Received March 21, 2019, accepted April 4, 2019, date of publication April 30, 2019, date of current version June 10, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2914071

Exact String Matching Algorithms: Survey, Issues, and Future Research Directions

SAQIB IQBAL HAKAK[1], AMIRRUDIN KAMSIN[1], PALAIAHNAKOTE SHIVAKUMARA[1], GULSHAN AMIN GILKAR[2], WAZIR ZADA KHAN[3], (Senior Member, IEEE), AND MUHAMMAD IMRAN[4]

[1]Faculty of Computer Systems and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]College of Computer Science, Shaqra University, Shaqra 11451, Saudi Arabia
[3]Faculty of Computer Science and Information Systems, Jazan University, Jazan 82822-6694, Saudi Arabia
[4]College of Applied Computer Science, King Saud University, Riyadh 11451, Saudi Arabia

Corresponding authors: Amirrudin Kamsin (amir@um.edu.my) and Wazir Zada Khan (wazirzadakhan@jazanu.edu.sa)

This work was supported in part by the University of Malaya Research Grant (UMRG) under Project RP043A-17HNE & RP003A 14HNE, and in part by the Deanship of Scientific Research, King Saud University through research group under Grant RG-1435-051.

ABSTRACT String matching has been an extensively studied research domain in the past two decades due to its various applications in the fields of text, image, signal, and speech processing. As a result, choosing an appropriate string matching algorithm for current applications and addressing challenges is difficult. Understanding different string matching approaches (such as exact string matching and approximate string matching algorithms), integrating several algorithms, and modifying algorithms to address related issues are also difficult. This paper presents a survey on single-pattern exact string matching algorithms. The main purpose of this survey is to propose new classification, identify new directions and highlight the possible challenges, current trends, and future works in the area of string matching algorithms with a core focus on exact string matching algorithms.

INDEX TERMS String matching, Boyer-Moore, Rabin-Karp, Knuth-Morris-Pratt, exact string matching, pattern matching, pattern recognition, pattern analysis.

I. INTRODUCTION

String matching is a universal technique for solving problems of different fields, such as text mining, natural language processing, image processing, speech processing, computer vision, and pattern recognition [1]. Natural language processing is an integral part of multimedia information retrieval. At present, information retrieval focuses on detecting and recognizing texts in videos, images, documents, and social media. After retrieval methods recognize text using an optical character recognizer, they use string matching algorithms to search for relevant words in the database. For digitized texts, such as annotated data of images or videos, the methods use string matching to define context for extracting relevant words at a high level from multimedia databases. In the methods in [2], string matching is used to index and retrieve information from multimedia databases at a high level. A string is a set of characters that can contain spaces and numbers. A string can be ordered or unordered because the main task of string matching is to find String *A* within String *B* regardless of the order of alphabets. The concept of string matching is illustrated in Figure 1, in which it finds all occurrences of a pattern *p* in a given text *t*.

Figure 1 shows the search for a substring pattern of defined length in a given string of defined length. This process involves a large amount of computation and is therefore time consuming. The best string matching algorithms should be selected out of *n* number of algorithms in literature according to application and complexity of the problems. Thus, we propose to survey string matching algorithms for investigating their strengths and weaknesses. This survey helps future researchers explore the strengths of different algorithms to overcome the drawbacks of existing algorithms.

Previous researchers have developed algorithms in different directions in dissimilar fields of various applications to fully utilize string matching algorithms. Szeto and Wong [4] used string matching algorithm to find patterns in musical databases. Srivastava *et al.* [5] proposed a framework

The associate editor coordinating the review of this manuscript and approving it for publication was Victor Hugo Albuquerque.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

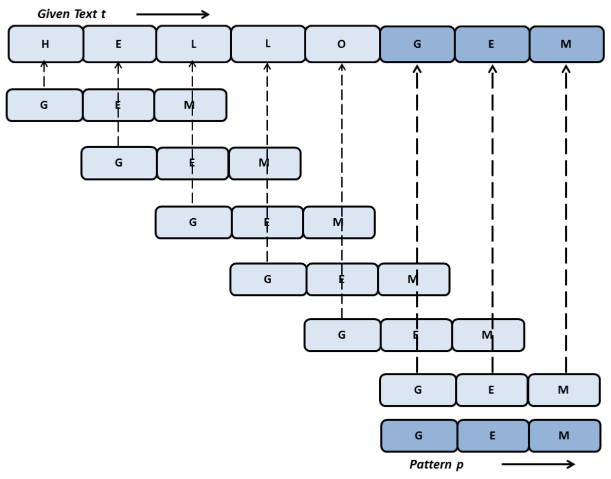


FIGURE 1. Definition of string matching [3].

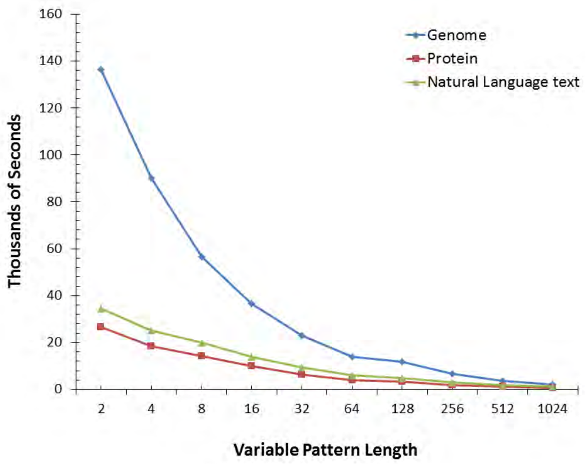


FIGURE 2. Performance analysis of backward oracle algorithm.

for searching multimedia contents, such as text, image, and audio. The authors used query-based algorithms that involve string matching approach to search texts. Kim *et al.* [6] noted that the demand for retrieval of multimedia contents, such as text, image, and audio, is increasing rapidly with the development of technology and suggested to develop new methods for facilitating searching and retrieval processes.

Motivation:

The main motivation of this survey is to overcome the limitations of previous survey articles related to string matching algorithms. Few works have summarized the algorithms used for string matching. Navarro [7] reviewed different approximate string matching algorithms in terms of pattern length and time complexity. Michailidis and Margaritis [8], [9] reviewed and experimentally examined online approximate and exact string matching algorithms. The review aimed to find the weaknesses and strengths of each algorithm. Charras and Lecroq [10] presented the methodologies related to working ideas of some exact string matching algorithms. Faro and Lecroq [11] experimentally evaluated exact string matching algorithms and classified them into four: character comparison, automata-based, bit-parallel, and constant space string matching approaches. Hendawi and Baharudin [3] surveyed and experimentally explored five popular string matching algorithms. Two parameters, namely, text and pattern sizes, were varied to check the efficiency of the algorithms in terms of execution time. Ahmed and Khare [12] presented a survey based on the applicability of string matching algorithms on hardware-based systems with prime focus on Knuth-Morris-Pratt (KMP), Aho-Corasick, and brute-force algorithms.

This backdrop reveals that existing surveys have focused on experimental evaluations to analyze time complexity of the algorithms. Whether the algorithms reviewed are based on single or multiple patterns is also unclear [13]. No any new taxonomy is presented, and no any future directions, challenges, and applications are discussed in detail. As a result, the future scope of string matching algorithms becomes uncertain. Choosing an appropriate algorithm for a particular application is a research challenge that has not been addressed yet by any of the existing studies. As shown in Figure 2, backward oracle algorithm obtains different processing times for dissimilar datasets or applications. The performance of the methods shown in the figure changes when the dataset and application change. Thus, all the above-mentioned factors have motivated us to carry out the proposed survey. The rest of the paper is organized as follows. Section 2 provides the general classification of string matching algorithms and proposes an extended classification of exact string matching. Section 3 explains the survey work in the area of software-based single-pattern matching algorithms along with an illustration of three fundamental algorithms of Boyer-Moore (BM), KMP, and Karp-Rabin. Section 4 describes the classification of single-pattern exact matching algorithms based on applicability. Section 5 contains a summary of single-pattern algorithms with their time complexity and limitations. Section 6 highlights latest trends and issues with open challenges related to string matching. Section 7 elaborates the conclusions.

