# Fast Hybrid String Matching Algorithm based on the Quick-Skip and Tuned Boyer-Moore Algorithms

Sinan Sameer Mahmood Al-Dabbagh
Department of Parallel and Distributed Processing
School of Computer Sciences Universiti Sains Malaysia
(USM), 11800 Pulau Pinang,
Malaysia

Mustafa Abdul Sahib Naser
Department of Software Engineering and Information Technology,
Al-Mansour University College,
Baghdad, Iraq

Nuraini bint Abdul Rashid (PhD)
Associate Professor, Department of Computer Sciences,
College of Computer & Information Sciences,
Princess Nourah bint Abdulrahman University, KSA.

Nawaf Hazim Barnouti
Al-Mansour University College
Baghdad,
Iraq

***Abstract*—The string matching problem is considered as one of the most interesting research areas in the computer science field because it can be applied in many essential different applications such as intrusion detection, search analysis, editors, internet search engines, information retrieval and computational biology. During the matching process two main factors are used to evaluate the performance of the string matching algorithm which are the total number of character comparisons and the total number of attempts. This study aims to produce an efficient hybrid exact string matching algorithm called Sinan Sameer Tuned Boyer Moore-Quick Skip Search (SSTBMQS) algorithm by blending the best features that were extracted from the two selected original algorithms which are Tuned Boyer-Moore and Quick-Skip Search. The SSTBMQS hybrid algorithm was tested on different benchmark datasets with different size and different pattern lengths. The sequential version of the proposed hybrid algorithm produces better results when compared with its original algorithms (TBM and Quick-Skip Search) and when compared with Maximum-Shift hybrid algorithm which is considered as one of the most recent hybrid algorithm. The proposed hybrid algorithm has less number of attempts and less number of character comparisons.**

***Keywords*—*Hybrid algorithm; string matching algorithm; Tuned Boyer-Moore algorithm; quick-skip search algorithm; Sinan Sameer Tuned Boyer Moore-Quick Skip Search (SSTBMQS)***

## I. INTRODUCTION

String matching, which involves locating all occurrences of a particular pattern in a large text, is considered one of the primary problems in computer science. Basically, the string matching algorithm accepts two inputs, namely, a short string called a pattern and a long string called a text. The pattern string is usually compared with the text string to determine if the former is a substring of the latter [1], [2]. Although many algorithms and strategies have been developed to solve this problem, scientists still attempt to develop far more efficient methods. String matching algorithms are extensively employed in different computer applications, such as information retrieval, DNA sequence, Web search engines, and artificial intelligence [3].

In the last two decades, string matching algorithms have elicited considerable attention, particularly when applied in various computer applications, such as text processing, DNA analysis, antivirus software, and anti-spam software. Such amount of attention may be attributed to the rapid growth of technology [4]. Current improvements in existing technologies pose numerous challenges to string matching algorithms [5]. String matching algorithms are of two types: exact and approximate string matching [4]. This research focuses on on-line exact string matching algorithms.

String matching algorithms are the basic components of existing applications, such as text processing, intrusion detection, search analysis, information retrieval, and computational biology [6]. All these applications involve a large amount of data because of the advancement in technology; moreover, all these applications involve different types of alphabets. Therefore, researchers continue to reiterate the need for significant string matching algorithms that can address different types of alphabets and large amounts of data [7].

Hybrid string matching approach was introduced to overcome the limitation of existing exact string matching algorithms. The former involves merging two or more algorithms. The Quick-Skip Search hybrid algorithm and Tuned Boyer-Moore algorithm are suitable for identifying all appearances of a pattern in a large text. However, both algorithms have limitations. The Quick-Skip Search hybrid algorithm consists of Skip Search and Quick Search algorithms. The latter exhibits good efficiency when large alphabets with a small pattern are utilized in the comparison operation, whereas the former exhibits good performance when small alphabets and a long pattern are employed [8].

However, the Skip Search algorithm consumes much time when a short DNA pattern and protein database are employed [7]. By contrast, the Tuned Boyer-Moore algorithm consumes much time when a long pattern of DNA alphabet is employed [7]. This algorithm has two disadvantages. First, it does not examine m−text characters to specify a starting search point as

the first step. Second, in the case of mismatch or entire pattern match, the shifting distance depends on a fixed shift value obtained in the preprocessing phase; this fixed shift value does not change until the text window reaches the end of the text. One of the advantages of this algorithm is that it checks the rightmost character in the text window as the first step before character comparison is implemented.

By contrast, the quick-skip search algorithm does not check the rightmost character in the text window as the first step before character comparison is implemented. The advantage of this algorithm is that it examines m − text characters to specify a starting search point as the first step; in the case of mismatch or entire pattern match, the shifting distance value depends on the Skip Search bucket and Quick Search bad character table. The larger shift value is adopted.

Owing to the contradictory behavior of the two algorithms, the important issue for this research is *"how to harness the significant advantages of the positive features of the two algorithms, overcome their performance weaknesses, and solve the string matching problem effectively during sequential and parallel stages for any alphabet type and any pattern length?"*

The remaining of the paper is structured as: Section 2 presents the look at of several hybrid algorithms. Section 3 the design principle of the proposed hybrid algorithm is discussed in detail. Moreover, an example is outlined in Section 4 to trace the hybrid algorithm. Section 5 discusses the experimental results of the hybrid algorithm when compared with its original algorithms and when compared with Maximum-Shift hybrid algorithm. In Section 6 summarizes the conclusion and suggests a future work that can be performed to improve the hybrid algorithm.

## II. Previous Works

Numerous studies on the string matching problem have been conducted continuously over the years to develop new efficient algorithms. These efficient algorithms are expected to reduce the work performed in each attempt, increase the amount of shift, and reduce the number of character comparisons during each attempt. Algorithms that acquire the positive properties and exclude the negative properties of original algorithms are called hybrid algorithms. The next subsections discuss some of these hybrid string matching algorithms.

SSABS algorithm [9] explained the advantage of combining two well-known exact string matching algorithms, namely, Quick Search and Raita. The new hybrid string matching algorithm exploits the fact that the dependency between neighboring characters is stronger than that between other characters. Therefore, putting off the comparisons of the neighboring characters would be better, which forms the fundamental idea of the new proposed algorithm. During the searching phase, which is similar to the Raita algorithm's searching phase, the rightmost character in the pattern is compared with the corresponding character in the text to determine if they match. The leftmost character in the pattern is then compared with the matched position character in the text. If they match, then the remaining characters are compared from right to left until a match or mismatch is observed in all $m-2$ characters. The shifting value to the sliding window after complete match or mismatch is determined based on the Quick Search bad character table.

Berry Ravindran-Fast Search (BRFS) algorithm, Yong [10] presented a new hybrid algorithm called BRFS by combining BR and Fast Search (FS) algorithms. Similar to most exact string matching algorithms, BRFS consists of preprocessing and searching phases. The preprocessing phase is constructed by computing the maximum shift value from BM good suffix shift (bmGs) and BR bad character (brBc) table. The searching phase depends on the searching method of the Fast Search algorithm, which performs comparison from right to left. After a complete match or mismatch, the sliding window shifts to the right side depending on the shift value provided by the preprocessing phase. The BRFS algorithm exhibits good performance in cases that involve small alphabets and long patterns. Hence, this algorithm is appropriate for use in applications related to biological sequence search.

