British Journal of Mathematics & Computer Science
20(2): XX-XX, 2017; Article no.BJMCS.30497
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

A New Efficient Hybrid String Matching Algorithm to Solve the Exact String Matching Problem

Sinan Sameer Mahmood Al-Dabbagh[1*] and Nawaf Hazim Barnouti[2]

[1]Department of Software Engineering and Information Technology, Al-Mansour University College, Baghdad, Iraq.
[2]Al-Mansour University College, Baghdad, Iraq.

Authors' contributions

This work was carried out in collaboration between both authors. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2017/30497
Editor(s):
(1)
(2)
Reviewers:
(1)
(2)
(3)
Complete Peer review History:

Original Research Article

Received: 12th November 2016
Accepted: 19th December 2016
Published: 29th December 2016

Abstract

The string matching algorithms are considered one of the most studied in the computer science field because the fundamental role they play in many different applications such as information retrieval, editors, security applications, firewall, and biological applications. This study aims to introduce a new hybrid algorithm based on two well-known algorithms, namely, the modified Horspool and SSABS hybrid algorithms. Two factors used to analyze the proposed algorithm which is the total number of character comparisons and total number of attempts. The ABSBMH algorithm which is the name chosen for the proposed hybrid algorithm was tested on different types of standard datatype. The ABSBMH algorithm shows less number of character comparisons when compared to the results of other algorithms, while show almost no big different in the results of number of attempts this is due to the proposed hybrid algorithm preprocessing phase based on SSABS algorithm which is the same preprocessing phase of the Quick Search algorithm, so for all these reasons the results of the ABSBMH and other algorithms in terms of total number of attempts have been shown a small different, this is because it use different pattern lengths which are selected randomly from the databases. The experiential results expose that

**Corresponding author: E-mail: sinan.aldabbagh815@gmail.com;*

performance of the hybrid algorithm influenced by the type of the dataset used, the DNA sequence shows the worst result, while the English text datatype show the best results in terms of total number of character comparisons.

Keywords: String matching algorithm; ABSBMH algorithm; pattern matching; exact string matching; SSABS algorithm; horspool algorithm; quick search algorithm; hybrid string matching.

1 Introduction

The exact string matching problem is defined as an effort to locate all occurrences of a short string P generally called pattern of length m in a long string T generally called text of length n [1,2]. It is a widely researched problem in computer science, primarily as a result of its extensive use in different software applications such as, signal processing, speech analysis, text searching, image, pattern recognition and computational biology [3-5]. Today, modern biology considers retrieving information and decoding the meaning of biological sequences as the central problem that needed to be solved [5]. In general, basic biological information is kept in strings of nucleic acids (DNA, RNA) or amino acids (proteins). Aligning sequences assists in exposing their common characteristics, whereas matching sequences could conclude valuable information from them. Because of the large amounts of DNA data available on internet websites, matching of nucleotide sequences is becoming an essential application and also there is a growing interest in extremely fast computer technique for data analysis and retrieval [6].

String matching algorithm open a text window which has the same length of the pattern length m, matching operation performed between the text window characters and pattern characters, if a match or miss match occur the text window shift to the right side with a specific distance determined by the type of string matching algorithm used, (this specific effort called attempt) [7]. The goal of an efficient algorithm is to reduce the amount of work performed in each attempt and to extend the length of the shifts. After the new shift has positioned, the algorithm quickly tries to find if the pattern characters occur in the text window characters or not, if not the text window continue shifting to the right side to exceed the right end of the text after that the process will finish [8].

In this paper the SSABS and modification of Horspool algorithms are adopted to be hybridized to produce a new efficient hybrid algorithm. The modified Horspool algorithm shows good behavior in terms of the number of character comparisons [9], while SSABS algorithm shows a good behavior in terms of the total number of attempts due to the Quick Search bad character table (qsBc) [10].

Depending on the opposite behavior of the SSABS and Quick Search algorithms which work with different alphabet types and different pattern lengths, in addition to the wasted time that spent when searching large data size, this paper try to find a solution to the overall performance drawbacks of the two present algorithms, by proposing a method to blending the good features and exclude the bad features of both algorithms, to solve the problem of the string matching algorithm efficiently with less number of characters comparisons.

2 Related String Matching Algorithms

Generally string matching algorithms are divided into two independent groups: Exact string matching algorithms and approximate string matching algorithms, the exact string matching algorithms are divided into two categories on-line and off-line string matching algorithm [11]. In the on-line string matching algorithms, the pattern preprocessing and the text leave intact, while in the off-line string matching algorithms the text is preprocessing [12]. In this paper, we focus on the on-line exact string matching algorithm with more specificity the hybrid exact string matching algorithms.

The Boyer moore algorithm was developed by Robert S. Boyer and J Strother Moore [13,14]. This algorithm considered the first sub linear string matching algorithm. The Boyer moore algorithm consists of two phases preprocessing phase and searching phase. In the preprocessing phase the algorithm constructs two tables, which are bad character table and good-suffix table. During the searching phase the algorithm search the text window from right to left, if the pattern doesn't occur in the text window the pattern shifts to the right side with a specific shift value. The shifting values determined with the help of the bad character table and good-suffix table.

The Horspool algorithm considers a simplified version of the Boyer moore algorithm. It was modified simply by sacrificing the good suffix shift table. The good suffix shift might not be of practical value in larger alphabets were no extensive internal repeats may be anticipated [15].

After a complete pattern match or mismatch occurs, this algorithm shifts the text window to the right side. During the searching phase the algorithm determines the shift value by examining the rightmost character of the text window with its corresponding shifting values in the bad character table [16].

