by comparing the pattern with the text window. The first three elements of the form are compared with the text window, and then the last three elements are compared. If a match is selected, the algorithm continues to compare the remaining characters within the form and text window. The method moves the window to the right by an amount taken from the qsBc table if no match is found at any stage. The E-AbdulRazzaq algorithm has two distinct phases. In the preprocessing phase, two tables, i.e., bmbc and brbc, are created. In the search phase, the hash is first examined for elements in primary positions, and only then is it checked for composite locations. If the two hashes are identical, then the algorithm checks each individual element to determine if it occupies a prime or composite [24].

4. PROPOSED ENHANCED HYBRID ALGORITHM

The study introduces the enhanced E-AbdulRazzaq-FOHM (E_ARFO) algorithm. The E_ARFO algorithm is an advanced hybrid algorithm that integrates the capabilities and methodologies of the E-AbdulRazzaq and FOHM algorithms. The primary objective of this methodology is to deliver effective and precise string matching while simultaneously enhancing the performance of different stages within the algorithm. The proposed algorithm is a fusion of techniques derived from two primary sources: modification of the enhanced E-AbdulRazzaq algorithm and the FOHM algorithm. The algorithm is divided into two main phases: preprocessing and search. The following is a detailed breakdown of each phase:

4.1 Preprocessing Phase

The preprocessing phase of the E_ARFO method incorporates essential approaches derived from the E-AbdulRazzaq algorithm. The aforementioned strategies are systematically structured into functions to establish a comprehensive preprocessing phase that is specifically designed for precise string matching. The following are the key functions utilized in this phase:

The Boyer–Moore bad character (bmBc) function is a key component of the Boyer–Moore algorithm, as shown in Eq. (1), which is widely used for string searching.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

$$bmB_C[c] = \begin{cases} \min\{i:1\le i<m-1 \text{ and } x[m-1-i]=c\} & \text{if } c \text{ occurs in } x \\ m & \text{otherwise} \end{cases} \quad (1)$$

The Berry–Ravindran bad character (BRBC) function is a computational algorithm [25] used in string matching, as shown in Eq. (2). The E_ARFO algorithm, influenced by the Berry–Ravindran algorithm, integrates the Brbc function. The functionality of this function is analogous to that of the Brbc function, with the key distinction being its emphasis on the selection of the maximum shifting value throughout the matching process. The method uses the Brbc table to ensure appropriate shifting of the window, hence enhancing performance.

$$brBc[u,v] = \min \begin{cases} 1 & \text{if } p[m-1]=u, \\ m-i+1 & \text{if } p[i]\,p[i+1]=uv, \\ m+1 & \text{if } p[0]=v, \\ m+2 & \text{otherwise} \end{cases} \quad (2)$$

Calculation of tables: This phase is when two important tables, i.e., bmBc and brBc, are calculated. These tables of importance in the shift operation draw inspiration from the E-AbdulRazzaq algorithm. Hash calculation: Finally, the algorithm computes hash values for the first three and last three letters in the pattern.

Pseudocode of Preprocessing phase

Algorithm E_ARFO (X [0 …..m−1]

```
//Input: Pattern X
//Output: Shift tables of (bmBc), (brBc) and compute the hush values.
//pre brBc (preprocessing Berry–Ravindran bad-character function)
brBc[ASIZE][ASIZE] //2D array to keep shift values
    For q ← 0 to ASIZE  Do
        For s ← 0 to ASIZE  Do
            brBc[q][s] ← m+2
        End For
    End For
    For q ← 0 to ASIZE   Do
            brBc[q][x[0]]  ← m +1
    End For
    For p ← 0 to m−2 Do
```

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

14. brBc[x[p]][x[p+1]] ← m – p
15. **End For**
16. For q ← 0 to ASIZE Do
17. brBc[x[m−1]][q] ← 1
18. **End For**
19. //pre bmBc (preprocessing Boyer–Moore bad- character function)
20. For s ← 0 to size of alphabet Do
21. bmBc [s] ← m
22. **End For**
23. For s ← 0 to m−2 Do
24. bmBc [X[s]] ← m− s−1
25. **End For**
26. // Compute the hash values h = d^S−1 mod q
27. For i ← w to S−1 Do
28. hy ← (hy<<1) +y[i]
29. **End For**
30. first Ch x[0], second Ch x[1], third Ch x[2]
31. last Ch x[m−1], second last Ch x[m−2] , third last Ch[m−3]
32. // Hash values of all steps in pattern and the first and last three characters in text window
33. fhx ← (fhx<<1)+ first Ch , fhx ← (fhx<<1)+ second Ch, fhx ← (fhx<<1) + third Ch
34. fhy ← (fhy<<1)+ y[0] , fhy ← (fhy<<1)+ y[1] , fhy ← (fhy<<1) + y[2]
35. Lhx ← (Lhx<<1)+ last Ch, Lhx ← (Lhx<<1)+ second last Ch, Lhx ← (Lhx<<1) + third last Ch
36. Lhy ← (Lhy<<1)+ y[m−1] , Lhy ← (Lhy<<1)+ y[m−2] , Lhy ← (Lhy<<1) + y[m−3]