II. CLASSIFICATION OF STRING MATCHING ALGORITHMS

As mentioned in the previous section, several methods that use string matching are available in different fields of literature. As a result, finding suitable methods for solving new issues and choosing appropriate methods depending on applications and requirements are difficult. Therefore, one of the primary objectives of this work is to provide critical analysis of basis or benchmark methods in terms of merits and demerits, that is, strengths and weakness of the methods. In this work, instead of focusing on n number of algorithms, we focus on the basis on which general string matching methods are developed. This way help readers find new directions, choose appropriate string matching methods, and

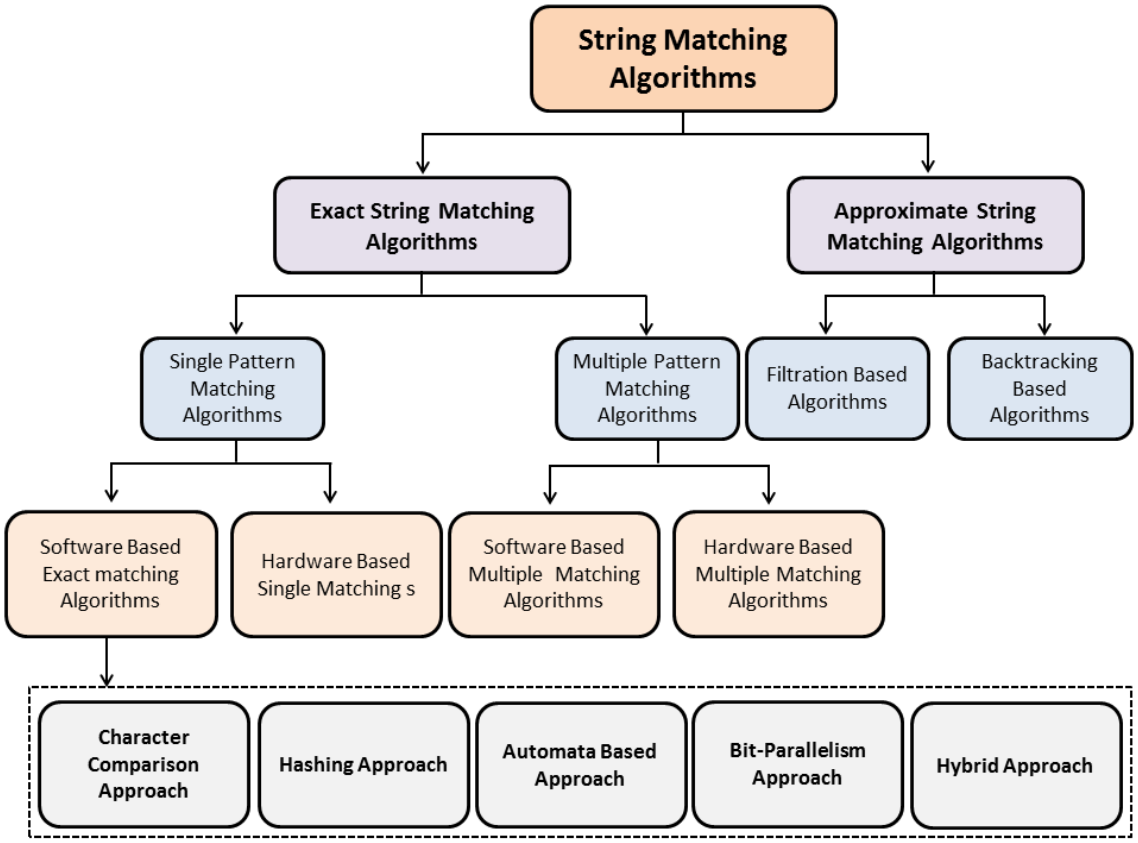

String Matching Algorithms
Exact String Matching Algorithms
Approximate String Matching Algorithms
Single Pattern Matching Algorithms
Multiple Pattern Matching Algorithms
Filtration Based Algorithms
Backtracking Based Algorithms
Software Based Exact matching Algorithms
Hardware Based Single Matching s
Software Based Multiple Matching Algorithms
Hardware Based Multiple Matching Algorithms
Character Comparison Approach
Hashing Approach
Automata Based Approach
Bit-Parallelism Approach
Hybrid Approach

FIGURE 3. New Taxonomy of Exact Matching Algorithms.

combine or integrate strengths of different methods to address new challenges. The analysis of different bases contributes to the community. We propose a new class according to concept and basis of the methods to save time in choosing an appropriate method depending on merits and demerits of the methods. For example, we categorize exact matching algorithms as single and multiple-pattern matching methods. The two categories are further classified as software and hardware-based methods. This new taxonomy helps readers understand the concept and suitability of the methods depending on their requirements.

The methods can be broadly classified into two main categories: exact string matching approach that does not allow any tolerance, and approximate string matching approach (also known as k-mismatch approach) that allows tolerance while matching. Exact string matching algorithms can be further divided into single- and multiple-pattern exact matching approaches, as shown in Figure. 3. Single-pattern exact matching can be grouped into software and hardware-based exact string matching algorithms. The software-based string matching algorithms can be divided into character comparison, hashing, bit-parallel, and hybrid approaches. However, this work focuses on software-based exact string matching algorithms rather than hardware-based exact string matching algorithms due to the vast scope of the latter, which goes beyond the scope of the proposed work. We discuss approaches that fall under software-based exact string matching algorithms in the following section.

A. APPROXIMATE STRING MATCHING ALGORITHMS

Approximate string matching algorithm finds a substring that is close to a given pattern string. This algorithm is contrary to exact string matching algorithm that expects a full match. In this case, deciding the degree of closeness is challenging but interesting, which depends on the application and complexity of the issues. According to Wu and Manber [14], this approach consists of finding all substrings S with K or fewer differences within given text t such that $d(p, S \;\; \geq \;\; K)$, where p denotes a short pattern string with length m, d denotes distance function, and K denotes an integer with value $K \;\; \geq \;\; 0$. In other words, the algorithms count the number matches and fix some threshold as K while matching substring with the strings. This approach is generally used when K mismatches are found between the pattern and the given text. Two popular distance measures, namely, Hamming distance measure and Levenshtein distance function, are used in this approach [15], [16]. These algorithms are introduced to address spell errors present in patterns or texts, low quality of texts, and difficulty in searching foreign names [17]. Approximate string matching algorithms can be classified as follows.

- **Filtration-based algorithms:** These algorithms are two-stage process algorithms. In the first stage, the locations of possible occurrence of patterns within the text are identified. In the second stage, all those locations are fully verified. Some of the recent algorithms that follow this approach include those in [18]–[20]. However, these algorithms are insufficient for worst case scenarios [21]. Most filtration indexes [19], [21], [25], [29], [35], [37] usually differ in terms of text sampling, pattern sampling, and alignment conditions [22].
- **Back Tracking-based algorithms:** These algorithms are generally an extension of exact string matching algorithms. In this approach, existing exact string matching algorithms are modified to enable approximate search using edit distance operations. The use of succinct (compressed data) and suffix index-based data structures is encouraged in these types of approaches. Some of the recent works include those in [23]–[26].

In many cases, approximate string matching does not work well, particularly in medical domain that expects 100% matching to find a solution without any approximation. In this context, exact string matching is more useful than approximate string matching.

B. EXACT STRING MATCHING

In exact string matching approach, all occurrences of a given pattern *p* from a given text *t* are found [27]. In this string matching, the characters present in a pattern window and a text window are compared. The length of both windows must be of equal length during the comparison phase. Shifting of characters in case of a mismatch is necessary to develop efficient algorithms [27]. As mentioned earlier, exact string matching algorithms can be classified as single- and multiple-pattern string matching algorithms.