The Quick Search algorithm (QS) its simplification of the Boyer moore algorithm by taking only the bad character table and dropping the good suffix shift table. During the searching phase the character that next to the rightmost character in the text window its value in the bad character table determine the next shift distance [17]. On the other hand, the Raita algorithm proposed by RAITA in 1992, the algorithm uses only the bad character table of the Boyer moore algorithm. During the searching phase the matching process, start by checking the rightmost character, then leftmost character and the middle one [18]. Due to the fact neither of them the pattern and the text are random in practice, the Raita algorithm suggests a new implementation that makes utilize of the dependencies between consecutive characters [19].

The Smith hybrid string matching algorithm proposed by Dr. Smith in 1991 [20]. This algorithm is a result of blending the good advantage of the Horspool and Quick Search algorithms. The preprocessing phase consists of building two tables, which are the Quick Search bad character table (qsBc) and Horspool bad character table. During the searching phase the Smith algorithm compares the two shifting values from the two tables and choose the maximum shift value.

The SSABS hybrid string matching algorithm proposed by Sheik [21] it's a combination of the good characteristic of Quick Search and Raita string matching algorithms. In this algorithm the matching process between the text window characters and the pattern characters are performed in a fixed order during each attempt, while the shifting values depend on the Quick Search bad character table (qsBc). The algorithm starts the matching process with the rightmost character of the pattern with its corresponding character in the text window if its match, then leftmost character of the pattern compares with its corresponding character in the text window if it matches, the remaining characters compares from right to left.

Maximum-Shift algorithm [22] is the result of combining the Quick-Search, Zuh-Takaoka and Horspool algorithms. The preprocessing phase of this algorithm consists of computing the Quick Search bad character table (qsBc) and Zuh-Takaoka bad character table. In the searching phase the author adopts the idea of modified Horspool algorithm proposed by Liu [9] which checks instead of checking only the last character in the text window the algorithm checks the last two successive characters in the text window with its corresponding characters in the pattern.

3 The Proposed Algorithm

This section concentrates on solving the exact string matching problem by blending two well-known algorithms SSABS and modification of Horspool algorithms, to propose an efficient hybrid string matching algorithm can deal with different dataset size and different pattern length, with less number of character comparisons along with the total number of attempts. The preprocessing and searching phases of the proposed hybrid string matching algorithm, are summarized in the next subsections (see Fig. 1).

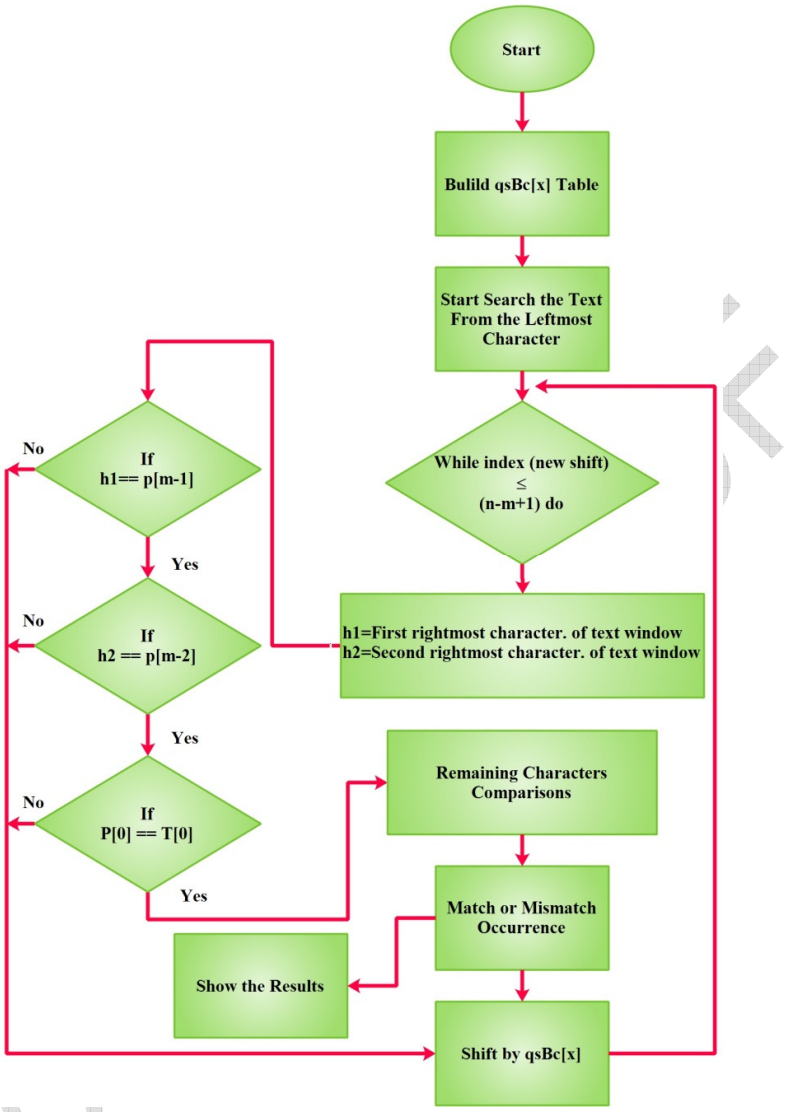


Fig. 1. Flowchart for ABSBMH hybrid algorithm overview

3.1 Preprocessing phase

In general, the preprocessing phase is utilized to preprocess the pattern characters as a way to gather information which are useful in the searching phase. The information gathered in the preprocessing phase is used to minimize the total number of character comparisons along with the total number of attempts. In the proposed hybrid string matching algorithm, the preprocessing phase depends on the Quick Search bad character table *(qsBc)* to determine the next position of the text window. The Quick Search algorithm gives the maximum shift value *(m+1)* when the character next to the rightmost character on the text window does not appear in the pattern. On the other hand, when the character next to the rightmost character on the text window equals to the rightmost character, the Quick Search algorithm gives the minimum shift value. See the Equation 1.