Berry Ravindran-Skip Search (BRSS) Algorithm Berry Ravindran-Skip Search (BRSS) algorithm [11] is a combination of Berry Ravindran and Skip Search (SS) algorithms. The preprocessing phase consists of building the bucket list for the SS algorithm and the (brBc) table. The process to calculate the shift value in the preprocessing phase aims to have highest shift value to shift pattern throughout the searching phase. The combination of the two algorithms improves the other's weaknesses. The BR algorithm provides optimum shift values through the use of two successive characters positioned after the rightmost character of the text window. However, the BR algorithm does not examine m-text characters to specify a starting search point as a first step. By contrast, the SS algorithm begins by examining m-text characters during the searching phase to assign the starting search point in the text characters prior to the matching process. The drawback of the SS algorithm comes from using all the locations of the examining character in the bucket list table in case of a match or mismatch. The BRSS hybrid algorithm shows the benefit of combining the two algorithms by reducing the total work performed in each attempt and the overall computational time.

Quick-Skip Search Hybrid Algorithm [12] developed another hybrid algorithm based on the Quick Search (QS) algorithm. The combination of Quick Search and Skip Search algorithms allows each algorithm to complement the other. The resultant algorithm is called Quick-Skip Search hybrid algorithm. Similar to the two original algorithms, the effectiveness of the resultant hybrid algorithm can be found in the preprocessing and searching phases. The preprocessing phase of the Quick-Skip Search hybrid algorithm consists of building two shifting value tables, namely, the bad character table for the Quick Search algorithm and the bucket list for the Skip Search algorithm. The searching phase of the hybrid algorithm depends on the original algorithms searching phase with some update related to matching operation (the searching process is performed in different orders). Throughout the searching phase, the decision on how much distance is required to shift the sliding window if a mismatch or a complete match is found between the pattern and text characters depends on selecting a large shift value from the Quick Search and Skip

Search shift values. The algorithm is effective for any alphabet type and pattern length.

Quick Search, Zuh-Takaoka and Boyer Moore-Horspool (Maximum-Shift) Algorithm [13] proposed a hybrid string matching algorithm called Maximum-Shift hybrid algorithm. This new hybrid algorithm is a combination of three existing algorithms, namely, QS, ZT, and Horspool. The Maximum-Shift hybrid algorithm consists of three phases: preprocessing, maximum shift, and searching phases. The pre-processing phase, which preprocesses the input pattern to be useful during the matching process, consists of building the shifting value tables of the QS bad character table and the (ztBc) table.

The two inputs to the maximum shift phase are the QS and ZT shift values. The output from this phase is considered the maximum shift value between these two inputs to shift the pattern to a longer distance and subsequently reduce the number of attempts and the total number of character comparisons. The searching phase depends on QS and modified Horspool algorithms. The searching phase of the Maximum-Shift hybrid algorithm follows the searching phase orders of the QS algorithm with some updates related to the matching process.

During the searching phase, the hybrid algorithm searches the text string from left to right and utilizes the idea of the Horspool algorithm with a slight modification by comparing the two rightmost characters of the pattern with the text window characters as an initial step before searching the remaining characters P[m −2] from left to right. The algorithm produces better results compared with the three original algorithms and also when compared with another two hybrid algorithms (BR and Smith) in terms of minimizing the number of attempts and character comparisons [13].

The author [14] in 2017 proposed a new hybrid algorithm its name ABSBMH, which is a result of combining the good features of the two well know algorithms the modified Horspool and SSABS hybrid algorithms, which are a single and hybrid algorithm respectively. In the preprocessing phase the ABSBMH hybrid algorithm generates the Quick Search bad character table (qsBc) as the SSABS algorithm do which is beneficial to calculate the shifting distance during the searching phase. In the searching phase the ABSBMH algorithm depending on the SSABS and modified Horspool searching phase algorithms, the ABSBMH algorithm inspects not only the rightmost character in the text window, but it checks the last two characters in the text window with its corresponding position in the pattern characters to inspect if it matches or mismatch, if it matches the algorithm start search the remaining characters from left to right.

## III. The Proposed Algorithm

The contribution of this research is discussed in this section, that is, a solution to the string matching problem that involves proposing a sequential hybrid algorithm that blends two existing algorithms to develop an efficient sequential hybrid algorithm.

The proposed hybrid algorithm, SSTBMQS algorithm, is the key point of this study. This algorithm comprises two phases: the preprocessing and searching phases. In the preprocessing phase, the pattern characters are preprocessed to collect information to be used in the searching phase to decrease the number of characters compared and the number of attempts. The preprocessing and searching phases, which consist of seven steps for the proposed hybrid algorithm, are summarized in the next subsections, as shown in Fig. 1.

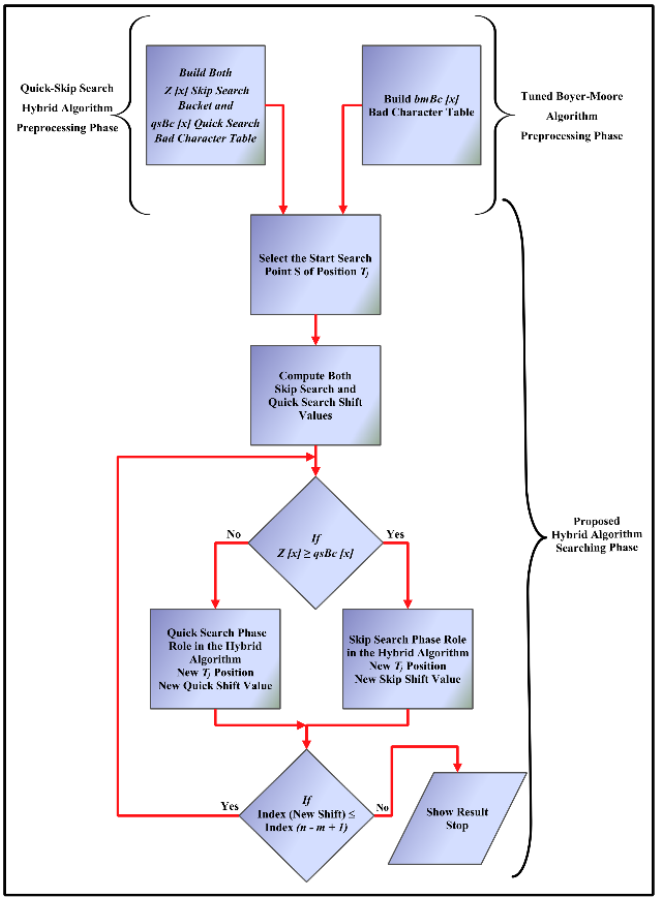


Fig. 1. Flowchart for SSTBMQS hybrid algorithm overview.