4.2 Search Phase

Window alignment: The search phase begins by comparing the pattern with the beginning of the text. A window is generated by extracting a portion of the text, specifically a segment that is of the same length as the pattern. The comparison is thereafter performed in a sequential manner, starting from the left and progressing toward the right. An algorithm computes hash functions for first three items and final three characters from a window of text for present window.

Hash comparison (first three): In this case, the algorithm will compare the first three hashes of the pattern with those that come on a text window.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

Hash comparison (last three): Additionally, the third last three hashes in the pattern are compared with those that occurred in the last three hashes of the text window.

Character comparison: The first three characters of the pattern will be compared with the first three characters of the text box. If these characters' match, the last three characters of the pattern will be compared with the last three characters of the text box. If these characters also match, the rest of the characters of the pattern will be compared with the Text Box.

The shift operation is initiated when some of the steps (specifically steps four to eight) lead to a mismatch in the algorithm. To address this, the algorithm proceeds to the shift stage, where the pattern is shifted toward the right side by a predetermined number of elements. This shift value is determined by selecting the maximum value from two tables, namely, bmBc and brBc. Thus, the proposed algorithm incorporates various preprocessing techniques inspired by E-AbdulRazaq, along with a search approach derived from FOHM. This combination offers a robust mechanism for conducting string matching and provides a systematic approach to pattern matching within extensive amounts of text data.

Pseudo code of Search phase

Algorithm E-ARFO (X [0 …..m−1], Y 0…….n−1])

```
1.  //Input: Pattern X, Text Y
2.  //Output: number of attempts and number of  character comparisons of pattern with text and
      the consumed time
3.  j← 0
4.  While j <= n − m Do
5.     c ←y[j + m – 1]
6.    // Comparing the Fh and Lh
7.   If (fhx == fhy && Lhx == Lhy)
8.      if  first Ch == y[j] && second Ch ==y[j + 1] && third Ch== y[j + 2] ) Then
9.        if last Ch == c  && second last Ch==y[j+m− 2] && third last Ch== y[j+m−3] ) Then
10.         // match
11.         If (match(x + 2, m−3, y, j + 2,j+m−3, &temp) == 1) Then
12.         Count  //The first occurrence of the pattern  in the text
13.       End If
14.     End If
15.   End If
16. End If
17.  Output the first attempt and character comparisons and the consumed time
18. //shifting//
19.  j + = max (brBc[y [j + m]][y[j+m+1]],bmBc [y[j + m−1]])
20. // Rehash operation for the text window
```

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

21 . fhy ← 0,fhy ← (fhy<<1)+ y[0] , fhy ← (fhy<<1)+ y[1] , fhy ←(fhy<<1) + y[2]

22 . Lhy ← 0, Lhy ←(Lhy<<1)+ y[m−1], Lh (Lhy<<1)+ y[m−2] , Lhy ←(Lhy<<1) + y[m−3]

5. DATABASES

This study examines the varying performance and characteristics of the string-matching algorithms when applied to different types of databases, each having a substantial data size of 230 MB. The method was evaluated using data derived from DNA and proteins [26], [27]. Then, the information was gathered from the United States' National Centre for Biotechnology Information. The datasets included in this study were classified into distinct groups, considering the duration of the patterns. The category consisted of concise patterns, ranging in length from eight to 1,024 characters.

6. IMPLIMENTATION AND ENVIRONMENT

The hybrid algorithm and other algorithms were implemented on a device with the following technical specifications: Intel(R) Core(TM) i7-8665U processor, which exhibits exceptional performance in complex computations; 16.0 GB RAM, 500 GB hard drive; Windows 10 operating system. For the implementation, Python was designated as the preferred programming language.