1) SINGLE PATTERN MATCHING

In single-pattern matching algorithms, the algorithm receives only a single pattern as an input and searches for that specific pattern from the target database. This group can be further divided into two subgroups of hardware- and software-based matching. Some applications require more than one pattern to be searched, such as in analyzing mutations in DNA. Multiple-pattern matching algorithms are proposed for such applications [28].

2) MULTIPLE-PATTERN MATCHING

Multiple-pattern matching is an advanced version of single-pattern matching. In multiple-pattern matching algorithms, one input is received by the algorithm, and multiple occurrences of that input are searched from the target database [29]. Multiple-pattern matching algorithms are usually applied in the area of bioinformatics, such as in DNA comparison and protein sequence [30]. In DNA and protein sequences, these algorithms are used to detect and analyze any anomaly in the given sequence [31]. Similar to single-pattern string matching algorithms, multiple-pattern algorithms can be divided into hardware- and software-based string matching algorithms.

3) HARDWARE-BASED PATTERN MATCHING

The implementation of hardware-based matching algorithms requires hardware devices, such as graphical processing units (GPUs) and field programmable graphical arrays (FPGAs). These algorithms can be implemented using parallel processing programming languages, such as CUDA, Open-MP, and other specific languages. The implementation of string matching algorithms in hardware devices, such as GPU or FPGA, produces more overhead than that of software-based pattern string matching algorithms, but the former approach is faster than the latter approach. As mentioned above, hardware-based pattern matching needs different hardware devices and is thus costly. In addition, after the implementation of hardware-based pattern matching algorithm, it cannot be applied on different data or applications because changing the hardware design is impossible. By contrast, software-based pattern matching is flexible and can be used for any number of times on different applications. Therefore, software-based pattern string matching algorithms are popular [32], [63]–[65].

III. ANALYSIS OF SOFTWARE-BASED SINGLE-PATTERN MATCHING ALGORITHMS

In contrast to hardware-based string matching algorithms, software-based algorithms use certain compilers and programming languages for implementation purposes and require less overhead. As shown in Figure 3, software-based algorithms can be divided into character, hashing, suffix automata, bit-parallel, and hybrid string matching algorithms. The following section explains these algorithms in detail.

A. CHARACTER-BASED APPROACH

Character-based approach is known as a classical approach that compares characters to solve string matching problems. This approach can be further divided into six subgroups: brute force, BM, skip search, automata, Morris-Pratt, and factorization [33]. In brute-force approach, each character is compared from left to right individually at the cost of extra time requirement. This approach does not involve any preprocessing. Character-based approaches have two key stages: searching and shift phases. Previous studies have attempted to improve the processing of both phases. Among many character-based approaches, BM algorithm [34] is the baseline and is a standard and benchmark approach [27]. The key step in the BM algorithm is that shift table sends information about the number of characters that can be skipped to find a match when a mismatch occurs [35]. Figure 5 shows different versions of the BM algorithm, such as BM, turbo BM, tuned BM, Horspool, Apostolico-Giancarlo, quick search, reverse Colussi, optimal mismatch, and Raita [33].

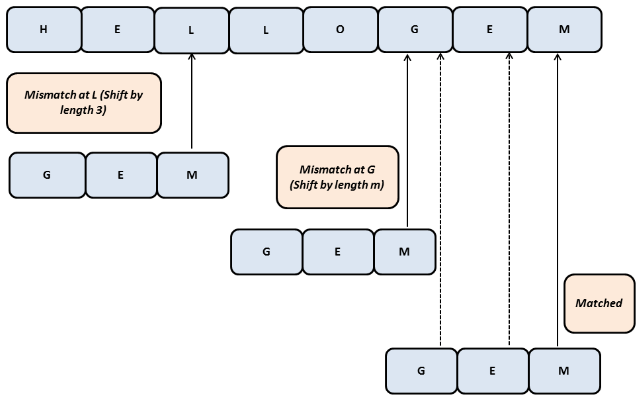

H
E
L
L
O
G
E
M
Mismatch at L (Shift by length 3)
G
E
M
Mismatch at G (Shift by length m)
G
E
M
Matched
G
E
M

FIGURE 4. **Working of Boyer Moore Algorithm.**

1) BM ALGORITHM [34]

BM algorithm starts searching characters from right to left of the given pattern. As shown in Figure 4, this algorithm can shift as many m characters as possible in case of a mismatch. BM algorithm involves two stages: pre-processing and searching for a given pattern from the right side of the window and using a bad-match table to skip characters in case of a mismatch.

- **Pre-processing:** During the pre-processing stage, a table is created to provide values regarding the amount of shift required in case of a mismatch. This table is also known as the bad-match table. Once a character mismatch occurs, the algorithm shifts to the right of the pattern in accordance with the value given in the bad-match table.
- **Searching for a given pattern:**
  Searching starts from the tail of the pattern (i.e., from right to left of the text) unlike in the naive algorithm in which the search starts from left to right. The algorithm works by computing the length of the search string and storing its value as default shift length. The time complexities of this algorithm are $O(n+m)$ and $O(n \times m)$ for the best and worst cases, respectively. Here, m denotes the length of pattern and n denotes the length of text to be searched. This process is suitable for a moderately sized alphabet with a long-length pattern [36]. However, not all characters within the text are compared [37]. In addition, the size of pattern and alphabet affects the pre-processing time [36]. To overcome these problems, extended BM approaches are proposed, as shown in Figure 5.

2) EXTENDED BOYER-MOORE APPROACHES

Researchers have optimized the BM algorithm on the basis of bad character rule, fast loop, or any other related parameter. The examples are discussed below. Horspool [38] proposed an algorithm called Horspool algorithm in 1980. This algorithm simplifies the BM algorithm by dropping the good suffix rule. The shift is computed in such a way that the rightmost character of the pattern gets aligned with the rightmost occurrence in the given text.

Apostolico and Giancarlo [39] proposed an improved version of BM approach [34]. In Apostolico-Giancarlo algorithm, character comparison is bounded by $3n/2$. The logic of the algorithm is the same as that of KMP, that is, it tracks the pattern that matched successfully. In the pre-processing stage, a priori knowledge regarding the structure of the pattern is used. This algorithm accesses each character twice at most. It does not work well for long-length patterns.

Boyer and Smith [8] proposed the Boyer-Moore-Smith (BMS) algorithm, which is an extension of BM algorithm and computes the shift with the text character. Given that the character next to the rightmost character provides a short drift in some cases, the approach works well. Maximum values among the two variables are taken.

Raita [40] proposed a modified form of BM algorithm called Raita algorithm, which takes advantage of strong dependencies that arise between successive characters. Dependencies can be extended up to 30 symbols. In this algorithm, the last character of the pattern is compared with the rightmost text character of the window for each attempt. If a match occurs, then the first character of the pattern is compared with the leftmost text character of the window or the middle character of the pattern is compared with the middle text character of the window.

