International Journal of Engineering & Technology, 7 (3) (2018) 1709-1717

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i3.12436
Research paper

The Enhanced Hybrid Algorithm for the AbdulRazzaq and Berry-Ravindran Algorithms

Atheer Akram Abdulrazzaq [1]*, Nur'Aini Abdul Rashid [2], Ahmed Majid Taha [3,4]

[1] Senior Lecturer, Businesses Informatics College, University of Information Technology and Communications, Baghdad, Iraq
[2] Associate Professor, College of Computer & Information Sciences, Princess Nourah bint Abdulrahman University, KSA
[3] Senior Lecturer, Businesses Informatics College, University of Information Technology and Communications, Baghdad, Iraq
[4]Soft Computing and Data Mining Center, Universiti Tun Hussein Onn, Johor, Malaysia
**Corresponding author E-mail: athproof@uoitc.edu.iq*

Abstract

Exact string matching is one of the critical issues in the field of computer science. This study proposed a hybrid string matching algorithm called E- AbdulRazzaq. This algorithm used the best properties of two original algorithms; AbdulRazzaq and Berry-Ravindran Algorithms. The proposed algorithm showed an efficient performance in the number of attempts and number of character comparison when compared the original and recent to the standard algorithms. The proposed algorithm was applied in several types of databases, which are DNA sequences, Protein sequences, XML structures, Pitch characters, English texts, and Source codes. The Pitch database was the best match for E-AbdulRazzaq with the number of attempts involving long and short patterns, while the DNA database was the worst match. No data is specified as the best or worst with the E-AbdulRazzaq algorithm in terms of the character comparisons. The E-AbdulRazzaq algorithms ranked first in most databases when using short and long patterns, in terms of number of attempts and character comparisons.

Keywords: *Exact String Matching Algorithms; E-AbdulRazzaq Algorithm; Database Type.*

1. Introduction

String matching is a searching operation carried out to check the optimal alignment by comparing two finite-length strings. The function of string matching used in many computer science applications, such as computational biology, intrusion detection system, operating systems, information retrieval, artificial intelligence, web search engine, and signal and image processing [1]. Other examples of string matching applications are library systems, error correction, text processing, speech and pattern recognition, bibliographic search, question-answer applications, analyze Protein sequences and pattern matching of DNA, and in the literature of dictionaries and memorized data [2].

The database duplicated in string matching per two years; therefore, to perform multiple tasks with increasing memory size and speed of computers must use an efficient string matching algorithm [1]. String matching depends on two strings, which are text and pattern, both of them subject a matching operation to identify their identical characters [2]. The efficient algorithm depends on two factors: the number of attempts and the number of character comparison [3]. In this paper, the proposed hybrid algorithm depends on good properties of two hybrid exact string matching algorithms and overcome the drawbacks of these algorithms which are AbdulRazzaq and Berry-Ravindran. All types of databases in benchmark standard are used to determine the suitable and unsuitable databases with this algorithm. The aim of this paper is to enhance the performance of existing string matching algorithms.

The proposed algorithm in this research depends on two original algorithms which are AbdulRazzaq and Berry-Ravindran. AbdulRazzaq algorithm selected the good features of two algorithms that are Smith, and Karp-Rabin and it depends on new searching technique [1]. There are five steps preprocessing of the AbdulRazzaq algorithm which are, Prime and composite numbers functions, the Boyer-Moore bad character (bmBc) step, the quick search bad character (qsBc) step and the hashing step. In the searching phase of the AbdulRazzaq algorithm, there are three steps. First, the hashing of the prime numbers in the text window is calculated using hashing equation and the value is compared with the equivalent hashing value of the prime number characters in the pattern. If matching is obtained between them, then the prime number characters of text window and pattern are compared one by one. Second step, if matching is occurred, then compared the hashing characters in the pattern and text window. The hashing of composite number characters in the text window is calculated by the hashing equation. When matching is obtained in the hashing value then the pattern and text window characters are compared. If matching occurred between these characters then third step is performed. In the third step the character number one in the pattern is compared with that in the text window. When a matching or mismatching occurs, the shifting of the pattern depend on the maximum value between the m value in the bmBc table and on the m+1 value in the qsBc table [1].

The Berry-Ravindran algorithm is a hybrid of the Zhu-Takaoka and Quick-Search algorithms and characterized by left-right character comparisons [4]. This algorithm has two phases, which are preprocessing and searching. Preprocessing phase depends on the Berry-Ravindran bad character (brBc) function, while searching phase depends on the shifting process of next two characters in window of text (m+1 and m+2). The comparison operation starts from leftmost character of pattern with leftmost character of the text window. When a match is get, the comparison will continue to next character

Copyright © 2018 Atheer Akram AbdulRazzaq et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

until obtain matching for all characters. When matching or mismatching occurs the shifting process depends on the next two characters of text window (m+1 and m+2) and the shifting value obtained from the brBc table in the preprocessing phase.

2. Proposed E-AbdulRazzaq algorithm

The proposed E-AR algorithm consists of two phases, namely, the preprocessing phase and searching phase.

2.1. Preprocessing phase

The preprocessing phase in the E-AR algorithm contains the same techniques selected from the AbdulRazzaq and Berry-Ravindran algorithms. These techniques are regulated in functions to obtain the preprocessing phase in the exact string matching of the E-AR algorithm. These functions are presented as follows:

2.1.1. Prime and composite numbers functions

The technique used in these functions is selected from the preprocessing phase of the AbdulRazzaq algorithm, and these functions are considered the initial step. The first function is related to the division of numbers to prime and composite numbers (Lines 5−10; as shown in Figure 1), whereas the second function is related to the calculation of the prime and composite numbers in the string. Each number is placed in a separate set from another set (Lines 12−18; as shown in Figure 1).

2.2.2. Boyer-moore bad character (bmBc) function

The technique used in this function is mentioned in the preprocessing phase of the current AbdulRazzaq algorithm. The E-AR algorithm uses this function in the preprocessing phase to prepare the bmBc table. The main purpose of using the bmBc table is to determine and choose the best shifting for each character in the matching operation.