7. RESULTS AND ANALYSIS

7.1 Implementation Evaluation And Result Analysis Of The E-Arfo Algorithm Compared With The Original Algorithms

This section presents the results, which offer an overall view of the efficiency of the E_ARFO, E-AbdulRazzaq, and FOHM algorithms under different character lengths for DNA sequencing. The DNA sequence average execution time is shown in Figure 1. E_ARFO is on top of the average runtime compared with E-AbdulRazzaq and FOHM for each character length. This result suggests that the E_ARFO algorithm is more efficient at handling DNA sequences than the other two. Additionally, E_ARFO has lower runtimes with the increased character length, demonstrating the efficiency of processing longer datasets. A critical parameter that shows the computational effort required by each method is the number of character comparisons. Figure 2 displays the amount of character comparisons performed by the E_ARFO algorithm for various character lengths. Regarding several character comparisons, E_ARFO compares competently with that of E-AbdulRazzaq, with both exhibiting comparable feats. By contrast, FOHM continuously entails more character comparison than the required number, making it a relatively less efficient pattern recognition among DNA sequences.

The number of attempts is an excellent measure for measuring the pattern identification efficiency of the algorithms, as illustrated in Figure 3, which shows how little iterations there were for E_ARFO. Thus, it is extremely good for finding patterns in DNA sequences. However, E-AbdulRazzaq made numerous attempts. Regarding this, the E-AbdulRazzaq algorithm requires as much attempts as ARFO. By contrast, FOHM needs many more attempts to obtain the same outcomes. DNA pattern matching appears to be improved by E_ARFO and E-AbdulRazzaq. Running times, character comparisons, and attempts taken altogether imply the advantage of the E_ARFO method for the problem of sequence alignment.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

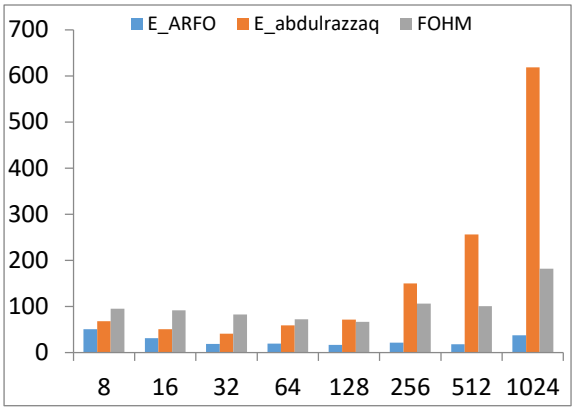

Fig. 1. Average runtime using the DNA database

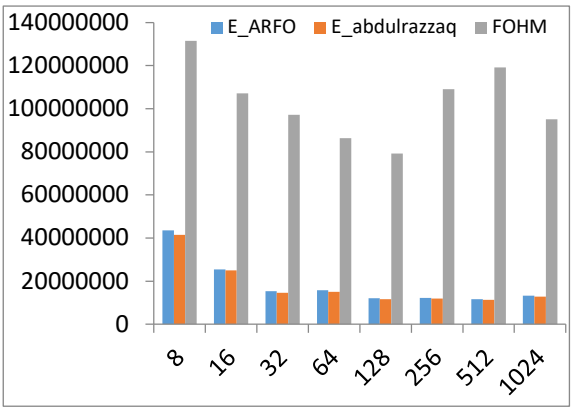

Fig. 2. Number of character comparisons using the DNA database

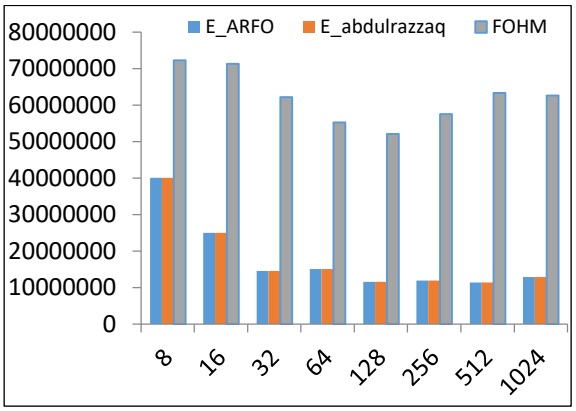

Fig. 3 Number of attempts using the DNA database

Figure 4 shows the typical execution time when a protein sequence is used. The comparison of average run time for proteins shows that all these algorithms are essentially different. By contrast, E_ARFO beats E-AbdulRazzaq and FOHM in all string lengths. The E_ARFO program has a decreasing run time with growing character length, which supports the program's capability of working on long protein sequence datasets. The reduction in E_ARFO's running times shows that this tool would be effective when processing large protein sequences with many features as needed. As far as the number of character comparisons is concerned (illustrated in Figure 5), E-AbdulRazzaq and E_ARFO have equivalent results when it comes to determining patterns in protein sequences. By contrast, the FOHM algorithm requires more frequent comparisons between characters, revealing its less efficient protein sequence detection strategy.