### A. Preprocessing Phase

To construct the preprocessing phase for SSTBMQS algorithm, the preprocessing phase for the Quick-Skip Search hybrid algorithm and the Tuned Boyer-Moore algorithm must be built first. The Quick-Skip Search hybrid algorithm preprocessing phase consists of building the QS bad character table and the SS bucket separately. The two preprocessing phases were not combined into one preprocessing phase because of the reverse behavior of the preprocessing phase for the QS and SS algorithms. Bad character table of QS stores all the rightmost indexes for each character in the pattern. The SS bucket contains all the leftmost indexes for each character in the pattern. Moreover, the bad character table for the Tuned Boyer-Moore algorithm consists of all the first rightmost appearances for each character in the pattern after scanning and indexing the pattern from right to left, starting with the rightmost character in the pattern, which always has the index 0. As a result, the preprocessing phase of SSTBMQS algorithm builds the preprocessing phase from each original algorithm.

*B. Searching Phase*

The searching phase of SSTBMQS algorithm depends on the technique used in the original algorithms, such as searching using different orders, with some development during the matching operation. During the matching operation, if a mismatch or an entire pattern string match occurs, the algorithm shifts the pattern according to the shift value from the original Quick-Skip Search hybrid algorithm. Basically, the searching phase of the hybrid algorithm follows these seven steps:

**Step 1:** Similar to the original Quick-Skip Search hybrid algorithm, this stage starts by examining the $m-text$ characters to specify a possible starting search point *S*. The starting search point has a $T_j$ position in the text characters, where both *j* and the pattern length *(m)* have the same size. Initially, after selecting the $T_j$ position in the text characters, At this point, the SSTBMQS algorithm starts performing the alignment operation between the text characters and the pattern characters in such a way that permits the alignment of the character at the $T_j$ position with its corresponding location in the bucket list. Also, as an initial step the SSTBMQS algorithm computes the shift value of the Quick-Skip Search hybrid algorithm, which contains the shift value for the Skip Search phase and Quick Search phase, because the underlying structure of SSTBMQS algorithm has two searching phases (Skip Search and Quick Search).

The SSTBMQS algorithm examines both shift values and uses the searching phase, which has the maximum shift value. When the alignment between the text characters and the pattern characters is performed, the character located at the $T_j$ position does not appear in the pattern characters. Thus, the algorithm continually shifts the pattern characters to the following $T_j$ position in the text character. This operation skips numerous unnecessary attempts, consequently minimizing the total number of character comparisons and avoiding the alignment of the leftmost character of the pattern with the leftmost character of the text at the beginning of the searching phase.

**Step 2:** The SSTBMQS algorithm calculates the $T_d$ value, where *d* is the difference between the $T_f$ position and the $T_j$ position. $T_f$ is the position of the inspection character, where *f* is the location of the last character in the $m-text$ characters. The $T_f$ position is determined in the next step. The $T_d$ value is calculated depending on two circumstances:

*1)* If the character at position $T_j$ occurs in the last position in the bucket, $T_d$ is calculated from Equation (1) after being subtracted from the last character index, which is equal to the pattern length minus one $(m-1)$ from the last $T_j$ position in the bucket. Then, the algorithm moves directly to **Step 3**.

$$T_d = (m-1) - (\text{The last } T_j \text{ position in the bucket}) \quad (1)$$

*2)* Whenever the character at position $T_j$ is not at the last position in the bucket, the $T_d$ value is calculated using Equation (2) after subtracting the last character index, which is equal to the pattern length minus one $(m-1)$ from the current $T_j$ position of the bucket.

$$T_d = (m-1) - (\text{The current } T_j \text{ position of the bucket}) \quad (2)$$

This process continues to execute until all positions of the character at $T_j$ position in the bucket are processed. The algorithm moves to the **Step 3**.

**Step 3:** This step comes after determining the $T_d$ value in **Step 2**. In this step, the algorithm determines the location of $T_f$, where *f* is the location of the last character in the $m-text$ characters, which is often called the inspection character. To determine the location of $T_f$, the SSTBMQS algorithm adds the $T_d$ value to the current position of $T_j$ in the text characters, as shown in Equation (3).

$$T_f = (\text{The current position of } T_j \text{ in the text characters}) + T_d \quad (3)$$

**Step 4:** The SSTBMQS algorithm verified whether a match is possible between the pattern and the text characters by checking the inspection character (which occurs at $T_f$ in the text). If the value of this character after referring to the (bmBc[a]) table is equal to (0) (that is, the last character in the pattern matches its corresponding character at the $T_f$ position in the text), where the value (0) in the (bmBc[a]) table is given only for the rightmost character in the pattern, The most significant property of the Tuned Boyer-Moore algorithm is the unique zero value given to the rightmost character in the pattern. Therefore, the value of the rightmost character in the (bmBc) table is always (0). The algorithm moves to **Step 5**. Otherwise, the algorithm goes to **Step 6**. By performing this process, the algorithm verifies whether a match is possible between the rightmost character in the pattern and its corresponding character at the $T_f$ position in the text without opening the text window and without performing a comparison operation. The latter is considered the most costly portion of a string matching algorithm, that is, when the algorithm verifies whether the character in the pattern occurs in the text window [15]. This process will reduce the number of character comparisons, as well as the number of attempts.

**Step 5:** This step is accomplished if the inspection character at $T_f$ equals (0) from the (bmBc[a]) table. Thus, a match between the pattern and the text characters is possible. At this step, comparisons occur between the pattern and text characters by opening a text window that is equal to pattern length *(m)*. The first comparison operation is performed from the leftmost character in the pattern to the corresponding character position in the text window to the right side. If a mismatch or a complete pattern match occurs, the SSTBMQS algorithm moves to the following step.

**Step 6:** In this step, the SSTBMQS algorithm computes the shift values for both the SS and QS algorithms. SSTBMQS hybrid algorithm computes the SS shift value in different ways depending on two circumstances:

*1)* When the SSTBMQS algorithm checks the character at the $T_j$ position and determines that the character appears in the last location of the bucket, the SS value is computed using (4) after the first bucket location of the character that appears in the next $T_j$ position is distinguished in the text characters. This position is considered the following start search point.

$$SS\_shift = m + \text{current } T_j \text{ position (from bucket)} - \text{the next } T_j \text{ position} \quad (4)$$

*2)* When the $T_j$ position does not appear in the last location of the bucket, the SS shift value is computed by using a subtracting operation performed between the following location value from the current location value of this character in the bucket.

SSTBMQS algorithm is used to compute the QS shift value depending on the character that follows the rightmost character of the text window. This character is used as an index that refers to the shift value stored in the QS bad character table, which represents the value of the rightmost occurrence of this character in the pattern.

SSTBMQS algorithm has two searching phases (Skip Search and Quick Search). At this point, SSTBMQS algorithm computes the shift values of the Skip Search and Quick Search phases. SSTBMQS algorithm examines both shift values and uses the searching phase, which has the maximum shift value. In other words, if the SS shift value is larger than or equal to the QS shift value, SSTBMQS algorithm depends on the Skip Search phase and goes to **Step 2**, as shown in Fig. 2. Otherwise, if the QS shift value is larger, then the SSTBMQS algorithm goes to **Step 7**.