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

2.2.3. Berry-Ravindran bad character (Brbc) function

This function is derived from the preprocessing phase of the Berry-Ravindran algorithm. The brBc table used the same technique in the E-AR algorithm. The technique of the brBc function is to determine and choose the maximum value in the shifting process (Lines 20–34, as shown in Figure 1).

$$brBc[u, v] = \min \begin{cases} 1 & \text{if } P[m-1] = u, \\ m-i+1 & \text{if } P[i]\ P[i+1] = uv, \\ m+1 & \text{if } P[0] = v, \\ m+2 & \text{otherwise.} \end{cases} \tag{2}$$

2.2.4. Hashing step

This function is used in the AbdulRazzaq algorithm and in the hashing technique of the E-AR algorithms. The hashing technique of enhanced hybrid algorithm depends on two steps which are, first the calculating the hash for the characters that have prime numbers in the pattern and are denoted by Ph1, while the second depends on calculating the hash for characters that have composite numbers in the pattern and are denoted by Ch2. The hashing values will calculated by using hashing equation. (Lines 44−50; as shown in Figure 1).

```
1.  E-AbdulRazzaq Algorithm (X [0 …..m−1])
2.  //Input: Pattern X
3.  //Output: prime and composite numbers, Shift tables of bmBc),
      (qsBc) and compute the hush values of (pv) and (npv).
4.  //Divide the set numbers to prime and composite
5.  For d  2 to d*d <= S Do
6.        If S % d == 0 Then
7.           Output the composite number
8.        End If
9.  End For
10.       Output the prime number
11. // Calculate set numbers of prime and composite in the pattern
12. For  K  1 to m−1 Do
13.       If (p(k+1)) Then
14.          pv[s]  <= k
15.       Else
16.          pv[s]  <= k
17.       End If
18. End For
19. // prebrBc (preprocessing Berry-Ravindran bad-character function)
20. brBc[ASIZE][ASIZE]  //2D array to keep shift values
21. For k  0 to ASIZE Do
22.       For j   0  to ASIZE Do
23.          brBc[k][j]  m+2
24.       End For
25. End For
26  For k 0 to ASIZE Do
27.      brBc[k][x[0]]  m +1
28. End For
29. For i  0 to m−2 Do
30.      brBc[x[i]][x[i+1]]  m − i
31. End For
32. For k  0 to ASIZE Do
33.      brBc[x[m−1]][k]  1
34. End For
35. //prebmBc (preprocessing Boyer-Moore bad-character function)
36. For j  0 to size of alphabet Do
37.      bmBc[j]  m
38. End For
39. For j  0 to m−2 Do
40.      bmBc [X[j]]  m− j−1
41. End For
42. // compute the hush values h = d^S−1 mod q
43. //calculate the hash values of (pv) and (npv) of pattern characters
44. c ← x[0]
45. For r ← 0 to Sp−1 Do
46.      hx1   (hx1<<1) + x[pv[r]]
47. End For
48. For r ← 0 to  Snp−1 Do
49.     hx2   (hx2<<1) + x[npv[r]]
50. End For
```

Fig. 1: Preprocessing Phase in the E-AbdulRazzaq Algorithm

2.3. Searching phase

The searching phase technique in the E-AbdulRazzaq algorithm depends on the searching phase techniques of the AbdulRazzaq and Berry-Ravindran algorithms and on some of changing on the matching process. There are three steps in the searching phase which are first, the hashing value of the prime numbers in the text window is calculated (Line 6, as shown in Figure 2). The hashing of the prime numbers in the text window is calculated using hashing equation. The hashing of prime number characters in the text window is denoted by (Phw1) and compared with the equivalent hashing value of the prime number characters (Ph1) in the pattern. If matching is obtained between them, then each value is individually compared with the prime number characters in the text window and pattern (Line 8, as shown in Figure 2). If matching is obtained between these characters, then the second step is performed. However, If a mismatch is obtained in the hashing comparison or in the character comparison the new shift of this algorithm will depend on the

maximum value of m from the bmBc table and the (m + 1 and m + 2) values from the brBc table (Line 20; Figure 2). The m refers to the last character in the text window, the m + 1 is the first character after the text window, and m+2 is the second character after the text window.

In the second step, if matching is obtained in the first step, the hashing value of the composite numbers in the text window is calculated (Line 9, as shown in Figure 2) using hashing equation, and the hashing values of the hashing characters (Ch2) in the pattern and hashing characters (Chw2) in the text window are then compared. The hashing of the composite number characters in the text window is denoted by (Chw2). If matching occurs among the hashing values, then the characters of the pattern and the text are compared. If matching is found among these characters, then the third step (L3) is performed (Line 11, as shown in Figure 2). If mismatching occurs between them, then shifting depends on the same technique used in the previous step.

In the third step, when matching is obtained in the second step, the character number one in the pattern (L3) is compared with the character number one in the text window (Lw3) (Line 13, as shown in Figure 2). If matching obtained between the first character of the pattern and the same character position in the text window, then the shifting depends on the same technique used in the previous steps. If mismatching is obtained, then shifting depends on same technique used in the previous step (Line 20, as shown in Figure 2).

```
1.  E-AbdulRazzaq Algorithm (X [0 …..m−1], Y [0…….n−1])
2.  //Input: Pattern X, Text Y
3.  //Output: number of attempts and number of character comparisons
      of pattern with text
4.  j ← 0
5.  While j <= n – m Do
6.          hy1 ← gethy(j,y,1) //calculate the hash of (Phw1)
7.          // Comparing the Ph1 and Phw1
8.          If (hx1 == hy1 && match(x,m,y,j,&temp,1) == 1) Then
9.              hy2 ← gethy(j,y,0) // calculate the hash of (Chw2)
10.             // Comparing the Ch2 and Chw2
11.             If (hx2 == hy2 && match(x,m,y,j,&temp,0) == 1) Then
12.                 // Comparing the L3 and Lw3
13.                 If y[j] == c Then
14.                     Count //The first occurrence of the pattern
                            in the text
15.                 End If
16.             End If
17.         End If
18.         Output the first attempt and character comparisons
19.         //shifting//
20.         j +=max((brBc[y [j + m]][y[j+m+1]],bmBc[y[j + m − 1]])
21. End While
```

Fig. 2: Searching Phase in the E-AbdulRazzaq Algorithm.

3. Complexity of the E-AR algorithm

In the preprocessing phase of E-AbdulRazzaq there are five functions which are; dividing the set numbers into prime and composite, calculating set numbers of prime and composite, brBc, bmBc and hash function. Time complexity dividing set numbers into prime and composite function is O ($s^{\wedge 1/2}$), calculating set numbers of prime and composite function is O (m + ($2^{\wedge 1/2}$), + ($3^{\wedge 1/2}$), + ... + ($m^{\wedge 1/2}$)) = O (m), brBc function is O ($m + \sigma^2$), the bmBc function is O (m + σ) and hash function is O (m). The space complexity and time complexity of preprocessing in the E-AbdulRazzaq algorithm is O ($m + \sigma^2$). The next section explains the time complexity of the searching phase.