$$qsBc(x) = \begin{cases} (i: 0 \le i < m \text{ and } P[m-i] = x) \text{ if } x \text{ occurs in } P, \text{ otherwise} \\ m+1 \end{cases} \quad (1)$$

3.2 Searching phase

The methods in this phase rely on the searching phase of the original algorithms using fixed orders with modification during the matching operation adopted from the modified Horspool algorithm [8]. The proposed algorithm first checks the last two successive characters in the text window with the pattern characters as an initial step, instead of checking only the last character in the text window. Moreover, as a second step, the algorithm checks the first most character in the text window with its corresponding character in the pattern, if its match the remaining characters compares from left to right, and if the mismatch occurs, the text window shifts to the right side depending on the shift values stored in the Quick Search bad character table *(qsBc)*. With this modification on the matching operation the number of characters' comparison will reduced in each attempt as well as the total number of characters' comparisons (See Fig. 2).

```
Algorithm ABSBMH (P [0…m-1], T[0…n-1])
1.  //Input: Pattern P, Text T
2.  //Output: All the indexes of P occurs in the T
3.  qsBc [Asize]
4.  preQsBc (P, m, qsBc)
5.  while j ≤ n-m+1 do
6.  h1 = T [j+m-1], h2 = T [j+m-2]
7.  for i→ 0 to m do
8.  If (h1= P [j+m-1] && h2= P [j+m-2] &&P [i] = T [i+j]) Then
9.  Print the first index of P occurrence in T
10. End if
11. End for loop
12. J+ = ( qsBc [T[j+m]] )
End while loop
```

Fig. 2. Pseudocode of the searching phase of the proposed hybrid algorithm

4 Experimental Design

This section describes the design of the proposed hybrid algorithm which adopt the good features and excluded the weak points of the original algorithms. The ABSBMH algorithm which can work efficiently with different alphabet size and pattern lengths. The ABSBMH algorithm was examined with the original algorithms, the final step in this study was evaluating the results of ABSBMH with the original algorithms.

4.1 Experimental databases

The ABSBMH algorithm was examined using three different data types which are DNA sequence, Proteins sequence and English text these data types were downloaded from the website (http://pizzachili.dcc.uchile.cl/) with the intention to assess the performance of the ABSBMH algorithm against its original algorithms. DNA sequence which considers a small alphabet with 4 characters only, which are A, C, G, T they mean Adenine, Cytosine, Guanine and Thymine respectively. Proteins sequence which considers a medium alphabet with 20 amino acids. English text which considers a large alphabet with more than 100 characters distributed as small and capital letters of the English language alphabet, numbers and special characters [23].

5 Results

The experiment was performed using a laptop with a processor Core i7 2.4 GHz and 8 GB Ram. Microsoft visual studio 2010 utilized as the programing environment with a visual C++ compiler to write and compiles the programs. The ABSBMH and its original algorithms was run 5 times with each pattern length, this process was repeated with the three databases mentioned in the subsection 4.1. The pattern lengths adopted in this study was selected randomly from the databases, in order to analyze the behavior of the hybrid algorithm and its original algorithms with different alphabet size and pattern lengths, these pattern lengths are: 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100.

5.1 Result analysis

This study aims to develop an efficient hybrid exact string matching algorithm which can be used in different alphabet size and pattern lengths. Total number of character comparisons and total number of attempts was adopted in this study as a standard performance measures to evaluate the ABSBMH and its original algorithms.

5.1.1 Evaluation the number of character comparisons

This section describes the results of comparing the hybrid algorithm with its original algorithms using DNA sequence, Proteins sequence and English text in terms of total number of character comparisons. Total number of character comparisons based on calculating the actual comparisons between the text window characters and the pattern characters, this factor are considered a standard benchmark to measure the performance of a string matching algorithm [12].

5.1.2 Analyzing the total number of character comparisons

Table 1 shows the average number of character comparison results when comparing the hybrid algorithm with its original algorithms using DNA sequence, the DNA sequence is considered a small alphabet with 4 characters only, which it can be helpful to analyze the behavior of these algorithms in which small alphabet size.

Table 1. Total number of character comparisons of DNA sequence

Pattern length	Horspool	QS	SSABS	ABSBMH
10	79,814,354	76,931,784	70,136,549	50,912,342
20	72,087,133	71,029,473	62,965,607	39,818,881
30	67,265,948	73,601,807	64,196,177	49,708,609
40	81,545,058	84,584,456	63,637,559	37,660,468
50	91,124,982	89,890,959	79,105,778	41,023,048
60	83,381,134	75,950,549	61,572,324	33,316,472
70	75,888,183	60,301,088	48,511,688	28,754,645
80	68,465,609	63,665,333	54,433,505	23,395,282
90	64,754,100	59,094,108	40,708,983	20,916,974
100	51,094,017	49,333,878	27,136,080	14,592,631

Table 2 shows the average number of character comparison results when comparing the hybrid algorithm with its original algorithms using Protein sequence, the Protein sequence is considered a medium alphabet with 20 amino acids, which it can be helpful to analyze the behavior of these algorithms in which a medium alphabet size.