Another point that reinforces the effectiveness of E_ARFO and E-AbdulRazzaq is the number of attempts made by the algorithms (Figure 6). The number of attempts is almost equal for both algorithms at different amino acid lengths, reflecting a high speed of pattern finding in protein sequences. By contrast, FOHM calls for much more trials, indicating its weaknesses on the effective alignment of protein sequences. A comparison of characters and attempts performed by E_ARFO demonstrates an optimal performance in protein sequence alignment.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

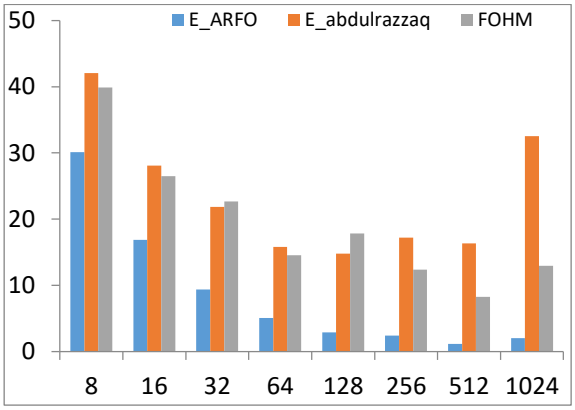

Fig. 4. Average runtime using a protein sequence

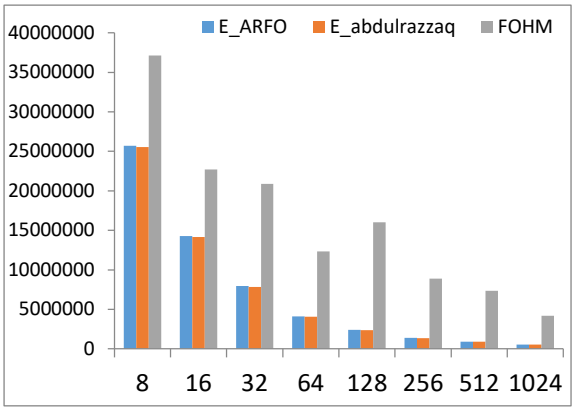

Fig. 5. Number of character comparisons using a protein sequence

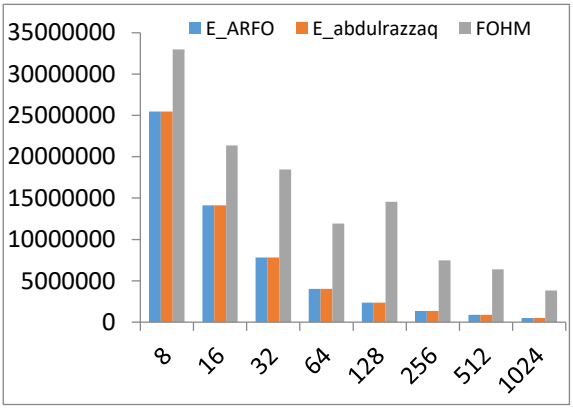

Fig. 6 Number of attempts using a protein Sequence

7.2 Evaluation And Result Analysis Of The E-ARFO Algorithm Compared With The Standard And Recent Algorithms

To evaluate the performance of our enhanced hybrid algorithm, we conducted a comprehensive comparison with existing string-matching algorithms, including Boyer–Moore, quick search, Raita, SSABS, ABSBMH, Ibrahim, and FLPM. All algorithms were tested on the same collection of text and pattern datasets. Key performance metrics, which researchers have used to evaluate algorithms, include attempts, i.e., instances repeated until a suitable pair is found; comparisons, i.e., the number of character comparisons performed throughout the matching process; and match, i.e., the total time consumed to find pattern matches in a text. Below is a description of our tests, a comparison of the improved hybrid algorithm to the current methods, and a discussion of their ramifications.

The results offered in this study offer valuable insights into the performance of several string-matching algorithms across a range of character lengths (ch). The purpose of this discussion is to examine the obtained results thoroughly and conduct a comprehensive analysis of the essential components.