Lemma 1: *The searching time complexity in the best case is O (n).*

Proof. When each character does not occur in the pattern in the matching process, then the shafting depends on the maximum value between m and (m + 1and m+2) from bmBc and brBc functions that computed in preprocessing phase. The time complexity of the computing the hash values of prime numbers (line 6; Figure 2) in the best case, O (n-m+1) / (m+2) ×m=O (n) and the comparisons of the hash values will be repeated (n-m+1)/ (m+2) times. However, the computing hash values of composite numbers (line 9; Figure 2) and the matching operation of the characters (line 11; Figure 2) equal 0 and 0, respectively. When all characters in the pattern differ than those in the text window, then the (j) will be increased m + 2 (line 20; Figure 2) and the body of the while loop is repeated (n-m+1)/ (m+2) times, and the overall time complexity is O (n).

For example: Text: yyyyyyyyyyyyyyy
Pattern: xxxx

Lemma 2: *The time complexity of the searching phase in worst case is O (n × m).*

Proof. In the matching process every character in the text does not take place more than m times and all the character comparisons for n characters of the text do not exceeded than n × m. The calculating of the hash values two times in (Lines 6 and 9; Figure 2) will be repeated (n-m+1) and (n-m+1), respectively, while time complexity is Θ(m) in line 6 and is Θ(m) in line 9. In addition, the matching operation (Lines 8 and 11; Figure 2) will be repeated (n-m+1) and (n-m+1), respectively. Moreover, the time complexity is Θ(m) in line 8 and Θ(m) in line 11. Thus, the total time complexity is O (Θ(m) × Θ(n) + Θ (m) × Θ(n) + Θ(m) × Θ(n) + Θ(m) × Θ(n)) = O (nm).

The worst case occurs when all the characters in the pattern and text window are totally similar in each attempt. Then the (j) will be increased 1 (line 20; Figure 2) and the time complexity will be O (n × m). All paragraphs must be indented. All paragraphs must be justified, i.e. both left-justified and right-justified.

4. Experimental design and evaluation of study outcomes

Design of proposed algorithm depended on choosing the good properties of original algorithms, which are the prime and composite numbers functions, hash and bmBc functions from AbdulRazzaq algorithm and brBc function from Berry-Ravindran algorithm. The proposed algorithm used all types of benchmark standard databases and the results of proposed algorithm compared to results of the original and recent and standard algorithms.

4.1. Databases

This research discuss the differences of string matching algorithms in performance and features when used various types of database (200 MB data size). Benchmark standard of databases used different types of data, such as DNA, Protein, XML Pitch, English text, and Source code. These databases were downloaded from the Pizza & Chili Corpus Web site (http://pizzachili.dcc.uchile.cl/ (Pizza Chili Corpus). There are two pattern lengths used in this study, which are: the short pattern length, which ranged from 4 characters to 28 characters, and the long pattern length (length power of two), which ranged from $2^{\wedge 5}$ characters to $2^{\wedge 10}$ characters [5], [6]. The DNA data sequence is consists of four nucleotides, namely, Adenine, Guanine, Cytosine, and Thiamine, and these types are encoded as A, G, C, and T respectively. [7]. The Proteins are necessary to the structure and function of the cells of an organism. The Protein data sequence is consists of 20 amino acids arranged in a linear series and encoded using uppercase letters [8], [9]. The XML structure database contains the bibliographic information of computer sciences. The Pitch (Midi Pitch values) database contains tuning data used in digital music [10]. The English text database contains all the characters in the English alphabet. The Gutenberg project has established this database [7]. The Source program code database is consists of all the characters used in the C and Java languages [11].

4.2. Implementation and environment

The experiment was run on the Al Biruni cluster in the PDCC lab of the School of Computer Science, USM (Biruni.cs.usm.my), using Ubuntu Linux 10.04, 4LTS of 64-bit with NVIDIA CUDA Toolkit v2.2 and GNU C Compiler (GCC) v4.4.3. The secure shell (SSH) software was utilized in accessing the Biruni cluster to implement the codes. The abbreviated forms of the algorithms were used: BR for Berry-Ravindran, AR for AbdulRazzaq, which are the original algorithms. The recent and standard algorithms included Horspool (H), Quick search (QS), Two-way (T-W), Fast search (FS), SSABS (SS), TVSBS (TV), AKRAM (AK), and Maximum shift (MS). E-AbdulRazzaq (E-AR).

For elucidation the results in the number of attempts made when the proposed hybrid algorithm was compared with the original, recent, and standard algorithms, the following parameters were set: logarithmic scale and base, 10; display units, 10000; minimum number, 100000. In the number of characters compared for the original, standard, and recent algorithms, the logarithmic scale and base (10) and the display units (10000) were applied. The evaluation tables for the hybrid algorithm were arranged with the best result presented first, followed by other algorithms results. The results were expressed as "first," "second," and "third." To evaluate proposed algorithm performance in various types of databases, the result was regarded as "best" when the hybrid algorithm performed better in a specific database compared with others. "Worst" was used to indicate the lowest performance of the hybrid algorithm for a given database. "All databases" was used to indicate when the hybrid algorithm or the other algorithms attained the best performance in all databases. "Most databases" was used to indicate that the hybrid algorithm or the other algorithms attained the best performance in most but not all databases.

5. Evaluation and results analysis of the E-AR algorithm compared with the original algorithms

5.1. Results and analysis on number of attempts by using a 200 MB data size

The E-AR obtained the best results as compared with the BR and AbdulRazzaq algorithms in short and long pattern lengths. The Pitch database shows the best results in number of attempts when using short and long patterns. The DNA database shows the worst results in both short and long patterns (Figures 3 and 4).

5.2. Results and analysis on number of characters comparison by using a 200 MB data size

The E-AR obtained the best results as compared with the AbdulRazzaq and BR algorithms in short and long pattern lengths. In using short and long pattern lengths, no specific data is provided to determine the best in number of character comparisons. Furthermore, no specific data is provided to determine the worst database by using short patterns. However, Pitch is considered the worst by using long patterns (Figures 5 and 6).