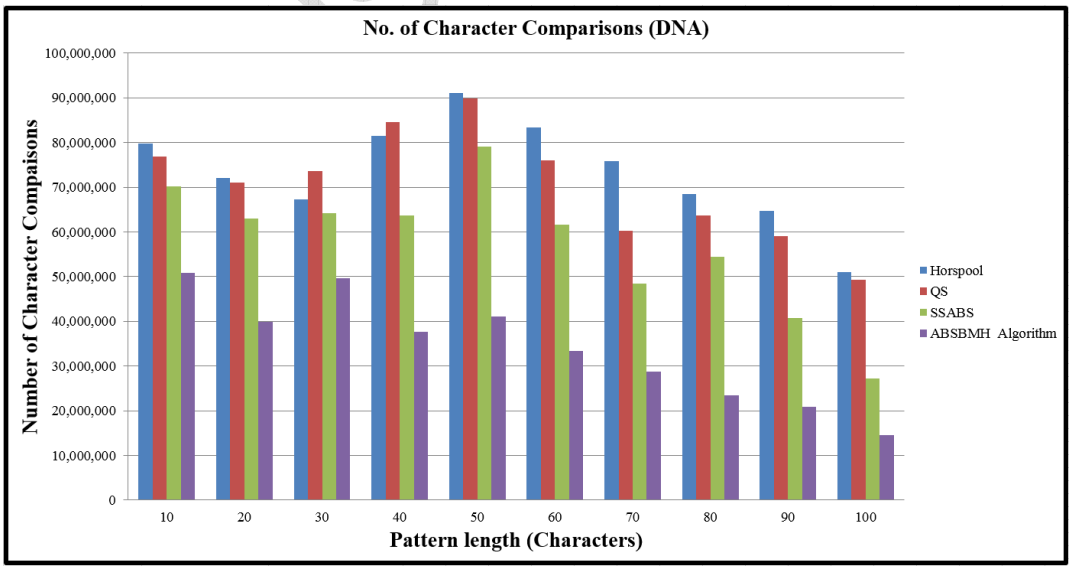
Table 2. Total number of character comparisons of protein sequence

Pattern length	Horspool	QS	SSABS	ABSBMH
10	29,100,477	26,657,033	24,559,788	13,125,669
20	18,111,296	17,157,965	15,067,732	9,357,597
30	14,842,321	14,825,460	13,008,056	9,106,514
40	13,497,913	12,566,666	11,663,913	8,702,883
50	14,864,635	13,054,518	12,720,537	7,538,567
60	11,843,043	12,525,298	11,032,603	6,644,172
70	10,723,295	11,204,529	10,855,943	5,877,978
80	13,689,713	11,033,560	9,826,810	4,844,674
90	14,587,608	11,898,467	8,991,582	3,912,522
100	15,376,301	11,728,947	7,626,340	2,537,843

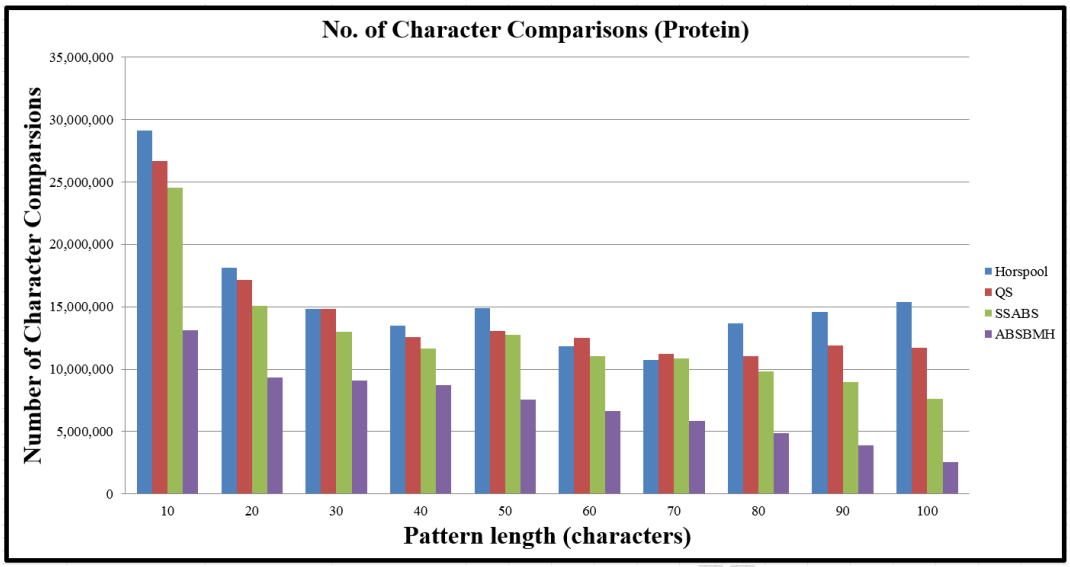
Table 3 shows the average number of character comparison results when comparing the hybrid algorithm with its original algorithms using English text data type, the English text is considered a large alphabet with more than 100 characters distributed as small and capital letters of the English language alphabet, numbers and special characters.