Table 1 shows that the E_ARFO algorithm performs better than the other considered algorithms for the analysis of DNA sequences. It is more effective than competitors in Boyer–Moore, quick search, Raita, SSABS, ABSBMMH, FLPM, and Ibrahim across different length patterns between eight and 1,024. E_ARFO has low average runtimes, which means that it processes short and long DNA sequences efficiently. E_ARFO is more effective in terms of scalability because its run times also decrease with an increase in the pattern lengths. Table 2 illustrates that in terms of character comparison, E_ARFO is great and surpasses most search algorithms, such as Boyer–Moore, quick search, Raita, SSABS, ABSBMH, FLPM, and Ibrahim. E_ARFO repeatedly delivers less number of character comparison, which suggests that it is good in recognizing existing patterns within DNA chains. This efficiency is valid for short and long pattern lengths, stressing that the algorithm has minimum computational needs. Table 3 shows the high quantity of E_ARFO attempts, which indicates greater efficiency with similar or lower numbers compared with other baseline techniques. Boyer–Moore, quick search, Raita, SSABS, ABSBMH, FLPM, and Ibrahim usually have several attempts more when compared with E_ARFO, proving that the latter does an excellent job in fast searching and recognizing a match. The algorithm proves its robustness by maintaining such efficiency for

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

different pattern lengths. A similar trend can be found when analyzing the protein sequences as E_ARFO beats Boyer–Moore, quick search, Raita, SSABS, ABSBMH, FLPM, and Ibrahim,

As shown in Table 4. The algorithm has a low average runtime on various protein sequence lengths, demonstrating its effectiveness.

Reducing runtime as the pattern length increases suggests an ability of the tool to process long sequences or patterns. E_ARFO also competes with several benchmark algorithms in the character comparison tests. It clearly performs different functions depending on the length of the pattern that involves protein sequences, as illustrated in Table 5. A low number of character comparison highlights proving itself as the most reliable method in terms of string matching. the effectiveness of E_ARFO against complexities associated with protein sequence matching. In the analysis of protein sequences, Table 6 shows that the number of times attempted by E-ARFO is either the same or less than that of other algorithms. This result implies that E_ARFO performs efficiently when locating and aligning protein sequence patterns. The consistent performance of E_ARFO at various pattern lengths reinforces the trustworthiness of the program. Furthermore, E_ARFO outperformed the other algorithms in DNA and protein sequence analyses,

TABLE I. AVERAGE TIME CONSUMPTION OF THE E-ARFO ALGORITHM COMPARED WITH RECENT AND STANDARD ALGORITHMS WHEN USING SHORT AND LONG PATTERN LENGTHS AND 230 MB DATA SIZE OF DNA DATABASE

Algorithm name	Pattern length							
	8	16	32	64	128	256	512	1024
E-ARFO	47.154	40.853	16.881	24.856	18.652	19.689	19.801	19.483
Boyer–Moore	112.743	113.670	102.222	47.450	66.885	48.275	40.276	29.837
Quick search	57.477	44.266	38.232	34.869	31.644	33.899	38.605	32.416
Raita	78.740	48.451	47.487	42.825	35.848	46.808	59.291	27.417
SSABS	95.807	67.669	60.864	53.214	50.835	59.559	75.679	74.416
ABSBMH	81.402	57.474	50.822	44.870	41.836	49.884	61.820	51.008
FLPM	128.820	91.876	92.356	92.447	92.611	87.692	89.681	111.661
Ibrahim	157.289	118.488	119.877	122.422	121.226	103.843	104.497	138.889

TABLE II. NUMBER OF CHARACTER COMPARISONS FOR THE E-ARFO ALGORITHM COMPARED WITH RECENT AND STANDARD ALGORITHMS WHEN USING SHORT AND LONG PATTERN LENGTHS AND 230 MB DATA SIZE OF DNA DATABASE.