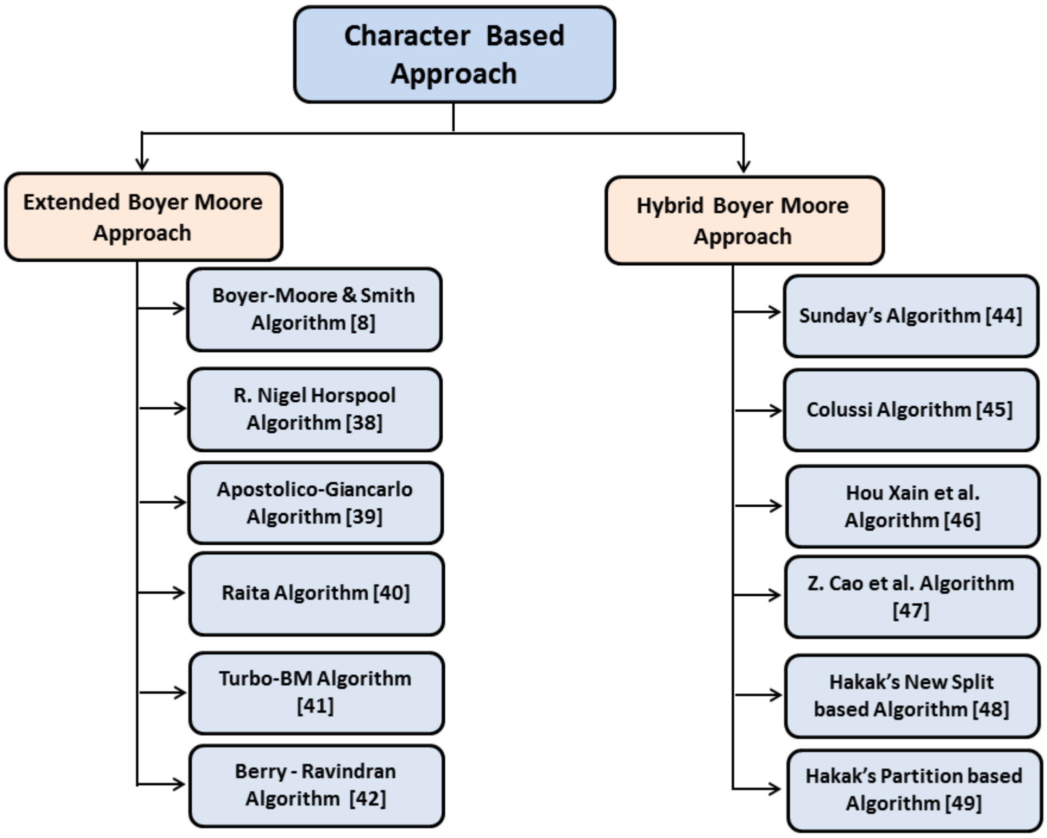
Crochemore *et al.* [41] proposed the turbo BM algorithm, which is a modified version of BM algorithm and is based on dynamic simulation technique. Turbo BM algorithm takes longer shifts than BM algorithm and scans the text segment until that segment is a suffix of the pattern. This algorithm remembers the suffix of the last matched substring of the pattern. Thus, this algorithm can jump over that substring and execute turbo-jump, which is a memory match.

Berry and Ravindran [42] proposed the Berry-Ravindran algorithm. This algorithm is an improvement of quick-search algorithm, which is also a modification of BM algorithm. Berry-Ravindran algorithm is based on bad character rule using character unrolling cycle. The shift is performed for two consecutive text characters that are immediate to the right of the window.

Ahmad [43] proposed the BBQ algorithm to improve search time by using two pointers in parallel approach. One pointer starts the search from the left side, and the other pointer starts from the right side. This way reduces the overall search time. The above-mentioned discussion indicates that different approaches based on BM algorithm are efficient in terms of processing time. However, flexibility must be added to ensure robustness of the BM algorithm. Therefore, hybrid BM algorithms are developed.

3) HYBRID BM APPROACHES

Hybrid BM approaches are combined or integrated by considering advantages of other methods to enhance the performance of each algorithm.



Character Based Approach
Extended Boyer Moore Approach
Hybrid Boyer Moore Approach
Boyer-Moore & Smith Algorithm [8]
R. Nigel Horspool Algorithm [38]
Apostolico-Giancarlo Algorithm [39]
Raita Algorithm [40]
Turbo-BM Algorithm [41]
Berry - Ravindran Algorithm [42]
Sunday's Algorithm [44]
Colussi Algorithm [45]
Hou Xain et al. Algorithm [46]
Z. Cao et al. Algorithm [47]
Hakak's New Split based Algorithm [48]
Hakak's Partition based Algorithm [49]

FIGURE 5. Classification of Character Based Approach.

Sunday [44] proposed an algorithm popularly known as Sunday algorithm, which is an improvement of BM algorithm. This method uses three key steps to scan the given pattern in three different orders. Sunday algorithm combines the logic of BM and KMP algorithms. In the first algorithm, a new function △1 is used to compute the index of the first leftmost occurrence of a character from the end of the text string. This way gives the absolute pattern shift required for shifting the pattern. This feature is also found in BM, but the shift in BM is relative to the position of *p* of the last mismatch. For scanning in a specific order, another function △2 is defined. The same feature is used in KMP and BM approaches. This function finds a position to shift from current mismatch position.

Colussi [45] proposed the Colussi algorithm to improve the efficiency of KMP algorithm. In Colussi algorithm, formal correctness proof of KMP algorithm is proposed by defining three assertions: Mch *(b)*, which asserts that ''the pattern matches the text in position *b*''; NMch *(b)*, which asserts that ''the pattern does not match position *b*''; and Eq *(b, i)*, which asserts that ''the first i characters of the pattern match the text starting at position *b*.'' On the basis of these assertions, the value of *''true''* null statement is searched using Hoarse axiomatic semantic proof rule. The results indicate some information is lost. This lost information is utilized to reduce the subsequent computational effort that is needed to attain the final result.

Xian-feng *et al.* [46] proposed the KMPBS algorithm by combining BM and KMP algorithms. The given pattern *P* of length *m* is searched from left to right within the text *T*. Searching is conducted by comparing the last character of *P* with the corresponding character of text *T*, and KMP algorithm is used to compare the rest of the characters in case of a match. Different automata approaches have been applied in character-based approaches, and a large amount of computational time is required by character-based algorithms. To reduce computational time, hashing-based algorithms are proposed and developed.

Cao *et al.* [47] proposed a character-based string matching algorithm, which calculates the statistical probability of each English letter in the pattern string in accordance with its special position in the pattern string. The proposed algorithm uses optimization based on evolution strategies to calculate the statistical probability and dynamic condition of each character in the pattern string. The main idea of the proposed algorithm is to search for a character with the lowest probability (also called lowlight character) among all the characters in the pattern string. After finding the lowlight character in the pattern string, the whole text is compared with the required lowlight character. If the lowlight character finds a match in the text, then the text is compared with target pattern string for the remaining characters. The proposed algorithm matches the target pattern string from the special position rather than from left or right ends. The proposed algorithm is compared with BM and KMP algorithms. However, the proposed algorithm requires more comparisons than BM algorithm.

Hakak *et al.* [48] proposed an algorithm based on pattern splitting. In this approach, the given pattern is divided into two parts. Only the second part of the pattern is searched using brute-force approach against the given text to enhance the search process. Once the second part of the pattern is found, the first part of the pattern is mapped directly on the basis of the location of the second part. The experimental results in the corpus of Arabic, English, Chinese, Italian, and French texts are promising. However, the time complexity of the algorithm may increase when searching for multiple patterns in a given text.

Hakak *et al.* [49] enhanced their previous work by splitting the given pattern. Similar to the previous work, the enhanced algorithm also works by splitting the given pattern into two parts. However, in this work, the split parts start the searching process. If the distance between the two parts is 0, then the pattern is found; otherwise, the searching process continues. Although the proposed approach is promising for non-unicode texts (e.g., Arabic), it may be time consuming for unicode texts because of the use of brute-force approach.

B. HASHING-BASED APPROACH

The previous section indicates that character-based approaches use characters to compare with the given string, whereas hashing-based approaches find hash values for the character to match rather than characters. The hashing-based approach saves a large amount of computation as it compares integer values instead of characters [27]. Karp and Rabin [50] used this approach for string matching problem. Karp-Rabin algorithm uses hashing value for matching process and conducts comparison from left to right. This approach is limited by hash collision, which occurs when two different strings map to the same number. To understand the mechanism of the hashing procedure, we present a basic version of hashing, that is, Karp-Rabin.

Karp-Rabin string matching algorithm is based on modular hashing and is the first to introduce the notion of hashing in string matching process. For a given pattern p with characters m, the hashing values are calculated first. Then, the hash values are used for matching between a given text t and a pattern p. After a hash function is used, the searching phase uses character comparison. Therefore, the algorithm involves two key steps: pre-processing and searching phases.