6. Evaluation and results analysis of E-AR algorithm compared with the recent and standard algorithms

6.1. Results and analysis of the number of attempts when using a 200 MB data size

The E-AR algorithm performs the best in all types of databases, ranking second only in the DNA algorithm, when a short pattern length is used. The Maximum shift algorithm showed the best performance in all databases when using a long pattern, The E-AR algorithm is second best algorithm in all data types. The Two-way algorithm shows the worst performance in both short and long patterns (Figures 7 and 8).

6.2. Results and analysis of the number of character comparison when using a 200 MB data size

The E-AR algorithm performs the best in all types of databases, ranking second only in the Pitch and English databases, when using short patterns. This algorithm shows the best performance in long pattern lengths in all databases but is only the second best algorithm in the English database. The Two-way algorithm shows the worst performance in both short and long patterns (Figures 9 and 10).

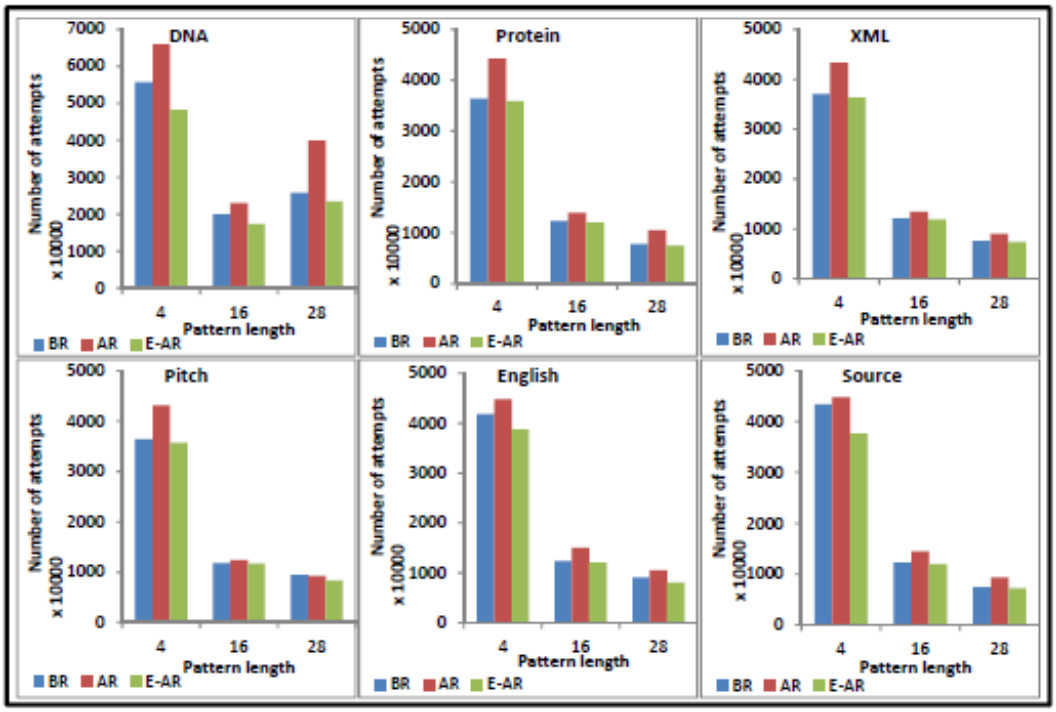


Fig. 3: Number of Attempts for E-AR against Original Algorithms when Using A Short Pattern and A 200 MB Data Size.

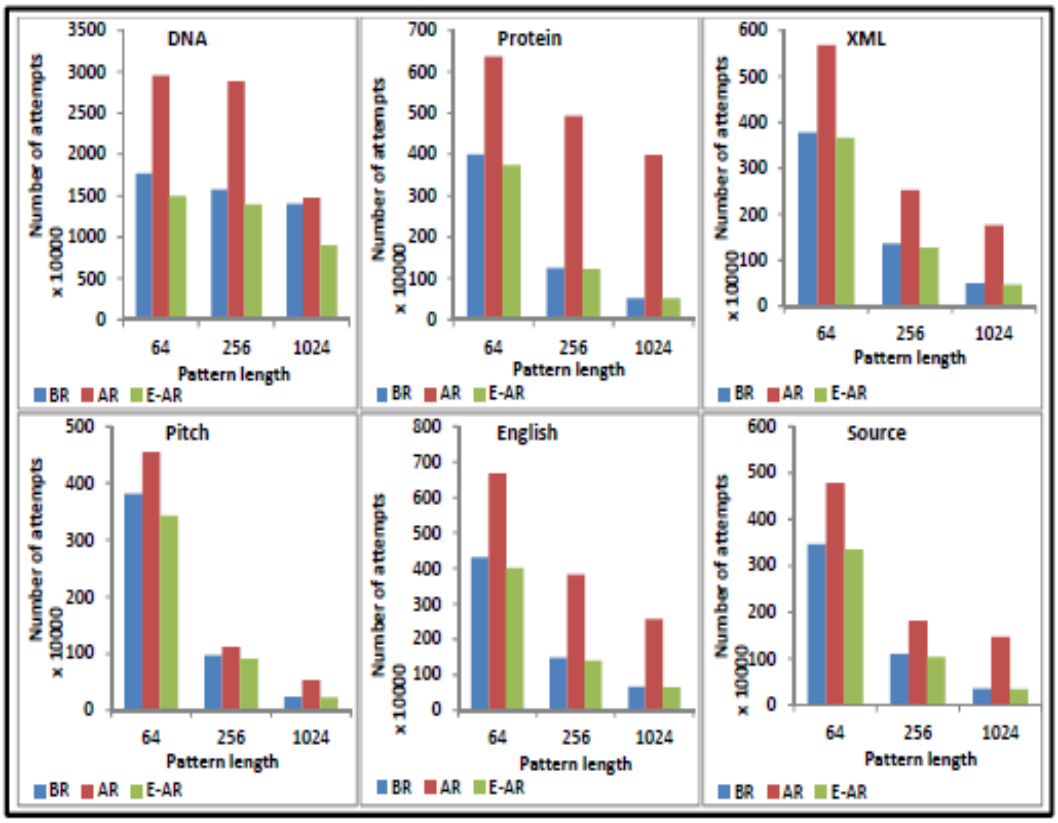


Fig. 4: Number of Attempts for E-AR against Original Algorithms when Using A Long Pattern and A 200 MB Data Size.