Table 3. Total number of character comparisons of English text data type

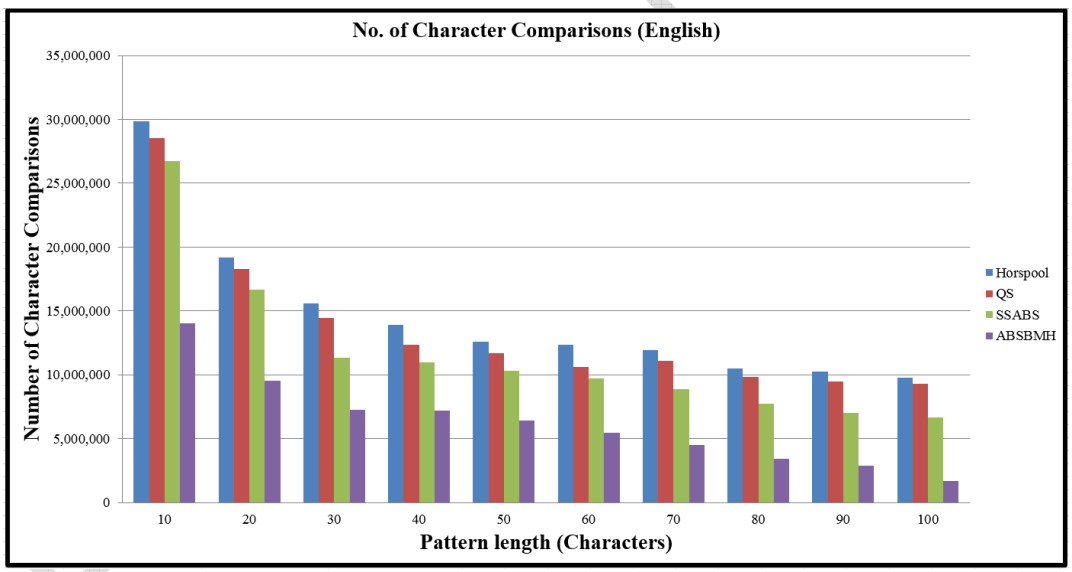
Pattern length	Horspool	QS	SSABS	ABSBMH
10	29,839,677	28,553,942	26,743,948	14,028,546
20	19,158,613	18,277,850	16,678,825	9,543,646
30	15,582,386	14,434,795	11,354,895	7,250,725
40	13,882,166	12,327,418	10,998,909	7,188,751
50	12,602,486	11,690,756	10,304,201	6,445,256
60	12,371,397	10,636,101	9,728,644	5,439,235
70	11,920,340	11,118,182	8,850,681	4,494,831
80	10,490,744	9,851,652	7,751,473	3,446,478
90	10,236,944	9,494,942	7,014,147	2,899,972
100	9,788,874	9,304,106	6,661,737	1,710,680



(a)



(b)



(c)

Fig. 3. Present number of character comparisons when using DNA sequence, proteins sequences and English text datatype

Fig. 3 above demonstrates the results of comparing the hybrid algorithm with its original algorithms when using DNA sequence, Protein sequence and English text datatype in terms of total number of character comparisons. (a) Describes the results of comparing the hybrid algorithm with its original algorithms when using DNA sequence. The Quick Search and Horspool algorithms give unstable behavior, it can be seen clearly from pattern length 10 to 40, this unstable behavior is due to the nature of the DNA sequence alphabet which is small in size. The Quick Search and Horspool algorithms have a horrible behavior when it comes to small pattern, but averagely the Quick Search algorithm gives more stable behavior, comparing

with the results of the Horspool algorithm, the Horspool algorithm overcome the Quick Search algorithm in some pattern length sizes. This is due to the characteristics of the preprocessing phase of the Quick Search algorithm, which depend on the value of the character next to the rightmost character in the text window to determine the next position of the text window in case of complete match or mismatch occurs, gaining a significantly less number of character comparisons. While the Horspool algorithm calculates the next position of the text window in case of complete match or mismatch occur depending on the last character in the text window.

On the other hand, the SSABS algorithm also shows the unstable behavior when using DNA sequence data type, but the results shown better performance when compared to Horspool and Quick Search algorithms this due to the nature of the algorithm which is consider a hybrid algorithm combined from Quick Search and Rita algorithms.

The results of ABSBMH algorithm which is the name of our proposed hybrid algorithm show the less number of character comparison when comparing to the original algorithms, this is due to the nature of the hybrid algorithm which composite of SSABS and Quick Search algorithms. The preprocessing phase depends on the Quick Search algorithm bad character table to shift the text window to the next position, while using the searching phase of the modified Horspool algorithm during the matching process between the text window and pattern characters, all these good characteristics made the hybrid algorithm performance super the other algorithms when using DNA sequence data type.

(b) Describes the results of comparing the hybrid algorithm with its original algorithms when using Protein sequence data type. The Quick Search and Horspool algorithms give more stable behavior than when using DNA sequence, it can be seen clearly from pattern 20 to 100, this unstable behavior is due to the nature of the Protein sequence alphabet which is medium in size. The two algorithms give a bad behavior in the pattern length 10 which is considered a small pattern. The SSABS hybrid algorithm also shows the more stable behavior than when using DNA sequence, this is due to the nature of its original algorithms the Quick Search and Raita algorithms. However, the ABSBMH algorithm shows better performance compared with all these algorithms, this is due to the good characteristics combined from the original algorithms SSABS and Horspool. The searching phase depends on the preprocessing phase of the SSABS algorithm, while the searching phase a adopted the modified Horspool algorithm to check the first rightmost character and the second rightmost character as a first step before matching the remaining characters, this matching process had decreased the total number of character comparisons.

Consequently, the good quality behavior of total number character comparisons of the ABSBMH algorithm is the response to these two reasons. (c) Describes the results of comparing the hybrid algorithm with its original algorithms when using English text datatype. The Quick Search and Horspool algorithms gives stable behavior when implemented using English text data type. The bad behavior appears in the pattern length 10, from 20 to 100 pattern length the two algorithms show good performance and the Quick Search algorithm beat the Horspool algorithm for all pattern lengths. The SSABS algorithm shows better performance than the Quick Search and Horspool algorithms this advantage comes from the Quick Search and Raita algorithms that composite the SSABS hybrid algorithm, also the nature of the English text data type which is considered a large alphabet. On the other hand, the ABSBMH algorithm gives less character comparisons than all other algorithms this because of depend on the SSABS preprocessing phase to shift the pattern when a complete match or mismatch occur, while during the searching phase it utilizes the modified Horspool algorithm to check the last two consecutive characters not just only the rightmost character, this process will significantly decrease the total number of character comparisons. Therefore, it can observe the performance of the hybrid algorithm overcome all others algorithms.