Algorithm name	Pattern length							
	8	16	32	64	128	256	512	1024
E-ARFO	43505433	25480008	15269871	15844021	11999374	12251059	11663221	9106589
Boyer–Moore	139583205	153829319	142413901	66449178	89771755	75277434	58625325	41023149
Quick search	99253097	98402350	85739481	79372990	72041463	72868751	80302186	68985914
Raita	129318035	94864743	93076559	82796186	68525210	95704192	106296138	49250641
SSABS	125234286	106802900	95388009	84104449	78230248	97114770	106356654	85007319
ABSBMH	131984027	107412431	96862143	85251578	78649001	104280969	114121634	85592424
FLPM	345937074	336080084	336538628	339320436	336852583	314237398	314261845	347907438
Ibrahim	339269803	339033031	338800664	354507264	340454395	312230331	312364294	345220957

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

TABLE III. NUMBER OF ATTEMPTS FOR E-ARFO AGAINST RECENT AND STANDARD ALGORITHMS WHEN USING SHORT AND LONG PATTERN LENGTHS AND A 230 MB DATA SIZE OF DNA DATABASE

Algorithm name	Pattern length							
	8	16	32	64	128	256	512	1024
E-ARFO	40112407	24949863	14573571	15067929	11556133	11931774	11356090	8732300
Boyer–Moore	96746582	122647089	112994118	51858207	71795055	51264563	40663144	29277301
Quick search	72232235	71255469	62127865	55214308	52051917	57559893	63291601	49049340
Raita	90534902	75321340	72711900	64186027	54257553	68690983	76577132	36026680
SSABS	72232235	71255469	62127865	55214308	52051917	57559893	63291601	49049340
ABSBMH	72232235	71255469	62127865	55214308	52051918	57559893	63291601	49049340
FLPM	21355015	13722928	14146739	13891259	13958057	14222206	14176386	21306141
Ibrahim	72614972	72614970	72614964	72614953	72424380	48914457	48914328	72424234

TABLE IV. AVERAGE TIME CONSUMPTION OF THE E-ARFO ALGORITHM COMPARED WITH RECENT AND STANDARD ALGORITHMS WHEN USING SHORT AND LONG PATTERN LENGTHS AND 230 MB DATA SIZE OF PROTEIN SEQUENCE DATABASE

Algorithm name	Pattern length							
	8	16	32	64	128	256	512	1024
E-ARFO	29.451	23.736	9.165	6.453	3.967	2.268	1.549	1.247
Boyer–Moore	229.558	155.620	148.616	125.627	174.298	122.825	100.847	152.602
Quick search	22.821	11.828	10.242	6.645	7.745	4.931	3.564	4.281
Raita	27.541	13.368	11.466	7.666	9.483	4.722	4.420	5.448
SSABS	30.547	15.443	13.628	8.063	10.687	5.777	5.604	6.385
ABSBMH	28.867	14.525	12.581	8.629	10.280	5.342	5.230	6.069
FLPM	90.647	67.409	70.149	66.657	67.495	67.647	68.571	75.693
Ibrahim	81.854	65.839	78.484	74.609	75.225	74.400	77.467	84.888

Table V. Number Of Character Comparisons For The E-Arfo Algorithm Compared With Recent And Standard Algorithms When Using Short And Long Pattern Lengths And 230 Mb Data Size Of Protein Sequence Database

Algorithm name	Pattern Length							
	8	16	32	64	128	256	512	1024
E-ARFO	25718933	14283197	7941016	4120680	2401470	1375525	895949	552585
Boyer–Moore	232108905	187345623	184987396	150037223	212245166	153983822	120879033	174283185
Quick search	35374116	23314223	20421161	12615660	15148875	7749799	6760284	7455468
Raita	39375707	23051136	20518920	12726632	16178910	8343667	6650141	7680099
SSABS	37149739	22606925	20973118	12294479	15920693	8809960	7298842	7895741
ABSBMH	37121856	22690387	20868132	12309667	15992852	8873128	7320759	7881768
FLPM	262633691	258263694	270625094	258581301	258013347	258223607	261868960	257917962
Ibrahim	262840327	258639370	271519743	259016070	257794833	258259629	261816437	258084870

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

Table VI: Number Of Attempts For E-Arfo Against Recent And Standard Algorithms When Using Short And Long Pattern Lengths And 230 Mb Data Size Of Protein Sequence Database

Algorithm name	Pattern Length							
	8	16	32	64	128	256	512	1024
E-ARFO	25459195	14139251	7841113	4049334	2370659	1364439	880518	546945
Boyer–Moore	219201934	182604690	175505731	148173369	203863168	143258918	114915223	167396938
Quick search	32989276	21382028	18479897	11924776	14541698	7473724	6406753	7152774
Raita	36967086	22398090	19148218	12510131	15427425	7621713	6217205	7296716
SSABS	32989276	21382028	18479897	11924776	14541698	7473724	6406753	7152774
ABSBMH	32989276	21382028	18479897	11924776	14541698	7473724	6406753	7152774
FLPM	829164	269932	1349998	146024	518965	701759	789483	424815
Ibrahim	14610739	10844001	22003528	11255803	10336669	10336664	13896782	10336628