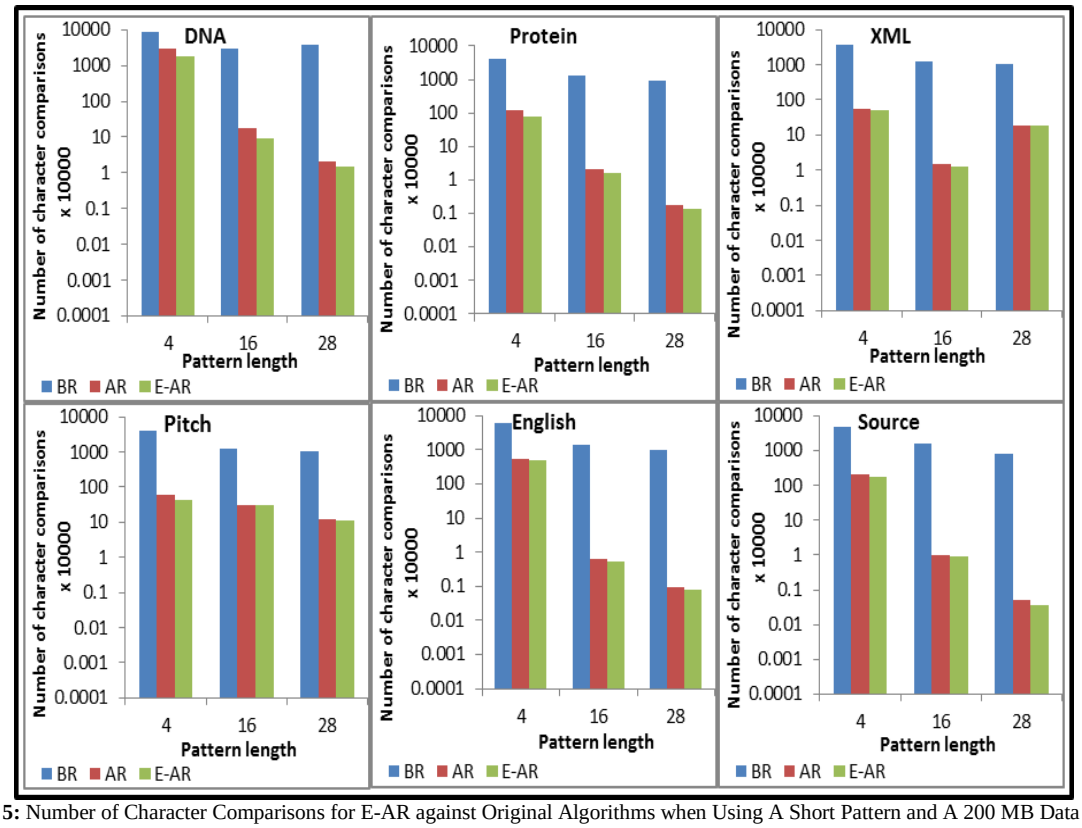


Fig. 5: Number of Character Comparisons for E-AR against Original Algorithms when Using A Short Pattern and A 200 MB Data Size.

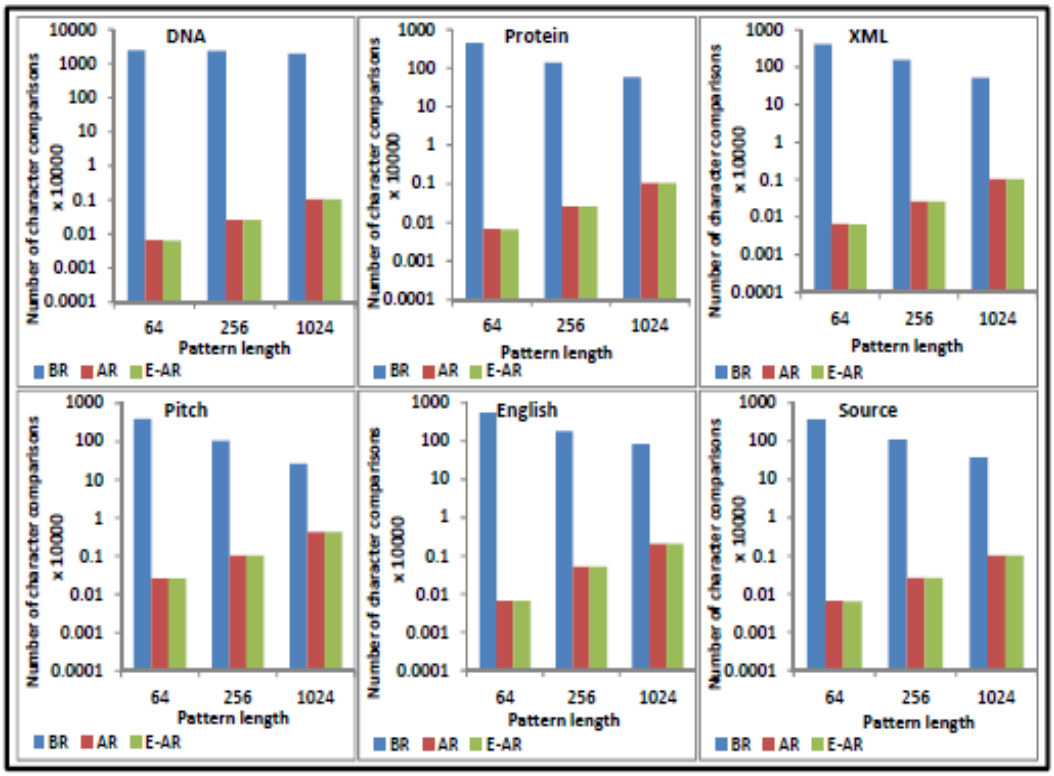


Fig. 6: Number of Character Comparisons for E-AR against Original Algorithms when Using A Long Pattern and A 200 MB Data Size.