The pre-processing phase generally converts a string into a decimal number. That is, a string of c characters is converted into a string of d decimal numbers (Radix based). Hash for all pattern characters is computed up to $m - 1$, where

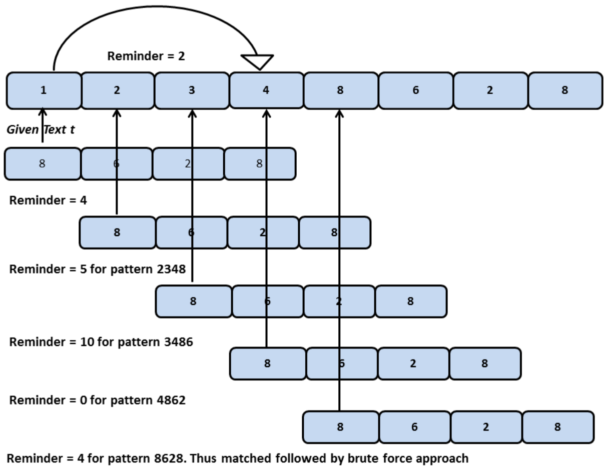

Reminder = 2
1 2 3 4 8 6 2 8
Given Text t
8 6 2 8
Reminder = 4
Reminder = 5 for pattern 2348
Reminder = 10 for pattern 3486
Reminder = 0 for pattern 4862
Reminder = 4 for pattern 8628. Thus matched followed by brute force approach

FIGURE 6. Working of Karp-Rabin Algorithm.

m is a pattern window that comprises m characters. Hash values are computed using Horners rule [51]. The pattern p is divided by pre-defined prime number q. Modulus operation is then used to calculate the remainder of pattern p with q. For each shift that ranges from shift ($s = 0 \ to \ n - m$), the remainders of pattern and text are compared for matching. Once a match is found, brute-force approach is implemented to verify the result of matching. Figure 6 shows an illustration to understand this concept. In this example, a given text is t and a pattern is p. The number of patterns to be searched is 8628, which is divided by a pre-fixed prime number q to obtain some specific remainder, that is, 4 in this example. Thus, during the searching phase, each new pattern window is divided by the same prime number to check whether the remainder of that window and p is the same. If the condition is positive, then brute-force approach is used to verify the condition further.

- Given Text $t = 12348628$
- Pattern $p = 8628$
- $q = 11$
- Remainder $(t) = 6$
- Remainder $(p) = 4$

However, the following issues may arise due to division operation.

1) If remainder $r1 = r2$, then the match is successful.
2) If remainder $r1 = r2$ but $r1 \neq r2$ after brute force, then it is spurious hit occurs.
3) If remainder $r1 \neq r2$, then the match is unsuccessful.

The estimated time complexity for pre-processing phase is $\Theta(m)$. For matching, the time complexities are $\Theta(m + n)$ and $\Theta(mn)$ for average and worst cases, respectively. Here, m denotes the pattern and n denotes the given text. However, the process of matching is relatively slow for long pattern shifting [52]–[54], and large prime number can cause overflow [54]. Hashing-based approaches can be classified further as q-gram and non q-gram approaches, as shown in Figure 7.

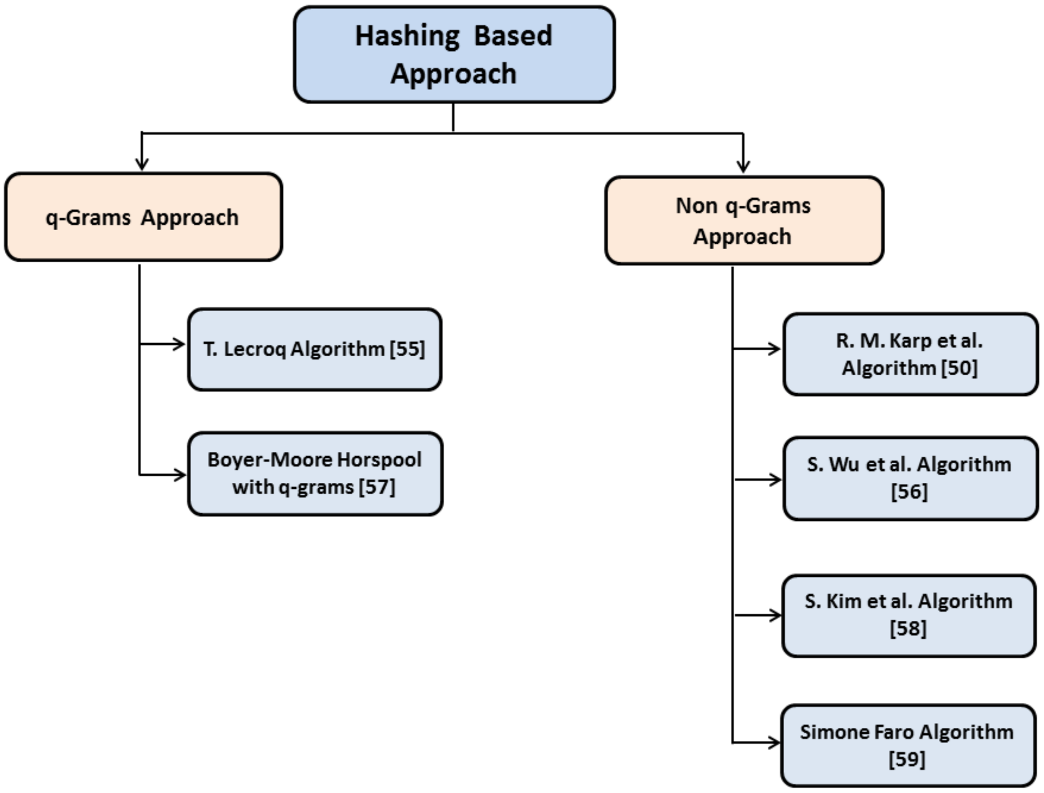

Hashing Based Approach
q-Grams Approach
Non q-Grams Approach
T. Lecroq Algorithm [55]
Boyer-Moore Horspool with q-grams [57]
R. M. Karp et al. Algorithm [50]
S. Wu et al. Algorithm [56]
S. Kim et al. Algorithm [58]
Simone Faro Algorithm [59]

FIGURE 7. Classification of Hashing-Based Approach.

1) Q-QRAMS APPROACH

The q-gram approach divides a given sequence into n subsequences for matching. On this basis, several approaches are developed as described below.

Lecroq [55] proposed a modification to the method of Wu and Manber [56]. The modification obtains a single matching pattern to accelerate the matching process. For the pattern of length l, the approach finds hash values using h function for each substring. In other words, using the window of string, $w = t[s + m - l..s + m - 1]$ with length l, the hash value is computed. If the value of w of the substring is greater than 0, then the shift of length $[h(w)]$ is applied; otherwise, the pattern is naively checked [57].

Boyer-Moore-Horspool with q-grams (BMHq) [57] is an efficient modification of Horspool algorithm and is suitable for DNA alphabet using q-gram approach. To inspect a single character at each alignment, q gram is read and an integer called *fingerprint* is computed. The idea consists of mapping ASCII codes of respective characters in DNA alphabet to a range of four characters, that is, $r : \{a, c, g, t\} \rightarrow \{0, 1, 2, 3\}$, such that the computation can be minimized or limited. The comparison for equality is performed by comparing the last q gram of pattern with corresponding q gram (in current window).

2) NON Q-GRAMS APPROACH

In non q-gram approach, the whole input pattern is encoded and scanned. A few approaches that use this concept are discussed as follows.

The algorithm of Wu and Manber [56] searches for all the occurrences of the patterns in a finite set $X = [x0, x1, \ldots, xk - 1]$ with a given text y and is based on BM algorithm. Substrings are considered to be of length q. The shift for all possible strings of length q is computed during the pre-processing phase. From X *finite* set, all substrings B of length q are hashed, followed by a shift. This step is followed by searching phase, which consists of reading substrings B of length q. Three tables are used in the pre-processing phase (i.e., SHIFT, HASH, and PREFIX).

The algorithm of S. Kim and Kim [58] follows the hashing approach and fully utilizes the encoding scheme. The input pattern is encoded, and the given text is scanned from left to right. S. Kim and Y. Kim claimed that the algorithm is efficient for large patterns and suitable for multiple-pattern strings.