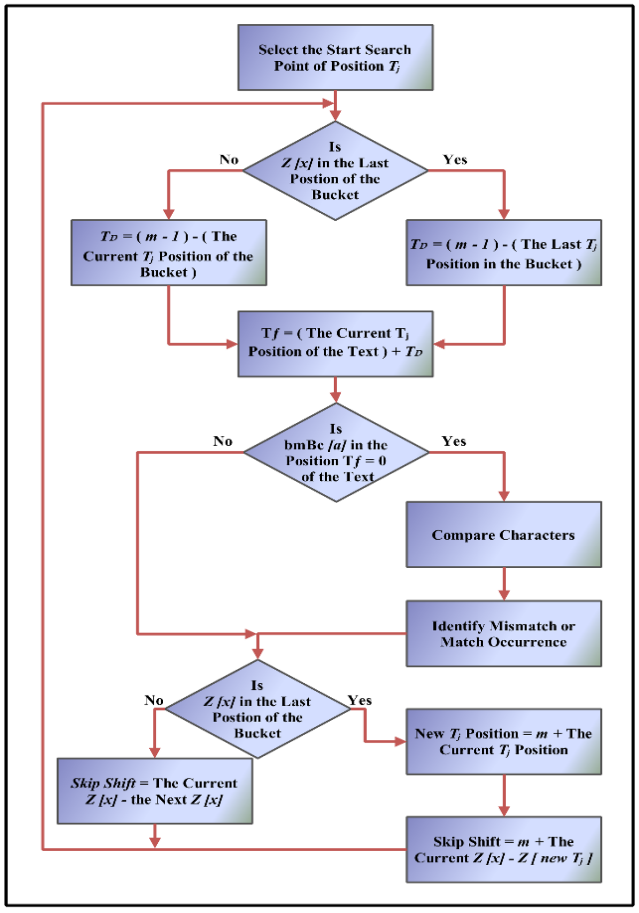


Fig. 2. Flowchart for Skip Search Phase Role in the SSTBMQS algorithm.

**Step 7:** This step is employed when SSTBMQS algorithm depends on the Quick Search phase. The Quick Search phase computes the $T_j$ position depending on two circumstances.

*1)* When the value of the character is positioned next to the rightmost character of the text window is lower than or equivalent to pattern length *(m)*, the new $T_j$ position computes the current $T_j$ position in the text character to become equivalent to that positioned immediately next to the window, which is considered to be the new beginning search point. Then, the algorithm moves to **Step 2**, as presented in the following condition.

***If (QS_Shift > SS_Shift) & (QS_Shift ≤ m)***

***Then***

***New $T_j$ Position = First Position after the Window***

*2)* When the Quick Search phase checks the shift value of the character that follows the rightmost character of the text window and it is larger than pattern length *(m)*, the new $T_j$ position is computed by summing the current $T_j$ position in the text and is made equal to the character position immediately next to the text window plus the value of pattern length *(m)* as presented in the following condition.

***If (QS_Shift > SS_Shift) & (QS_Shift > m)***

***Then***

***New $T_j$ Position = First Position after the Window + m***

However, after the new $T_j$ position is computed, if the character positioned at the new $T_j$ position does not appear in the pattern characters, SSTBMQS algorithm continually shifts the pattern to the following potential beginning search point, and SSTBMQS algorithm goes into **Step 2**. Fig. 3 shows the functionality of the Quick Search phase throughout the searching phase of SSTBMQS algorithm. All of the steps of the searching phase are repeated until the window is placed beyond *(n − m + 1)*.

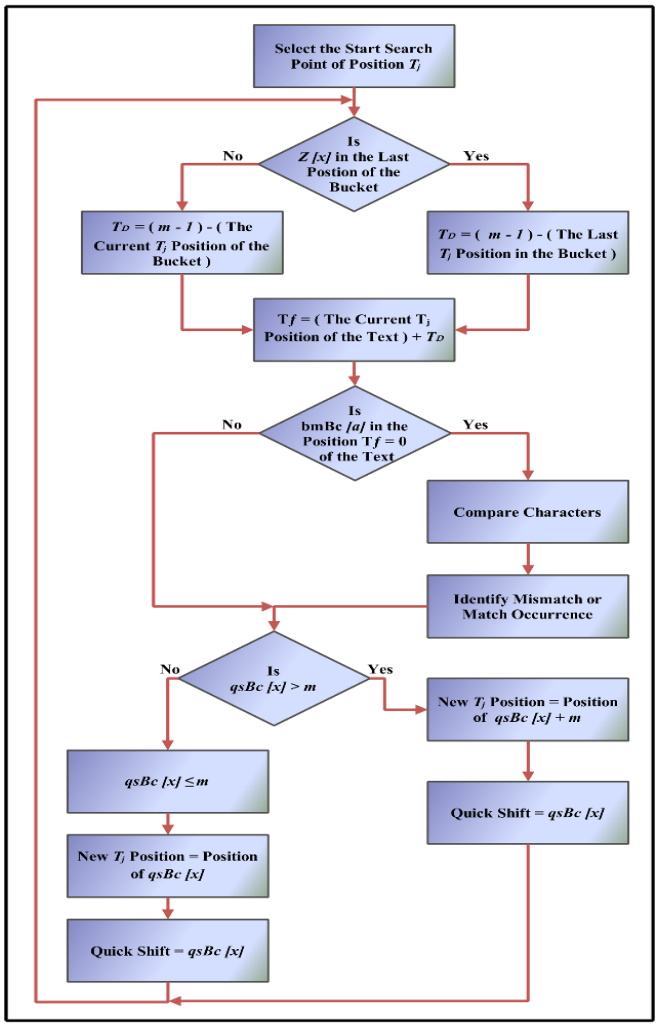


Fig. 3. Flowchart for Quick Search Phase Role in the SSTBMQS algorithm.

## IV. SSTBMQS Algorithm Tracing Example

This section demonstrates an example of tracing by using SSTBMQS algorithm. The example shows the steps of the preprocessing and searching phases of SSTBMQS algorithm. Two strings are used as input: text and a pattern, as displayed in Fig. 4.

*Text = " A D F T C D C T D A D A A T A A M A C A G C T T C A C G A C D F C C A C G A*

*C M A C "*

*Pattern = "A C G A C M A C "*

*The text length n = 42*

*The pattern length m = 8*

*The alphabet set* $\sum = (A, C, D, F, G, M, T)$ *of size* $\sigma = 7$

Fig. 4. Algorithm Inputs.

The preprocessing phase of SSTBMQS algorithm is built by constructing the pre-processing phase for the two original algorithms: The Quick-Skip Search hybrid algorithm and the TBM algorithm. The Quick-Skip Search hybrid algorithm is used to build the SS buckets and the QS bad character table, whereas the TBM algorithm is used to build the (bmBc[a]) bad character table for the input pattern, as shown in Fig. 5.

| X | Z[x] |
|---|---|
| A | (6, 3, 0) |
| C | (7, 4, 1) |
| D | Φ |
| F | Φ |
| G | (2) |
| M | (5) |
| T | Φ |

Skip Search Buckets

| X | A | C | D | F | G | M | T |
|---|---|---|---|---|---|---|---|
| qsBc[x] | 2 | 1 | 9 | 9 | 6 | 3 | 9 |

Quick Search Bad Character Table

| X | A | C | D | F | G | M | T |
|---|---|---|---|---|---|---|---|
| bmBc[x] | 1 | 0 | 8 | 8 | 5 | 2 | 8 |

Tuned Boyer-Moore Bad character Table

Fig. 5. Preprocessing phase.