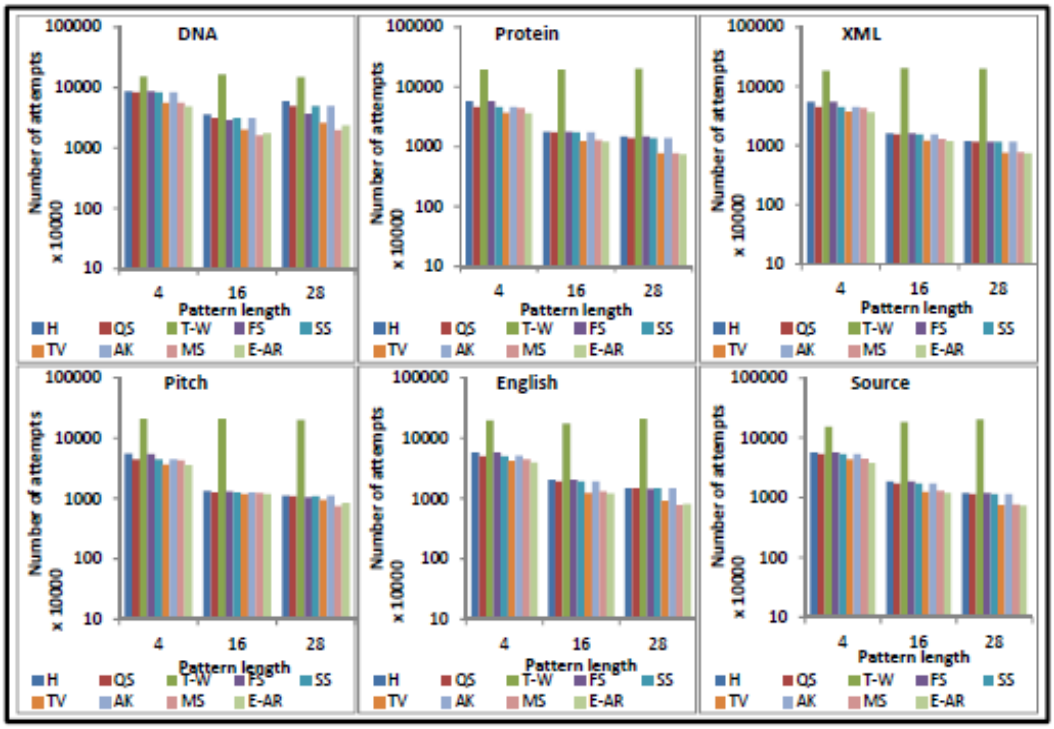


Fig. 7: Number of Attempts for the E-AR Algorithm Compared with the Recent and Standard Algorithms when Using A Short Pattern and A 200 MB Data Size.

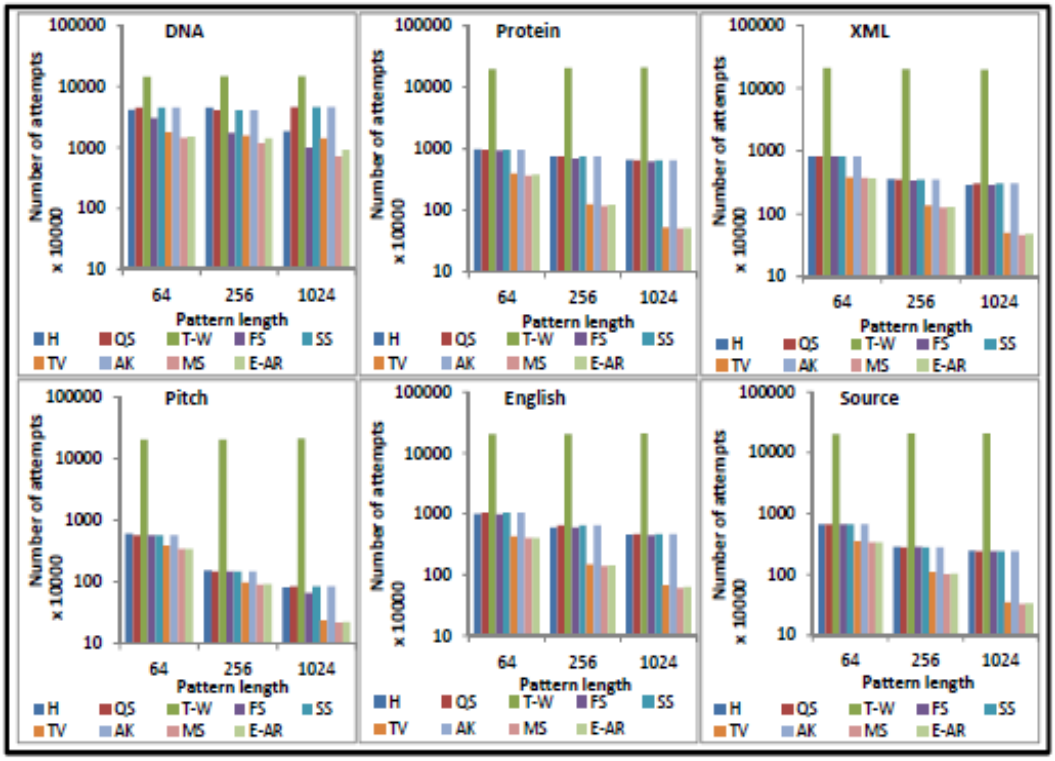


Fig. 8: Number of Attempts of the E-AR Algorithm Compared with the Recent and Standard Algorithms when Using A Long Pattern and A 200 MB Data Size.