8. CONCLUSION

A thorough assessment of string-matching algorithms, including an evaluation of the E-ARFO algorithm, presents a sophisticated understanding of how these algorithms operate when working with characters of varying lengths in protein and DNA strands. When it comes to DNA sequences, E_ARFO leads the pack by being superior in runtime, character comparisons, and attempts. Despite increasing character length, it manages to have a low runtime, confirming its computing capacity. This algorithm exhibits high competitiveness in the process of comparing letters, especially when compared with E-AbdulRazzaq. This result demonstrates the algorithm's effectiveness in quickly spotting patterns within DNA-related sequences. In addition, the similar number of attempts of E_ARFO and E-AbdulRazzaq indicates that they have the same efficiency in solving complexities involved with DNA sequences. Although FOHM exhibits some competencies, it remains slower than E_ARFO.

Extending the scope to protein sequences further bolsters E_ARFO's supremacy. The algorithm outperforms its opponents with shorter runtime and fast character comparison for different character lengths. The decreasing runtime as the complexity of protein sequences increases highlights the scaling property of this algorithm to accommodate big and complex datasets. Parallel character comparison and attempts by these findings go beyond algorithmic comparisons. E_ARFO is also faster and more accurate in the potential for pattern recognition for genomic studies or other computational areas.

References

[1] P. Mahmud, A. Rahman, and K. H. Talukder, "An Efficient Hashing Method for Exact String Matching Problems," 2022. doi: 10.1007/978-981-16-6460-1_21.

[2] M. Bicer and X. Zhang, "An Efficient, Hybrid, Double-Hash String-Matching Algorithm," in 2019 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2019, 2019. doi: 10.1109/LISAT.2019.8816827.

[3] S. Faro and T. Lecroq, "The Exact Online String Matching Problem: a Review of the Most Recent Results", ACM computing survey, vol. 45(2), pp.1-42, 2013.

[4] M. G. Osamah S . Alrouwab, "Evaluating Efficiency of Some Exact String-Matching Algorithms on Large-Scale Genome," American Journal of Computer Science and Information Technology, vol. 9, no. 112, 2021.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

[5] A.W. Mahmood, N.A. Abdul Rashid, and A.A. AbdulRazzaq, "BM-KMP HYBRID ALGORITHM FOR EXACT AND SUBSEQUENCE STRING MATCHING', Proceeding of the 3rd International Conference on Informatics and Technology, Informatics '09,pp. 81-87, 2009.

[6] A. A. AbdulRazzaq, N. A. Abdul Rashid, A. A. Abbood, and Z. Zainol, "The Improved Hybrid Algorithm for the Atheer and Berry-Ravindran Algorithms," International Journal of Electrical and Computer Engineering(IJECE), vol. 8, no. 6, pp. 4321-4333, 2018, doi: 10.11591 /ijece .v8i6.pp4321-4333.

[7] A.A. AbdulRazzaq, N.A. Abdul Rashid, M. A. Abu-Hashem, and A. A. Hasan, "A New Efficient Hybrid Exact String Matching Algorithm and Its Applications", LifeScience Journal (Life Sci J), vol. 11(10), pp.474-488, 2014.

[8] S. Al-Dabbagh and N. Barnouti, "A New Efficient Hybrid String Matching Algorithm to Solve the Exact String Matching Problem," British Journal of Mathematics & Computer Science, vol. 20, no. 2, pp. 1–14, Jan. 2017, doi: 10.9734/bjmcs/2017/30497.

[9] A.A. AbdulRazzaq, N.A. Abdul Rashid, H. B. Y. Hamdani, R. M. Ghadban, and A.W. Mahmood, "Influenced Factors on Computation Among Quick Search, Two-Way and Karp-Rabin Algorithms", Proceeding of the 3rd International Conference on Informatics and Technology, Informatics '09, pp. 100-106, 2009.

[10] A. A. Karcioglu and H. Bulut, "The WM-q multiple exact string matching algorithm for DNA sequences," Comput Biol Med, vol. 136, Sep. 2021, doi: 10.1016/j.compbiomed.2021.104656.