The searching phase starts by choosing the start search point, which is at location $T_j$ in the text, as shown in Fig. 6.

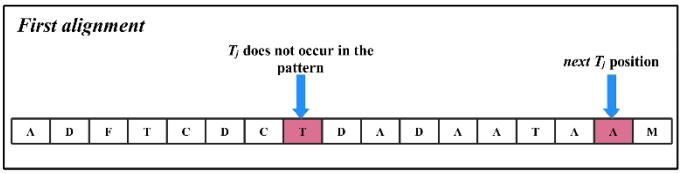


Fig. 6. First Alignment.

In the first alignment, the chosen beginning point (T) does not exist in the pattern. SSTBMQS algorithm checks the following potential starting point (A), as mentioned in **Step 1** of Section III. In the second alignment (see Fig. 7), the shift value is calculated by subtracting the next position value from the current position value of the character (A) that appears in the SS bucket, as explained in the second circumstance in **Step 6** in Section III.

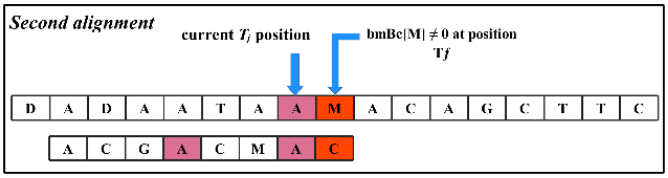


Fig. 7. Second Alignment.

***Shift = Skip Shift = 6 - 3 = 3***

SSTBMQS algorithm checks the character at position $T_f$ in the (bmBc[M] 6= 0), which is found to be unequal to 0. By performing this process, SSTBMQS algorithm avoids opening a text window and reduces the number of attempts, as well as the number of character comparisons, as explained in **Step 4** in Section III. SSTBMQS algorithm then computes the shift value of the SS shift value = 3, where the QS shift value = 2 from the QS bad character table. SSTBMQS algorithm depends on the SS shift value, as explained in **Step 6** in Section III.

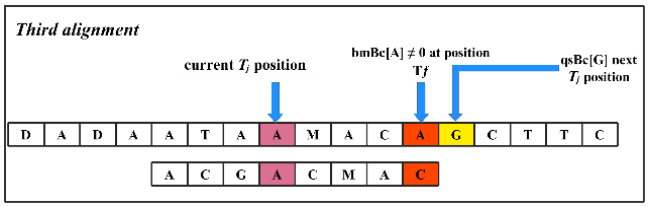


Fig. 8. Third Alignment.

***Shift = Quick Search bad character table = 6***

The third alignment shows a situation in which SSTBMQS algorithm depends on the QS shift value, which is equal to the position of the character (G) in the pattern (see Fig. 8). After checking the character (A) at position $T_f$ in (bmBc[A] 6= 0), as mentioned previously in **Step 4** in Section III, the new $T_j$ position becomes equivalent to the position of the character (G) in the text characters, as explained in the first circumstance of **Step 7** in Section III.

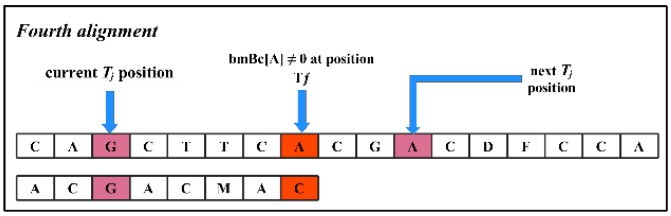


Fig. 9. Fourth Alignment.

***Shift = Skip Shift = 8 + 2 - 6 = 4***

The value of the character (A) in the TBM bad character table is equal to (1). Thus, a match between the pattern and the text characters is impossible (see Fig. 9). The SSTBMQS algorithm computes shift value depending on the shift value from the SS bucket. The (G) character at position $T_j$ appears in the last position of the bucket. SSTBMQS algorithm computes the shift value by adding the value of the (G) character in the SS bucket to the pattern length *(m)*. Then, the summation is subtracted from the first value of the character at position $T_j$ from the bucket, as explained in the first circumstance of **Step 6** in Section III.

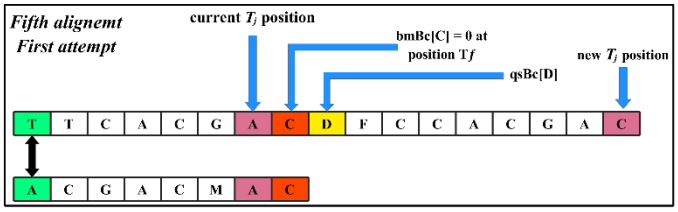


Fig. 10. Fifth Alignment and First Attemp.

***Shift = Quick Search bad character table = 9***

In the fifth alignment the character (C) at position $T_f$ equals (0) in the (bmBc[c] = 0) bad character table. Thus, matching can possibly occur between pattern and text characters. SSTBMQS algorithm opens a text window and starts comparing characters from left to right, considering the first attempt, as shown in Fig. 10. After a mismatch occurs, SSTBMQS algorithm computes both SS and QS shift values. The QS shift value becomes larger than the SS shift value. By computing for the shift value of the character (D), which is equal to (9) in the QS bad character table, the QS shift value becomes larger than the pattern length *(m)*. Thus, the new $T_j$ position becomes equal to the summation of the current (qsBc[D]) and the pattern length *(m)*, as explained in the second circumstance of **Step 7** in Section III.

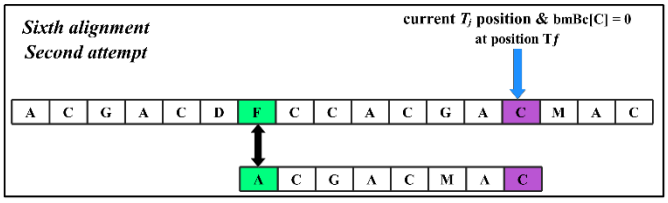


Fig. 11. Sixth Alignment and Second Attempt.

***Shift = Skip Shift = 7 - 4 = 3***

The sixth alignment shows a situation in which the pattern aligns its character (C) at position $T_j$ that is, at the same time, the $T_f$ position. After examining the (bmBc[C]=0), SSTBMQS algorithm opens a text window and starts comparing characters from left to right, considering the second attempt (see Fig. 11). SSTBMQS algorithm depends on the SS shift value by subtracting the next position value from the current position value of the character (C) in the SS bucket, as explained in the second circumstance of **Step 6** in Section III.

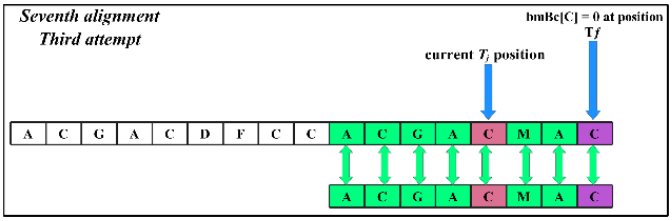


Fig. 12. Seventh Alignment and Third Attempt.

In the seventh alignment, SSTBMQS algorithm first examines the value of the character (C) at the $T_f$ position in the TBM bad character table (bmBc[C] = 0) and finds it to be equal to (0). Then, SSTBMQS algorithm starts comparing the characters from left to right until all characters' match, considering the third attempt as explained in Fig. 12.