5.2 Evaluation the total number of attempts

This section describes the results of comparing the hybrid algorithm with its original algorithms using DNA sequence, Proteins sequence and English text in terms of total number of attempts. Total number of attempts based on calculating the distances that the pattern need to jump over the whole given text. Clearly, whenever

the total number of attempts for an algorithm is less than other algorithms, this algorithm has better performance [24].

5.2.1 Analyzing the total number of attempts

Table 4 shows the average number of number of attempts of character comparison results when comparing the hybrid algorithm with its original algorithms using DNA sequence, as mentioned in the previous subsection the alphabet of DNA sequence data type is consists only from 4 characters which is considered a small alphabet, that could be used to test the performance of the hybrid algorithm over the performance of its original algorithms.

Table 4. Total number of attempts using DNA sequence data type

Pattern length	Horspool	QS	SSABS	ABSBMH
10	66,703,668	57,477,883	52,080,613	50,097,018
20	57,663,549	52,468,482	46,500,581	46,870,478
30	61,323,371	53,934,288	47,233,564	47,201,390
40	62,376,335	61,563,433	57,232,010	48,582,339
50	62,233,196	65,995,037	66,346,480	60,810,768
60	63,922,306	61,862,007	63,684,418	55,685,549
70	49,949,028	47,632,186	44,290,229	47,352,621
80	45,575,263	47,600,632	49,327,410	48,910,786
90	47,325,239	45,840,130	50,325,733	51,387,417
100	40,320,846	38,065,292	35,317,942	32,479,216

Table 5 shows the average number of number of attempts results when comparing the hybrid algorithm with its original algorithms using Protein sequence, as mentioned in the previous subsection the Protein sequence data type is considered a medium alphabet with 20 amino acids, the performance of the hybrid algorithm with its original algorithms is tested using such a medium data type as shown in the results below.

Table 5. Total number of attempts using protein sequence data type

Pattern length	Horspool	QS	SSABS	ABSBMH
10	27,063,576	24,360,106	25,419,533	24,919,393
20	17,420,635	16,133,875	16,715,668	15,803,364
30	14,532,122	13,816,309	13,107,727	12,738,423
40	10,550,915	9,446,881	9,251,502	10,140,390
50	13,979,330	11,954,299	11,357,820	9,921,654
60	9,725,286	9,126,159	10,023,996	9,776,403
70	9,709,458	10,259,679	10,065,174	10,289,591
80	11,773,959	10,205,066	10,832,723	9,957,163
90	12,845,778	11,222,140	10,882,415	9,391,718
100	12,544,204	11,210,448	9,506,798	8,860,536

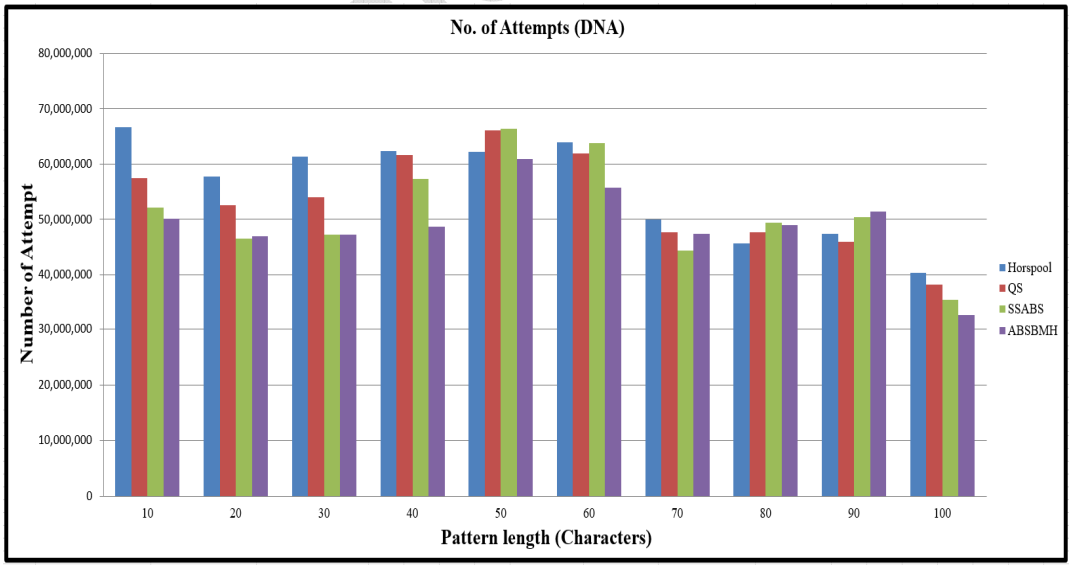
Table 6 shows the average number of number of attempts results when comparing the performance of the hybrid algorithm with its original algorithms using English text data type, the English text is considered a large alphabet with more than 100 characters, with this large alphabet is helpful to analyses the performance of the hybrid algorithm when compared it to the performance of its original algorithms.