Simone [59] proposed a condensed alphabet-based string matching algorithm. The proposed algorithm is an enhanced version of an existing skip-search string matching algorithm. The proposed algorithm involves two phases: pre-processing and searching. In the pre-processing phase for each substring, a numeric value called fingerprint is calculated and then stored in the table. Similarly, subsequences for each pattern are also stored in a table for searching purposes. In the searching phase, similar to skip search, a numeric value (fingerprint) for each position text is calculated and is matched with the stored fingerprint of the pattern. If the fingerprint matches, then the location is verified for possible matching. If the fingerprint match results in empty location, then no verification is required. The experimental results show that the proposed algorithm performs well when the length of the pattern is long. However, the performance of the proposed algorithm decreases when the length of the pattern is short.

In summary, although hashing-based approaches are faster than character-based approaches, hashing process still suffers from certain drawbacks (e.g., hashing collision) and is unsuitable for short-length patterns. Hashing approaches are also sensitive to capital and small case letters due to the use of different encoding schemes. For these applications, hashing algorithms may not perform well. For short-length patterns, automata approaches work considerably better than hashing processes according to our experiments.

C. SUFFIX AUTOMATA-BASED APPROACH

Suffix automaton/automation is an automaton that comprises two related but distinct automata constructors: deterministic acyclic finite state automaton (a data structure representing a finite set of strings) and suffix automaton (a finite automaton acting as suffix index) [60] for matching. It can be defined as $D(p) = \{Q, q_0, F, \Sigma, \delta\}$.

Here, $Q = \{q_1, q_2, q_3 \ldots . q_m\}$ is a set of states, $F = \{q_m\}$ is set to accept states, and $\delta = Q \times \Sigma \rightarrow Q$ is the transition function. This approach uses a directed acyclic graph in which nodes/vertices are called states, and edges between the nodes are considered a transition between the states. This approach uses the suffix automaton data structure that recognizes all the suffixes of the pattern. One of the states (node) denoted by $''q_0''$ is called the *"Initial state"* of the suffix automaton from where we can reach to all other states in the automaton. One or more of these states are marked as *"Terminal states"*. Thus, if we go from $''q_0''$ to any of these terminal states and note down the labels of the edges, then we obtain a suffix of the original string *"S"*. The following example in Figure 9 explains the concept of suffix automaton.

As shown in Figure 9, we have a pattern input "abbabb". Each state represents one character, and searching starts from state $''q_0''$ denoted by "0" in the example. The suffix automaton while traversing from state *0* to terminal nodes (denoted by double circles) must represent a suffix that is a substring of the main pattern, that is, "abbabb". In this case, the possible suffixes of "abbabb" are "b", "bb', "abb", "babb", and "bbabb". Figure 9 show that each terminal node results in the suffix that is a substring of the main pattern.

The final state is reached using the path given by the terminal states. This process reduces the large number of comparisons among patterns using the longest suffix. Therefore, time efficiency is guaranteed [61]. The approaches that use this concept are discussed below.

KMP string matching algorithm is a basic and fundamental algorithm that uses the concept of automata in string matching and was proposed by Knuth *et al.* [29]. The basic idea behind the algorithm is that the text *t* is scanned from left to right, and the algorithm decides the number of patterns *p* to be shifted to avoid redundant comparisons during a mismatch. Thus, this algorithm tracks information gained from previous comparisons. This algorithm skips characters depending on prefix and suffix rules and is illustrated using the example below.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 1$, no possible prefixes and suffixes are available for character "A". Thus, the value of prefix will be 0, and the pointer will be moved to index 2.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 2$, the possible prefixes and suffixes for character "C". C does not have a match. Thus, the value of prefix will be 0.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 3$, the possible prefixes for character "A" are A and AC. The possible suffixes are A and CA. A is prefix and suffix. Thus, the value of prefix will be 1.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 4$ the possible prefixes for character"C" are A, AC, and ACA. The possible suffixes are C, AC, and CAC. AC is prefix and suffix. Thus, the value of prefix will be 2.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 5$, the possible prefixes for character "A" are A, AC, ACA, and ACAC. The possible suffixes are A, CA, and CACA. ACA is prefix and suffix. Thus, the value of prefix will be 3.

I	1	2	3	4	5	6
Pattern	A	C	A	C	A	G
Prefix	0	0	1	2	3	0

When $i = 6$, the possible prefixes for character "G" are A, AC, ACA, ACAC, and ACACA. The possible suffixes are G, AG, CAG, ACAG, CACA, and ACACAG.

The above-mentioned process of matching shows that this procedure does not find the element with suffix and prefix. This is the novel idea of automata-based string matching,

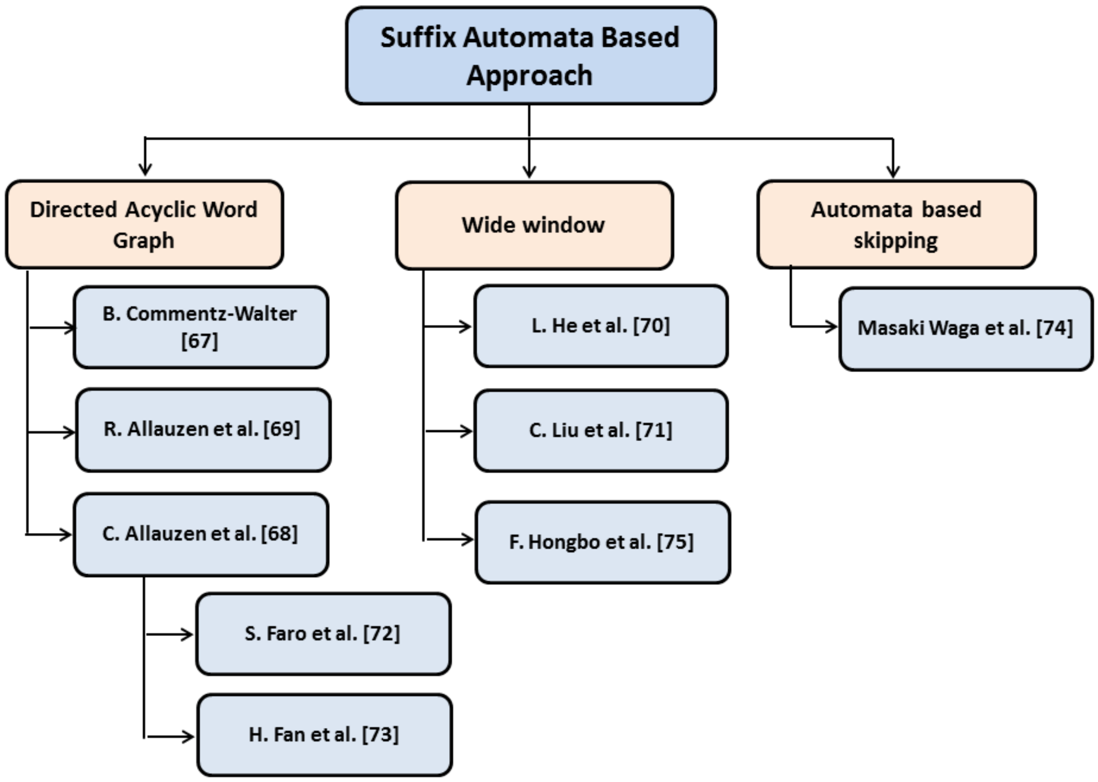


FIGURE 8. Classification of Suffix Automata Based Approach.

which is different from other existing methods. Given that a pre-generated prefix table is used, the procedure allows skipping certain comparisons during matching. The entire process requires search time complexity $O(n)$, and the pre-processing phase requires $O(m)$ space. However, this approach is inefficient for a small set of data [62]. We discuss the algorithms shown in Figure 8 that use this basis for string matching as follows.

1) DIRECTED ACYCLIC WORD GRAPH-BASED APPROACHES

A directed acyclic word graph (DAWG) approach is a data structure that allows fast word searches. In DAWG, each node represents a character. The first character represents the entry point. One can travel from one node to another to find a proper match.