***Number of attempts = 3***
***Number of characters' comparison = 10***

## V. Results and Discussion

This section presents the experimental design adopted in this study for the sequential version of SSTBMQS algorithm. The execution of the sequential program, performance and evaluation are discussed.

### A. Experimental Databases

DNA sequence, protein sequence, and English text datasets are used in this work to evaluate the results of the sequential version of SSTBMQS algorithm with the two original algorithms. Such datasets are chosen because they are defined

as a benchmark standard that demonstrates the typical utilization of string matching applications. They also vary in alphabet size; thus, a variety of algorithm behavior with different alphabet sizes can be evaluated. The data size chosen for testing the sequential behavior of SSTBMQS algorithm with the two original algorithms is 100 MB.

*a) DNA Sequence*

DNA sequence is created from a long string that holds hereditary information arranged in a sequence of four nucleotides represented by four uppercase letters. Usually, adenine is indicated by (A), thymine is indicated by (T), guanine is indicated by (G), and cytosine is indicated by (C) [$\sum$ = (A, C, G, T) and $\sigma = 4$]. To examine the algorithm behavior in a small alphabet size, DNA sequence is considered in this study. Database is downloaded from Gutenberg Project [16].

*b) Protein Sequence*

Protein sequence is composed of 20 amino acids indicated by uppercase characters [$\sum$ = (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y) and $\sigma = 20$]. Protein sequence has an essential responsibility in biochemistry science, especially in protein structure and functionality.

*c) English Text*

English text data type comprises over 100 various alphabet types split into English language (lowercase and uppercase), numbers, and samples. The large size of this alphabet data type allows testing the algorithm behavior in such a large dataset. This data type is gathered from the Gutenberg Project [16].

### B. Performance and Evaluation

The major goal of this research is to offer an effective algorithm to be utilized basically with different string matching applications. Two common factors are typically considered to evaluate the performance of a string matching algorithm with different applications [17]. The two factors are presented below.

*a) Total Number of Character Comparisons*

This factor refers to the summation of exact comparison that occurs between the pattern and characters of text window. The algorithm with a significantly less number of character comparisons is identified as a powerful algorithm with better performance.

*b) Total Number of Attempt*

This factor denotes the distance that a pattern needs to skip along the entire assigned text. When the amount of attempts is significantly less, the overall performance of an algorithm is better. These two factors are used as a basis to evaluate the efficiency of the sequential version of SSTBMQS algorithm and to specify its overall performance with various datasets implemented.

### C. Sequential Program Execution

The sequential program of SSTBMQS hybrid algorithm with the two sequential original string matching algorithms (i.e., TBM and Quick-Skip Search) is examined on each kind of dataset outlined in part (A) of Section V. The three algorithms are run using a personal computer with 2.4 GHZ Inter®Core™with 7 cores and 8 GB RAM. The operating system used is Microsoft Windows 8 Single language, which is a 64-bits operating system. Microsoft visual studio 2010 is utilized to write down the codes. The compiler used to build and run the codes is visual C++ compiler.

This section elucidates the evaluation results acquired from executing the sequential programs of SSTBMQS algorithm when compared with TBM and Quick-Skip Search hybrid algorithms. As indicated in Table 1, TBM is a single algorithm used in developing SSTBMQS algorithm of this study. Quick-Skip Search and Maximum-Shift are both hybrid algorithms.

Quick-Skip Search is used in developing SSTBMQS algorithm. Maximum-Shift consists of QS, BMH, and ZT algorithms. The results of Maximum-Shift are compared with those of SSTBMQS algorithm and the two chosen string matching algorithms. This hybrid algorithm is chosen for comparison because it is considered as one of the latest hybrid algorithm in the literature. The QS algorithm that is included in developing Quick-Skip Search hybrid algorithm is also used to develop the Maximum-Shift hybrid algorithm. All the algorithms indicated above are elucidated in Section II of this research.

These algorithms are evaluated according to the total numbers of character comparisons and number of attempts. As mentioned previously in part (A) of Section V, various kinds of datasets are employed, which are DNA sequence, protein sequence, and English text. The patterns are selected randomly from the words inside each dataset and have various lengths that range from 8 to 100 [10], where 8 and 10 are short patterns; 20, 30, 40, 50, 60, 70, 80, 90, and 100 are long patterns [13]. Each pattern length is searched five times, and the average is obtained. The Maximum-Shift hybrid algorithm results are generated from [13] (Table 1).

TABLE. I. RELATIONSHIP AMONG TBM, QUICK-SKIP SEARCH, MAXIMUM-SHIFT, AND THE PROPOSED ALGORITHM

| ***Algorithms*** | ***Algorithm Type*** | ***Underlying Structure*** | ***Relationship with the Proposed Algorithm*** |
|---|---|---|---|
| Tuned Boyer-Moore (TBM) | Single algorithm | TBM | Used in the preprocessing and searching phases |
| Quick-Skip Search algorithm | Hybrid algorithm | QS+Skip Search | Used in the preprocessing and searching phases |
| Maximum-Shift (Max-Shift) | Hybrid algorithm | QS+BMH+ZT | Used the QS in the preprocessing phase |

### D. Analyzing Number of Character Comparisons

Based on the empirical results presented in Fig. 13 to 15, DNA alphabet delivers a great number of character comparisons, especially when the size of pattern length is short. This behavior is due to the structure nature of the DNA alphabet itself, which considers a small alphabet size. The DNA alphabet structure consists of four characters, thereby leading to a small shift distance of pattern during the searching operation of pattern string into text string. The use of a small

size of alphabet in implementing algorithms causes considerable exact matching between inspected pattern string and text window, particularly when utilizing short pattern lengths. Subsequently, the amount of character comparisons is influenced by the size of the alphabet used.

For all algorithms with all dataset types, the results show that when pattern lengths increase, the total number of character comparisons decreases significantly. This behavior is due to the increasing amount of shift distance provided by the algorithms when a mismatch occurs. Based on this observation, the DNA dataset is excluded, especially for TBM algorithm. The performance of TBM algorithm in DNA dataset shows an unstable behavior, which is due to a fixed shift value provided by this algorithm that leads to a small shift of pattern after a mismatch occurs. Furthermore, DNA dataset generates small numbers in TBM bad character table, which leads to a small shift of pattern during an unrolled operation in each attempt. This condition can be considered another reason for the unstable behavior provided by TBM algorithm that tends to increase the total number of character comparisons.

In protein and English datasets, the performance of Quick-Skip Search hybrid algorithm surpasses that of Maximum-Shift hybrid algorithm when a short pattern length is used. This result is ascribed to that the Maximum-Shift hybrid algorithm starts the searching phase without checking $T_j$ starting point. The probability of $T_j$ position character taking place in pattern characters is low when using medium and large alphabet sizes for protein and English datasets, respectively. The results of protein and English datasets also show that the Maximum-Shift hybrid algorithm beats the Quick-Skip Search hybrid algorithm in 30 to 100 pattern lengths.