The empirical results of counting the total number of attempts for DNA sequence, Protein sequence and English text data type are shown in Fig. 4. Counting the total number of attempts for the hybrid algorithm as well as for its original algorithms depends on the efficiency of the preprocessing phase of each algorithm. (a) Describes the results show whenever the DNA sequence datatype is used the Horspool and Quick Search algorithm gives unstable behavior, it can be observed especially when the small pattern length is used. The

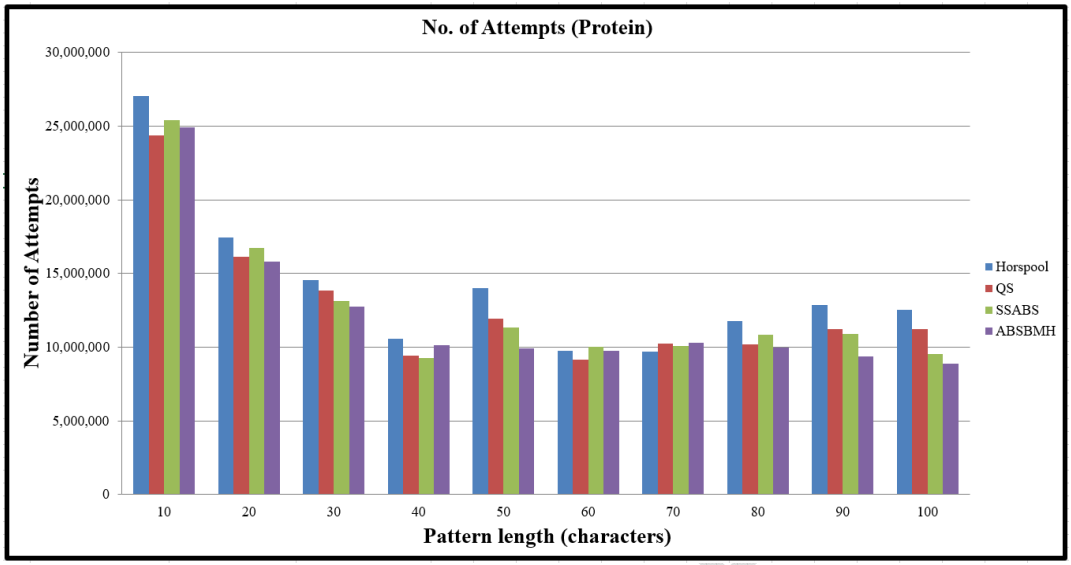
hybrid and SSABS algorithms show also unstable behavior when the DNA sequence datatype is used, this is due to the nature of the dataset which is considered a small alphabet. Therefore, (b) describe whenever the Protein sequence datatype is using the hybrid algorithm and its original algorithms show more stable behavior when compared to the DNA dataset results. On the other hand, (c) describe when the hybrid algorithm and its original algorithms implemented using English text datatype the algorithms shown stable behavior when compared to the DNA and Protein sequence datatype, this due to the nature of the alphabet used which is considered a large alphabet. However, the results show whatever the datatype used the total number of attempts of the hybrid algorithm is almost the same results of its original algorithms the reason behind that the hybrid algorithm is a combination of SSABS and the modified Horspool algorithm, where the shifting distance of the pattern whenever a complete match or mismatch occur is based on the preprocessing phase of the SSABS algorithm. The SSABS algorithm is also a hybrid algorithm which depends on the preprocessing phase of the Quick Search algorithm, so for that two reasons the ABSBMH and SSABS algorithms with the Quick Search algorithm have almost the same results because it all depends on the preprocessing phase of the Quick Search algorithm, whatever the dataset used. The small differences from the results between the Quick Search, ABSBMH and SSABS algorithms is due to the pattern lengths which are selected randomly from the databases.

Table 6. Total number of attempts using English text data type

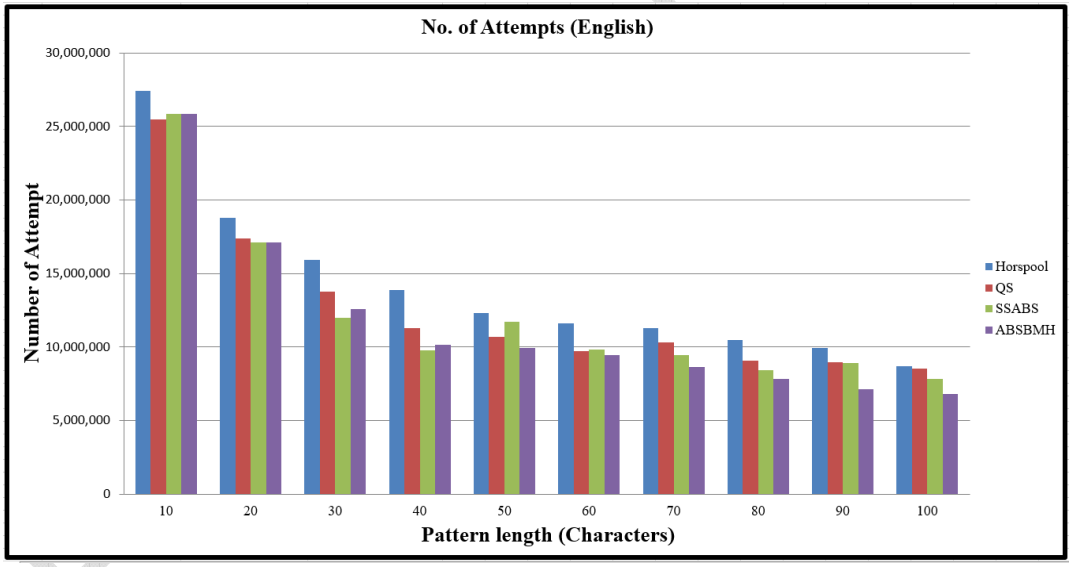
Pattern length	Horspool	QS	SSABS	ABSBMH
10	27,403,832	25,476,247	25,872,440	25,872,440
20	18,791,663	17,401,603	17,101,036	17,101,036
30	15,911,547	13,746,559	12,000,275	12,600,275
40	13,854,832	11,259,784	9,760,219	10,160,219
50	12,322,629	10,709,255	11,713,919	9,913,919
60	11,608,808	9,713,455	9,827,563	9,427,563
70	11,278,424	10,310,124	9,439,215	8,639,215
80	10,458,012	9,050,026	8,409,843	7,809,843
90	9,932,749	8,946,716	8,907,759	7,107,759
100	8,689,255	8,550,596	7,804,464	6,804,464