Backward non-deterministic DAWG (BNDM) matching [67] is based on the concept of non-deterministic automaton approach along with bit-parallel concept. A window of length m is shifted over a given text t. For each alignment, a pattern p is searched by scanning the current window backward while automaton configuration is updated accordingly.

Double-forward DAWG matching [69] uses two automata. The key idea is to divide window into two parts, and each window is scanned with a factor automaton of p. The two positions for each text window are represented by α and β, and the algorithm starts at position β and reads forward the text in the current window for each attempt.

Backward oracle matching (BOM) algorithm [68] is based on acyclic automaton and recognizes at least the factors of p with $m + 1$ states. The key idea is that, if back searching fails on any letter (e.g., character c) after reading a particular word w, then cw is not a factor of pattern p and the window can be moved after c. An intermediate structure, which is called factor oracle, is built. Oracle is an automaton to ensure that Q has exactly $m + 1$ states. This intermediate structure called factor oracle must satisfy four conditions: 1) automaton should be acyclic, 2) states should be as few as possible, 3) factors of p should be recognized at least, and 4) linear number of transitions should be used. Once a window of size m is moved on text, a pattern is searched by scanning the current window backward to realize a secure shift.

BOM-based algorithms have different variants, such as extended BOM, forward BOM, and simplified extended BOM. Extended BOM [72] is a modification of BOM algorithm by enhancing the speed of the algorithm through introducing a fast loop. The transitions are computed in one step for two rightmost characters to determine an undefined transition with high probability. Forward BOM is an improved version of extended BOM and combines the ideas of quick-search and extended BOM algorithms. The idea of this algorithm is to compute the shift advancement while focusing on a character that follows the forward character (current window).

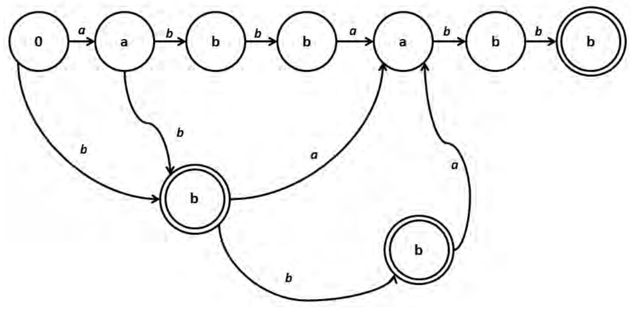


FIGURE 9. **Working of Suffix automaton.**

Simplified extended BOM algorithm [73] replaces the two-dimensional table with one-dimensional array and has the same procedure as extended BOM. This algorithm saves memory compared with its predecessor.

2) WIDE WINDOW BASED APPROACHES

He *et al.* [70] proposed the wide window (WW) algorithm, which is different from the traditional window system used in string matching. WW algorithm divides the text into n/m overlapping windows (of size 2 $m - 1$). This algorithm also uses suffix automaton approach. The suffix of text is scanned from middle to right using forward suffix automaton. Corresponding prefixes are scanned backward using reverse prefix automaton if required. m rightmost characters are scanned from left to right with the initial state q_0 until a full match or lack of transition is achieved. The remaining $m - 1$ leftmost characters are scanned from right to left.

Liu *et al.* [71] improved the WW algorithm proposed by He *et al.* [70] by changing the parameters of the older version. In the improved version, the second phase of WW algorithm is modified by changing the length of the longest remembered prefix to more than 0 for the first improvement and $m/2$ for the second one.

Hongbo *et al.* [75] proposed multi-window and integer comparison based on three suffix string matching algorithms. The proposed algorithms include the enhanced version of three existing suffix string matching algorithms, namely, quick search, tuned BM, and BMHq, by adding the functionality of unaligned read integer comparison and multi-window. The main objective of enhancement is to reduce the comparisons (integer comparison) and accelerate the matching process (multi-window). In multi-window (i.e., jump distance calculation mechanism), the text is divided equally into areas, and two windows belong to one single area. The matching process starts in each area from both ends and continues toward the center of the area and stops when two windows overlap each other. The functionalities of integer comparison and multi-window (continuous jump) are added to the three existing algorithms (e.g., quick search, tuned BM, and BMHq) for fast and cost-effective string matching.

3) AUTOMATA-BASED SKIPPING APPROACHES

Masaki *et al.* [74] proposed an automata-based skipping algorithm for fast and efficient real-time pattern matching in embedded online applications. The proposed algorithm is a modified version of an existing string matching algorithm called Franek-Jennings-Smyth (FJS). The above-mentioned authors extend the skipping value functionality for timed pattern matching from FJS string matching algorithm using automata states by language, substring, and word over-approximation. Two versions of the FJS-type algorithm (e.g., untimed and timed) are presented for offline and online pattern matching. The FJS skipping value functionality is the combination of two skip value algorithms, namely, ∇ (quick search) and β (KMP). For online versions of the FJS-type algorithm, zone abstraction is combined with two value skipping functionalities. The skipping value algorithms help in unnecessary matching executions, and the proposed FJS-type algorithm uses part of the pattern instead of the whole target word for fast online pattern matching.

D. BIT-PARALLEL APPROACH

Automata-based string matching algorithms are excellent for long-length patterns but are unsuitable for short-length ones. Thus, bit-parallel approach, which involves parallel processing, was proposed by Domolki in 1968 to accelerate the matching process. This concept is based on parallel computing. In this approach, the number of operations within algorithm is decreased to the number of bits in computer word [11]. This algorithm is fast and efficient, especially when the length of the given pattern p is less than the word length [27]. We classify bit-parallel algorithms depending on bit-level operations: Shift-OR (SO), Shift-AND, and Single Instruction/Multiple Data (SIMD)-based instruction approaches. The classification is presented in Figure 10.

The approaches that use bit operation are discussed as follows. Shift-based algorithms use logical bitwise operations such as NOT, OR, AND, and XOR to compare strings. Bitwise NOT performs logical negation for each input bit. Bitwise OR performs bitwise OR operation, that is, it compares two given inputs of equal length and outputs 0 in case both input bits comprise 0; otherwise, it outputs 1. Bitwise AND performs logical multiplication of two bit patterns and outputs 1 in case both input bits comprise 1s; otherwise, it outputs 0. Similarly, logical bitwise XOR operation performs exclusive OR operations and outputs 1 if two bit patterns are different; it outputs 0 in case bit patterns are equal. The following algorithms implement bitwise operations for string matching.

SO algorithm [76] uses bitwise operation for string matching unlike other approaches. The key idea is to perform the parallel operation of NFA while searching. Here, NFA represents a vector of m different states, and each state i indicates the state of the search between the positions of the pattern and the positions of the text. The basic methodology follows the concept of KMP and BM as discussed above. Non-active states are represented by 1 and active states by 0.

Fredriksson [77] proposed an algorithm based on SO algorithm [76] by improving average and worst case