[11] A. A. Karcioglu and H. Bulut, "Improving hash-q exact string matching algorithm with perfect hashing for DNA sequences," *Comput Biol Med*, vol. 131, Apr. 2021, doi: 10.1016/j.compbiomed.2021.104292.

[12] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, "Exact String Matching Algorithms: Survey, Issues, and Future Research Directions," *IEEE Access*, vol. 7, 2019, doi: 10.1109/ACCESS.2019.2914071.

[13] A.A. AbdulRazzaq, N. A. Abdul Rashid, w. A. Hasan, M.A. Abu-Hashem, and Z. Zainol "New Searching Technique of Hybrid Exact String Matching algorithm". International Review on Computers and Software(I.RE.CO.S.), vol.11 (10), pp. 884-897, 2017.

[14] A. Fadlil, S. Sunardi, and R. Ramdhani, "Similarity Identification Based on Word Trigrams Using Exact String Matching Algorithms," *INTENSIF: Jurnal Ilmiah Penelitian dan Penerapan Teknologi Sistem Informasi*, vol. 6, no. 2, 2022, doi: 10.29407/intensif.v6i2.18141.

[15] P. Pangestu and S. E. Wahyuningrum, "WORD SEARCH USING BOYER-MOORE ALGORITHM," *Proxies : Jurnal Informatika*, vol. 2, no. 1, 2021, doi: 10.24167/proxies.v2i1.3195.

[16] J. Allmer, "Exact pattern matching: Adapting the Boyer-Moore algorithm for DNA searches," 2016. doi: 10.7287/peerj.preprints.1758v1.

[17] A. A. Almazroi et al., "A Hybrid Algorithm for Pattern Matching: An Integration of Berry-Ravindran and Raita Algorithms," in Lecture Notes on Data Engineering and Communications Technologies, 2022. doi: 10.1007/978-3-030-98741-1_15.

[18] R. Rahim et al., "Searching Process with Raita Algorithm and its Application," in Journal of Physics: Conference Series, 2018. doi: 10.1088/1742-6596/1007/1/012004.

Iraqi Journal for Computers and Informatics
Information Technology and Communications University
Vol. 50, No.1 , 2024, pp. 20-33
DOI: https://doi.org/10.25195/ijci.v50i1.452
Print ISSN: 2313-190X, Online ISSN: 2520-4912

[19] T. Raita, "Tuning the Boyer-Moore-Horspool String Searching Algorithm," 1992.

[20] Al-Dabbagh, Sinan Sameer Mahmood, et al. "Parallel quick search algorithm for the exact string matching problem using OpenMP." Journal of Computer and Communications 4.13 (2016): 1- 11

[21] T. Islam and K. H. Talukder, "An improved algorithm for string matching using index based shifting approach," in 20th International Conference of Computer and Information Technology, ICCIT 2017, 2018. doi: 10.1109/ICCITECHN.2017.8281772.

[22] B. A. Hamed, O. A. S. Ibrahim, and T. Abd El-Hafeez, "Optimizing classification efficiency with machine learning techniques for pattern matching," J Big Data, vol. 10, no. 1, 2023, doi: 10.1186/s40537-023-00804-6.

[23] O. A. S. Ibrahim, B. A. Hamed, and T. A. El-Hafeez, "A new fast technique for pattern matching in biological sequences," Journal of Supercomputing, vol. 79, no. 1, 2023, doi: 10.1007/s11227-022-04673-3.

[24] A. A. Abdulrazzaq, N. A. Abdul Rashid, and A. M. Taha, "The Enhanced Hybrid Algorithm for the AbdulRazzaq and Berry-Ravindran Algorithms," International Journal of Engineering & Technology, vol. 7, no. 3,pp. 1709-1717, 2018, doi: 10.14419/ijet.v7i3.12436.

[25] T. Berry and S.A. Ravindran, "fast string matching algorithm and experimental results", In Proceedings of the Prague Stringology Club Workshop`99, J. Holub and M. Simáneked, Collaborative, Report DC-99-05, pp. 1-17, 1999.

[26] RefSeq: NCBI Reference Sequence Database, "DNA Dataset," Homo sapiens isolate CHM13 chromosome 2, alternate assembly. Accessed: Nov. 19, 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/nuccore/NC_060926.1.

[27] RefSeq: NCBI Reference Sequence Database, "Protein Dataset," RefSeq: NCBI Reference Sequence Database. Accessed: Nov. 19, 2023. [Online]. Available: https://ftp.ncbi.nih.gov/refseq/daily/