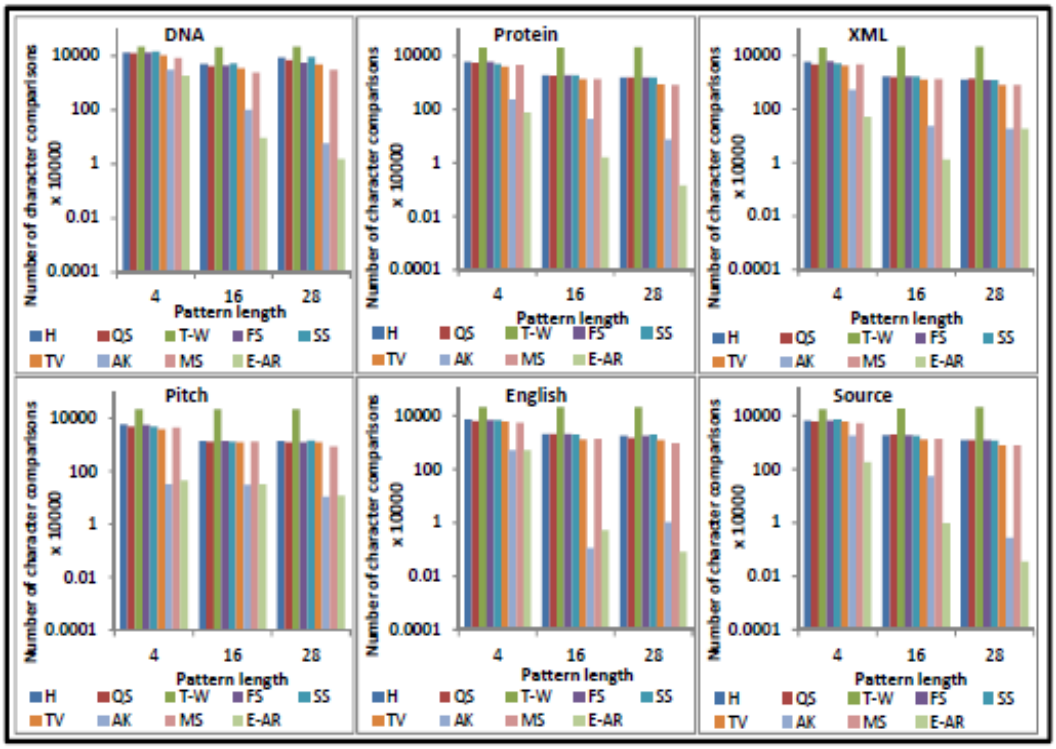


Fig. 9: Number of Character Comparisons for the E-AR Algorithm Compared with the Recent and Standard Algorithms when Using A Short Pattern and A 200 MB Data Size.

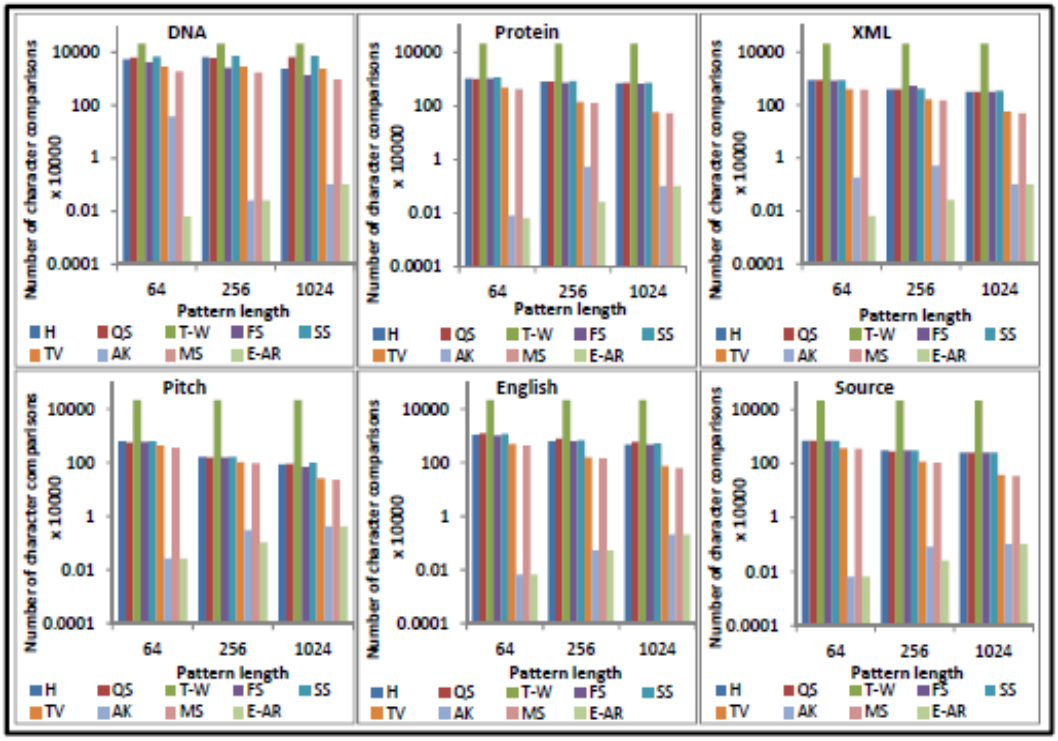


Fig. 10: Number of Character Comparisons for the E-AR Algorithm Compared with the Recent and Standard Algorithms when Using A Long Pattern and A 200 MB Data Size.

7. Discussion and analysis

7.1. Evaluation of the E-AR algorithm compared with the original algorithms

The performance results of the E-AR algorithm and the original algorithms are compared in terms of the number of attempts and character comparisons when using short and long pattern lengths with different data types and sizes.

Table 1: Comparison of the Results of the E-AR Algorithm and the Original Algorithms

Algorithms	Data size (MB)	
	200	
	Short	Long
Best performance in number of attempts		
Berry-Ravindran	Second	Second
AbdulRazzaq	Third	Third
E-AR	First	First
Best performance in number of character comparisons		
Berry-Ravindran	Third	Third
AbdulRazzaq	Second	Second
E-AR	First	First

The E-AR algorithm shows the fewest number of attempts because it depends on the best shifting function by obtaining the maximum value between the good shifting functions of brBc and bmBc. The algorithm obtains the lowest number of character comparisons because this algorithm depends on the hash function, which is considered an efficient function for obtaining the fewest character comparisons [12], and on the good shifting of the brBc function (Table 1.).

Table 2: Performance of the E-AR Algorithm in Different Database Types

Performance	Databases		
	Data size	200 MB	
	Pattern length	Short	Long
Attempts	Best	Pitch	Pitch
	Worst	DNA	DNA
Character comparisons	Best	No specific data	No specific data
	Worst	No specific data	Pitch

The Pitch database shows the fewest number of attempts because this algorithm depends on the AbdulRazzaq technique, which has efficient functions (bmBc and hash) for managing the Pitch database because data are encoded as MIDI pitch numbers in computer applications [13], [14]. The hash function also uses integer numbers and the good shifting of the bmBc function to reduce the number of attempts. No specific data are found in the E-AR algorithm in terms of the number of character comparisons, similar to the results in the AbdulRazzaq algorithm. This finding can be attributed to the dependency of the E-AR algorithm on the hash of the prime number, that is, characters are selected depending on prime number location and not on character data types. The DNA database shows the worst results in terms of the number of attempts performed because of its small alphabet size (Table 2.).

7.2. Evaluation of the E-AR algorithm compared with recent and standard algorithms

The performance results of the E-AR algorithm are compared with those of the recent and standard algorithms in terms of the number of attempts and character comparisons when using short and long patterns with different data types and sizes. The standard and recent algorithms employed in this study are Horspool, Quick-search, Two-way, Fast search, SSABS, TVSBS, AKRAM, and Maximum shift.