(a)



(b)



(c)

Fig. 4. Present number of attempt when using DNA sequence, proteins sequences and English text datatype

6 Conclusions

This paper aims to take the good characteristics of the modified Horspool and SSABS hybrid algorithms and put it together in a new efficient hybrid exact string matching algorithm which can take the advantage of both original algorithms and exclude the weak points of each algorithm. The proposed hybrid algorithm which is called ABSBMH have shown higher performance when compared it with the original algorithms

and a Quick Search algorithm by providing a less number of character comparisons. While in terms of total number of attempts the ABSBMH and the other algorithms show almost the same results this is because the hybrid algorithm based on a SSABS preprocessing phase, which in turns depend on the Quick Search preprocessing phase, the results of average number of number of attempts of the ABSBMH and other algorithms have small different, this is because different pattern lengths used which are selected randomly from the databases. The performance of the ABSBMH and the other algorithms is affected by the type of the dataset used, the English text datatype show higher performance while the DNA sequence shows the lowest performance in terms of total number of character comparisons.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Faro Simone, Oğuzhan Külekci. Efficient algorithms for the order preserving pattern matching problem. arXiv preprint arXiv:1501.04001; 2015.

[2] Al-Dabbagh, Sinan Sameer Mahmood, et al. Parallel quick search algorithm for the exact string matching problem using openMP. Journal of Computer and Communications. 2016;4.13:1.

[3] Faro Simone, Thierry Lecroq. The exact string matching problem: A comprehensive experimental evaluation. arXiv preprint arXiv:1012.2547; 2010.

[4] Russo Luís, et al. Approximate string matching with compressed indexes. Algorithms. 2009;2(3): 1105-1136.

[5] Nicolae Marius, Sanguthevar Rajasekaran. On string matching with mismatches. Algorithms. 2015; 8(2):248-270.

[6] Cantone, Domenico, Salvatore Cristofaro, Simone Faro. Efficient string-matching allowing for non-overlapping inversions. Theoretical Computer Science. 2013;483:85-95.

[7] Charras C, Lecroq T, Pehoushek JD. A very fast string matching algorithm for small alphabets and long patterns. In M. Farach-Colton, editor, Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching, number 1448 in Lecture Notes in Computer Science, Piscataway, NJ, Springer-Verlag, Berlin. 1998;55–64.

[8] Lecroq Thierry. Fast exact string matching algorithms. Information Processing Letters. 2007;102(6):229-235.

[9] Liu Y. Comparison studies of exact pattern matching algorithms, Technical Report, Computational Biology Algorithm, Department of Computer Science University of Victoria; 2008.

[10] Faro Simone, Thierry Lecroq. The exact online string matching problem: A review of the most recent results. ACM Computing Surveys (CSUR). 2013;45(2):13.

[11] Navarro Gonzalo. A guided tour to approximate string matching. ACM Computing Surveys (CSUR). 2001;33(1):31-88.

[12] Michailidis Panagiotis D, Konstantinos G. Margaritis. On-line string matching algorithms: Survey and experimental results. International Journal of Computer Mathematics. 2001;76(4):411-434.

[13] Boyer RS, Moore JS. A fast string-searching algorithm. Communication of ACM. 1977;20(10):762-772.

[14] Richa Sharma, Vibhor Gupta, Vikas Kumar. Efficient parameterized string matching algorithm. International Journal of Emerging Research in Management and Technology (IJERMT); 2015.

[15] Allmer Jens. Exact pattern matching: Adapting the boyer-moore algorithm for DNA searches. No. e1758v1. Peer J Pre Prints; 2016.

[16] Ibrahim Amin Mubark Alamin, Mustafa Elgili Mustafa. Comparison criteria between matching algorithms texts application on (horspool's and brute force algorithms). Journal of Advanced Computer Science & Technology. 2015;4(1):175-179.

[17] Charras Christian, Thierry Lecroq. Handbook of exact string matching algorithms. King's College; 2004.

[18] Raita Timo. Tuning the Boyer-Moore-Horspool string searching algorithm. Softw., Pract. Exper. 1992;22(10):879-884.

[19] Liao Yi-Ching. A survey of software-based string matching algorithms for forensic analysis. Proceedings of the Conference on Digital Forensics, Security and Law. Association of Digital Forensics, Security and Law; 2015.

[20] Smith PD. Experiments with a very fast substring search algorithm. Softw., Pract. Exper. 1991;21(10):1065-1074.

[21] Sheik SS, Aggarwal SK, Poddar A, Balakrishnan N, Sekar K. A fast pattern matching algorithm. Journal of Chemical Information and Computer Sciences. 2004;44:1251-1256.

[22] Kadhim, Hakem Adil, NurAini Abdul Rashidx. Maximum-shift string matching algorithms. Computer and Information Sciences (ICCOINS), 2014 International Conference on. IEEE; 2014.

[23] Abdul Razzaq, Atheer Akram, Muhannad A. Nur'Aini Abdul Rashid, Awsan Abdulrahman Hasan Abu-Hashem. A new efficient hybrid exact string matching algorithm and its applications. Life Science Journal. 2014;11(10).

[24] Naser Mustafa Abdul Sahib, Mohammed Faiz Aboalmaaly. Quick-skip search hybrid algorithm for the exact string matching problem. International Journal of Computer Theory and Engineering. 2012;4(2):259.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/17377