This behavior is due to employing both QS and ZT preprocessing phases to obtain a maximum shift distance to shift a pattern when a mismatch or a complete match occurs. The largest shift value can be obtained from the QS algorithm preprocessing phase, which is equal to the pattern length plus one, when the character following the rightmost character of text window is not occurring in the pattern characters. However, ZT preprocessing phase depends on two consecutive rightmost characters in the text window to calculate the shift distance value. Using these methods avoids many unnecessary potential numbers of character comparisons during the matching process of Maximum-Shift hybrid algorithm.

Quick-Skip Search hybrid algorithm utilizes QS algorithm preprocessing phase with Skip buckets to determine the next position of pattern string in text string. Maximum-Shift hybrid algorithm strongly beats Quick-Skip Search hybrid algorithm when 30 to 100 pattern lengths are used. The Quick-Skip Search hybrid algorithm uses the Skip buckets with the QS algorithm in the preprocessing phase. The disadvantage of the Skip Search algorithm is used all the positions of the character at position $T_j$ in the bucket list in case of match or mismatch occurs. On the contrary, the Maximum-Shift hybrid algorithm uses the preprocessing phase of ZT algorithm, which is viewed as a highly effective algorithm with small alphabet size data type.

SSTBMQS algorithm outperforms all other algorithms by producing a less number of character comparisons for all pattern lengths and data types. Such a good performance is due to three reasons. First, the algorithm employs Quick-Skip Search preprocessing phase to determine the next position of a pattern in text string after a mismatch occurs. Second, the algorithm starts the searching phase by checking the occurrence of character at Tj position in pattern characters, which is considered a starting search point before actual comparison. Third, the algorithm employs modified TBM matching operation characteristic by checking the character at $T_f$ position before starting a comparison operation.

The results of the SSTBMQS algorithm are better than those of the two original algorithms and Maximum-Shift hybrid algorithm in all pattern lengths and data types. The good performance of the SSTBMQS algorithm implies that the integration of the two original algorithms provides a new hybrid algorithm with better performance.

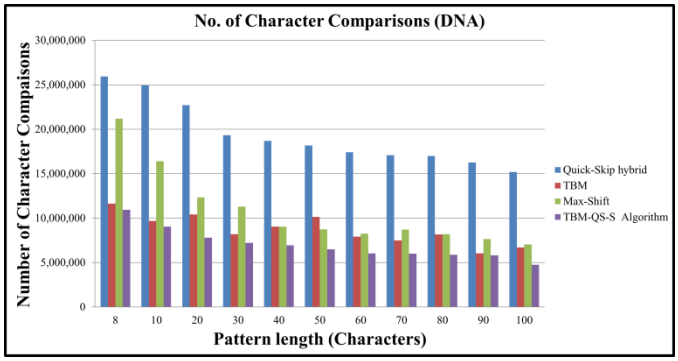


Fig. 13. Number of Character Comparisons in DNA Sequence Data.

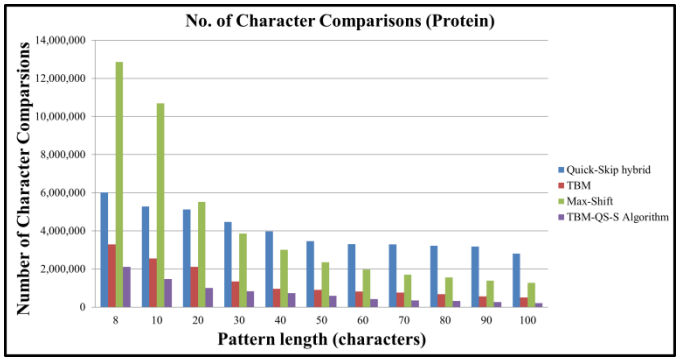


Fig. 14. Number of Character Comparisons in Protein Sequence Data.

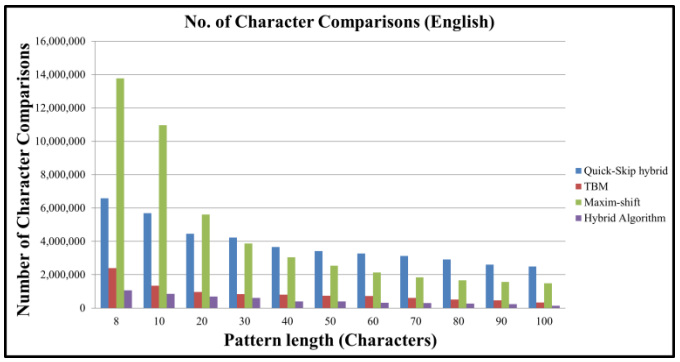


Fig. 15. Number of Character Comparisons in English Text Data.

### E. Analyzing Number of Attempts

The empirical results presented in Fig. 16 to 18, show the behavior of TBM, Quick-Skip Search, Maximum-Shift, and SSTBMQS algorithms when using with DNA sequence, protein sequence, and English text data types. The results show that all the algorithms have a stable behavior in medium and large sizes for protein and English alphabets, respectively. Generally, the total number of attempts is decreased when pattern lengths increase. In short, pattern lengths and the Quick-Skip Search hybrid algorithm outperform the Maximum-Shift hybrid algorithm. This result is due to the characteristic of the Quick-Skip Search hybrid algorithm, that is, it starts the searching phase by checking the probability of occurrences of the character at $T_j$ position in the pattern characters. This probability decreases when alphabet size increases. Hence, the Quick-Skip Search hybrid algorithm surpasses the Maximum-Shift hybrid algorithm in a short pattern length with protein and English alphabets.

The Maximum-Shift hybrid algorithm does not start the searching phase by checking $T_j$ position to specify the starting search point. DNA alphabet is accordingly excluded because of its small size, which consists of only four characters, and the ordering of the characters in the pattern itself, which increases the probability of occurrences of the character at $T_j$ position in the pattern. Thus, a small shift distance to the pattern is generated across the text string. The Maximum-Shift hybrid algorithm generally outperforms the Quick-Skip Search hybrid algorithm in long pattern, especially from 30 to 100 pattern lengths. This result is due to using ZT preprocessing function, which uses two rightmost characters at the text window to compute the shift distance and is considered as a powerful function with small alphabets.

The TBM algorithm shows a stable behavior in protein and English alphabets by decreasing the total number of attempts, which is related to the alphabet size of the dataset being used. DNA dataset is excluded because of the size of DNA alphabet, which increases the probability of finding inspection character, which is the position of the rightmost character in text window that is equal to zero in the TBM bad character table. This behavior will lead to many exact matching processes between the pattern and text characters, which contributes in increasing the total number of attempts.

The results of the SSTBMQS algorithm indicate that it outperforms all other algorithms in all pattern lengths and with any alphabet sizes. This good behavior is related to its good properties acquired from integrating the two original algorithms. The hybrid algorithm starts the searching phase by checking the occurrence of the character at $T_j$ position in the pattern characters. When the size of the alphabet used is large, the probability of the character occurrence at $T_j$ position is low. The Quick-Skip Search preprocessing function is used to compute a maximum shift distance to shift the pattern long distance when a mismatch or a complete pattern match occurs.

Before performing matching operations, the character at $T_f$ position is checked that is the position of the rightmost character at the text window. If the character at $T_f$ position equals to zero in the TBM bad character table, then the SSTBMQS algorithm starts a character comparison. If the character at $T_f$ position is not equal to zero value, then the SSTBMQS algorithm skips opening text window and starts character comparison. This technique contributes in reducing the total number of attempts. Therefore, SSTBMQS algorithm utilizes the significant advantages and excludes the disadvantages of the two original algorithms by producing a minimal number of attempts.