Table 3: Comparison of the Results of E-AR and Recent and Standard Algorithms

Algorithms	Data size (MB)	
	200	
	Short	Long
Number of attempts		
Best algorithm	E-AR (most databases)	Maximum shift (all databases)
Worst algorithm	Two-way (all databases)	Two-way (all databases)
Number of character comparisons		
Best algorithm	E-AR (most databases)	E-AR (most databases)
Worst algorithm	Two-way (all databases)	Two-way (all databases)

The E-AR algorithm obtains the fewest number of attempts in short patterns because this algorithm relies on the efficient functions (bmBc) of the AbdulRazzaq algorithm and (brBc) of the BR algorithm. In long patterns, the Maximum shift algorithm obtains the fewest number of attempts because it depends on efficient functions (ztBc) and (qsBc), which are useful in long patterns [15]. The E-AR algorithm obtains the lowest number of character comparisons because it depends on the good performance of the AbdulRazzaq algorithm. Both these algorithms calculate the hash values of prime characters in the first step, thereby reducing the probability of a mismatch between the pattern and text window owing to the accurate hash value of all prime characters. The Two-way algorithm obtains the highest number of attempts and character comparisons because this algorithm depends on its factorization technique in the preprocessing step [16] (Table 3.).

Table 4: Ranking of the E-AR Algorithm in Different Database Types

Databases	Ranking of E-AR algorithm	
	Pattern length	Data size
		200 MB
Attempts	**Short**	First in all databases (but second in DNA)
	Long	Second in all
Character comparisons	**Short**	First in all databases (but second in Pitch & English)
	Long	First in all databases (but second in English)

For the number of attempts, the E-AR algorithm ranks first in most database types and in all sizes when short patterns are used. This algorithm also ranks first in the Source database when long patterns are used. For the number of character comparisons, the E-AR algorithm ranks first in most databases when short and long patterns are used (Table 4.).

8. Conclusion

The proposed algorithm in this study obtain the best results in terms of the number of attempts compared with the original algorithms when short and long pattern lengths are used. The algorithm ranks first in short patterns compared with the recent and standard algorithms and ranks second in some of data types when long patterns are used. For the number of character comparisons, the proposed algorithm show the best results compared with the original algorithms in short and long pattern lengths. The E-AR algorithm also perform better than the recent and standard algorithms in short pattern lengths, while it was the first in long patterns. The Pitch database shows the best performance in the number of attempts with the proposed algorithms, whereas the DNA database performs the worst. No data is specified as the best or worst with the E-AR algorithm in terms of the number of character comparisons. In terms of number of attempts, the E-AR algorithms ranked first in most databases when using short patterns, and ranked first in some databases when using long patterns. In terms of number of character comparisons the E-AR algorithm ranked first when using short and long patterns in most databases.

References

[1] AbdulRazzaq AA, Abdul Rashid N A, Abu-Hashem MA & Zainol Z (20017) "New Searching Technique of Hybrid Exact String Matching lgorithm,".*International Review on Computers and Software (I.RE.CO.S.)*, vol.11 (10), pp. 884-897.

[2] AbdulRazzaq AA, Rashid NA, Abu-Hashem MA, & Hasan AA (2014) "A New Efficient Hybrid Exact String Matching Algorithm and Its Applications," *Life Science Journal (Life Sci J)*, vol.11 (10), pp.474-488.

[3] Al-Dabbagh SSM, & Barnouti. NH (2017) "A New Efficient Hybrid String Matching Algorithm to Solve the Exact String Matching Problem" *British Journal of Mathematics & Computer Science*, vol.20 (2), pp. 1-14. https://doi.org/10.9734/BJMCS/2017/30497.

[4] Berry T, & Ravindran SA (1999) "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.

[5] Huang Y, .Ping L, Pan X, Jiang L, & Jiang X (2008) "A Fast Improved Pattern Matching Algorithm for Biological Sequences" *International Symposium on Computational Intelligence and Design, ISCID '08*, IEEE, pp. 8-12.

[6] Cai G, Nie X, & Huang Y (2009) "A Fast Hybrid Pattern Matching Algorithm for Biological Sequences" *Proceedings of 2nd International Conference on Biomedical Engineering and Informatics, BMEI '09*, pp. 1-5. https://doi.org/10.1109/BMEI.2009.5305645.

[7] Karkkainen J, & Joong CN (2009) "Faster Filters for Approximate String Matching" *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX*, pp. 84-90.

[8] Klaib AF, & Osborne H (2009) "BRQS Matching Algorithm for Searching Protein Sequence Databases" *Proceedings of the 2009 International Conference on Future Computer and Communication, ICFCC '09*, IEEE, pp. 223-226. https://doi.org/10.1109/ICFCC.2009.40.

[9] Kurt V (2005) "Protein Structure Prediction using Decision Lists," M.Sc. Thesis, Dept, Computational Sciences and Engineering, Koç University, İstanbul, Turkey.

[10] Chew E, & Chen YC (2003) "Mapping Midi to the Spiral Array: Disambiguating Pitch Spellings, In H. K. Bhargava and Nong Ye, eds. Computational Modeling and Problem Solving in the Networked World" *Proceedings of the 8th INFORMS Computer Society Conference*, ICS2003, pp. 1-17.

[11] Ferragina P, & Fischer J (2007) "Suffix Arrays on Words. In: Bin Ma, Kaizhong Zhang (eds.)," Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching, CPM'07, pp.328-339, 2007. https://doi.org/10.1007/978-3-540-73437-6_33.

[12] Deighton RA (2012) "Using Rabin-Karp fingerprints and LevelDB for faster searches," M.Sc. Thesis, Dept, Department of Computer Science, University of Ontario Institute of Technology (UOIT), Oshawa, Canada.

[13] Cambouropoulos E (2001) "Automatic pitch spelling: From numbers to sharps and flats" *In VIII Brazilian Symposium on Computer Music SBC&M*.

[14] Honingh A (2007) "Pitch spelling: Investigating reductions of the search space," *Proceedings SMC'07, 4th Sound and Music Computing Conference.*

[15] Kadhim HA (2012) "New Sequential and Gpu-Based Hybrid String Matching Algorithms." M.Sc. Thesis, Dept, Computer Science School, University Science Malaysia (USM), Penang, Malaysia.

[16] Charras C & Lecroq T (2004) "Handbook of Exact String Matching Algorithms" *King's College Publications.*