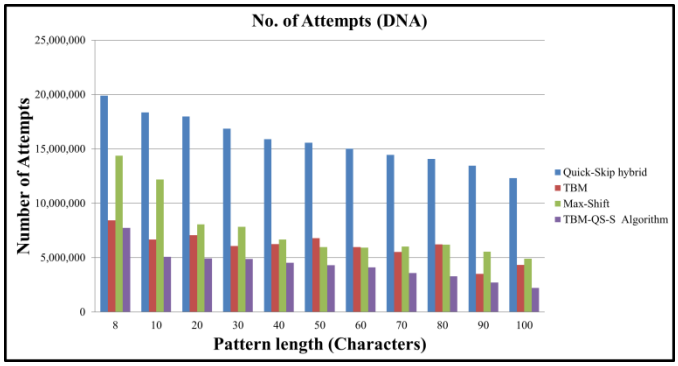


Fig. 16. Number of Attempts in DNA Sequence.

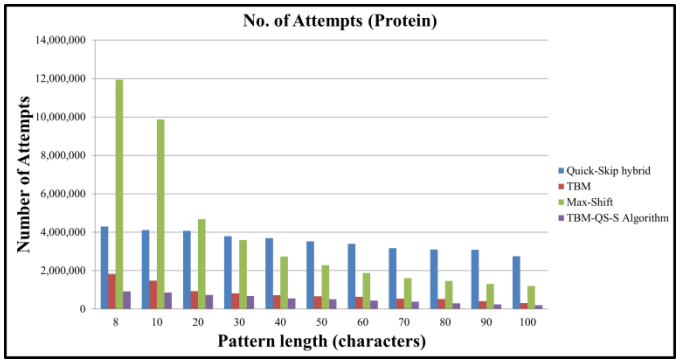


Fig. 17. Number of Attempts in Protein Data Type.

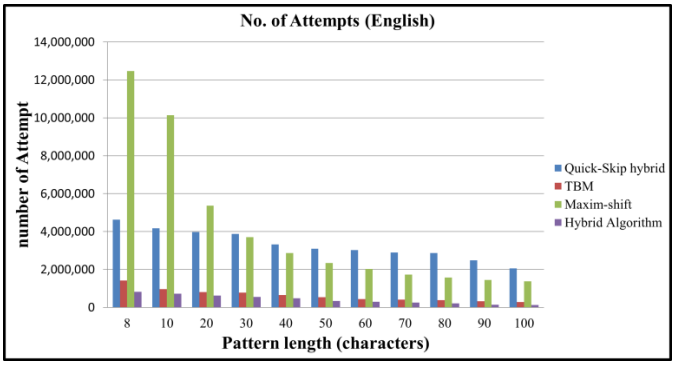


Fig. 18. Number of Attempts in English Text Data Type.

## VI. Conclusion and Future Work

This section presents the conclusion from achieving the objective of this study, that is, integrating two existing algorithms (i.e., TBM and Quick-Skip Search) to produce an efficient hybrid string matching algorithm called SSTBMQS. Particularly, this study aims to extract the good properties from the two original algorithms. The SSTBMQS algorithm uses the shift values produced from the preprocessing phase of the

Quick-Skip Search hybrid algorithm to compute the next expected position of the pattern during the searching phase.

In the searching phase, the SSTBMQS algorithm examines *m−text* characters to specify a starting search point before the actual character comparisons. This process avoids performing many unnecessary attempts, which reduces the total number of attempts. The SSTBMQS algorithm also utilizes modified TBM matching operation during the searching phase by checking the character at $T_f$ position in the TBM bad character table before performing character comparisons and starting an attempt. Consequently, the total numbers of character compassions and attempts are significantly reduced.

Two parameters are used to evaluate the performance of the sequential version of the SSTBMQS algorithm, which are the total numbers of character comparisons and attempts. In Section V, the SSTBMQS algorithm is compared with three algorithms, namely, TBM and Quick-Skip Search as original algorithms and Maximum-Shift as hybrid string matching algorithm. Comparisons are performed in different datasets (DNA sequence, Protein sequence, and English text) with different pattern lengths. The SSTBMQS algorithm outperforms all other algorithms by producing fewer total numbers of attempts and character comparisons. In future work, the running time of SSTBMQS algorithm should be enhanced by parallelizing it on the GPU using CUDA library.

## REFERENCES

[1] Michailidis, Panagiotis D., and Konstantinos G. Margaritis. "On-line string matching algorithms: Survey and experimental results." *International journal of computer mathematics* 76.4 411-434, (2001).

[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* 4.13 (2016): 1.

[3] Raju, S. Viswanadha, A. Vinaya Babu, and M. Mrudula. "Backend engine for parallel string matching using boolean matrix." *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*. IEEE, 2006.

[4] Kumar, K. S. M. V., S. Viswanadha Raju, and A. Govardhan. "A Survey of Parallel Algorithms for Text Matching In Large Databases and Hardware Implementations." International Journal of Engineering and Innovative Technology (IJEIT) 1(2): 2277–3754, 2012.

[5] Hassan, Atif Agha. "Mixed heuristic algorithm for intelligent string matching for information retrieval." *Computational Intelligence and Multimedia Applications, 2005. Sixth International Conference on*. IEEE, 2005.

[6] Kun, Bi, et al. "A practical distributed string matching algorithm architecture and implementation." *World Acad Sci Eng Technol* 10 (2005): 1307-6884, 2005.

[7] AbdulRazzaq, A. A., Rashid, N. A., Hasan, A. A. and Abu-Hashem, M. A. The Exact String Matching Algorithms Efficiency Review, Global Journal on Technology 4(2): 576–589, 2013.

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

[9] Sheik, S. S., et al. "A fast pattern matching algorithm." *Journal of Chemical Information and Computer Sciences* 44.4, 1251-1256, 2004.

[10] Huang, Yong, et al. "A fast exact pattern matching algorithm for biological sequences." *BioMedical Engineering and Informatics, 2008. BMEI 2008. International Conference on*. Vol. 1. IEEE, 2008.

[11] Almazroi, Abdulwahab Ali. "A fast hybrid algorithm approach for the exact string matching problem via berry ravindran and alpha skip search algorithms." *Journal of Computer Science* 7.5 (2011): 644.

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

[13] Kadhim, Hakem Adil, and NurAini AbdulRashidx. "Maximum-shift string matching algorithms." *Computer and Information Sciences (ICCOINS), 2014 International Conference on*. IEEE, 2014.

[14] Al-Dabbagh, Sinan Sameer Mahmood, and Nawaf Hazim Barnouti. "A New Efficient Hybrid String Matching Algorithm to Solve the Exact String Matching Problem."

[15] Charras, Christian, and Thierry Lecroq. *Handbook of exact string matching algorithms*. King's College, 2004.

[16] Kärkkäinen, Juha, and Joong Chae Na. "Faster filters for approximate string matching." *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics, 2007.

[17] Thathoo, Rahul, et al. "TVSBS: A fast exact pattern matching algorithm for biological sequences." *Current Science* 91.1 (2006): 47-53.