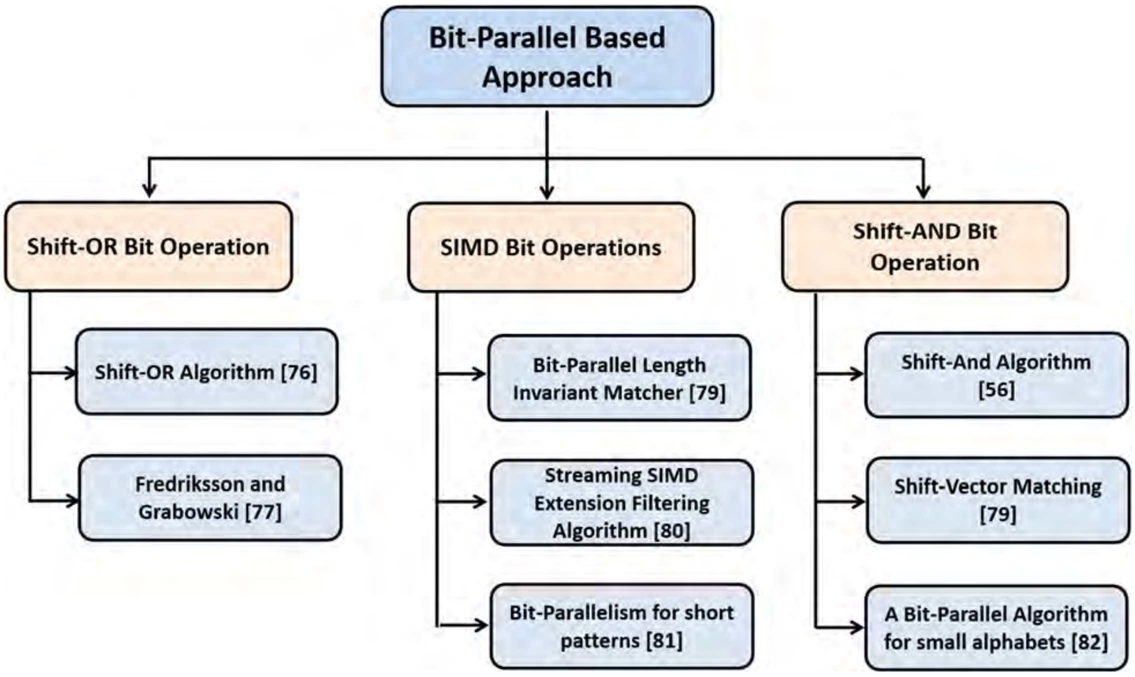

Bit-Parallel Based Approach
Shift-OR Bit Operation
SIMD Bit Operations
Shift-AND Bit Operation
Shift-OR Algorithm [76]
Fredriksson and Grabowski [77]
Bit-Parallel Length Invariant Matcher [79]
Streaming SIMD Extension Filtering Algorithm [80]
Bit-Parallelism for short patterns [81]
Shift-And Algorithm [56]
Shift-Vector Matching [79]
A Bit-Parallel Algorithm for small alphabets [82]

FIGURE 10. **Classification of Bit-Parallel Approaches.**

running times. The number of mismatches between pattern and text is computed using bit parallel.

Popular algorithms [14], [76] normally take $n[m/w]$ time to find the occurrences of pattern P in text T (w denotes the number of bits in machine word). Bit-parallel approach is extended by using super-alphabets. The idea is to process several characters using single step. The set of patterns is processed in the same way as that in SO algorithm, but the algorithm scans only q^{th} factor of the text. q^{th} factor is the set of new patterns generated from original pattern P, which is calculated during the pre-processing phase. The authors claimed that using this technique has accelerated the speed by a factor of $O(log\, n)$. However, the length of the input pattern is dependent on the computer word size (i.e., the number of bits that can be processed by the CPU). This method fails when pattern lengths as input are more than the computer word size.

SIMD-based algorithms are developed to accelerate string matching using hardware approach by utilizing the functionality of SIMD instructions. As in shift-based algorithms, logical operations are done in a parallel-wise manner to accelerate matching. However, in SIMD-based approach, the capability of core pre-processor is utilized to also enhance searching. For example, INTEL SSE4.2 instruction set can perform 256 comparison operations in a single instruction. The four main string processing operations are listed as follows:

(i) PCMPESTRI - Packed compare explicit-length strings, return index in ECX/RCX

(ii) PCMPESTRM - Packed compare explicit-length strings, return mask in XMM0

(iii) PCMPISTRI - Packed compare implicit-length strings, return index in ECX/RCX and

(iv) PCMPISTRM - Packed compare implicit-length strings, return mask in XMM0 [78].

All these instructions can be utilized to enhance string matching using different programming models. The algorithms that utilize SIMD instructions are discussed as follows.

Peltola [79] proposed the bit-parallel length-invariant matcher. In this approach, an alignment matrix is constructed consisting of ω rows (size of the word in target machine) such that, for each row_i, $(0 \leq i \leq \omega)$ contains a pattern that is right shifted by i characters. The algorithm operates by sliding the alignment matrix over the text and checks for any possible placement of input text. The system must allow hardware implementations to implement SIMD in real time for practical applications.

Ulekci [80] proposed the streaming SIMD extension filtering algorithm. The SIMD instruction is a feature of microprocessors that supports parallel processing on multiple datasets. Two phases, namely, filtering and verification, are used in this algorithm. In the case of filtering, the most observable probable pattern with fast heuristic is detected within portions of text. The output of filtering is verified in the verification step. This process requires worst time complexity of $0(nm)$, best time complexity of $O(n/m)$, and average time complexity of $O(n/m) + n.m/2^{16}$, where m represents the length of text pattern and n represents the length of given text.

Ulekci [81] concluded that this algorithm is suitable for long-length patterns. Two 16-byte blocks, namely, $N = [n/16]$ and $M = [m/16]$, are created. Considering that this algorithm focuses on long-length patterns, the lowest limit for m is set to $32 (32 \leq m)$. This operation cannot be generalized for different types of inputs because these approaches are

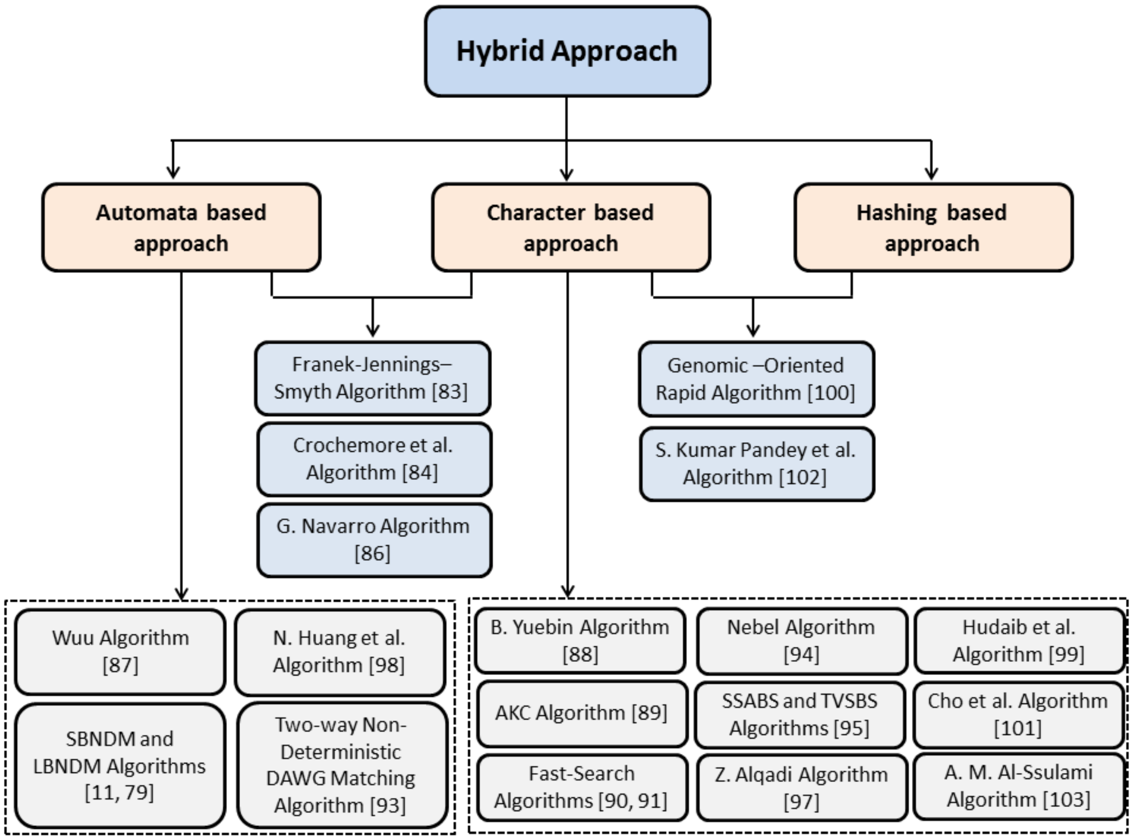

Hybrid Approach
Automata based approach
Character based approach
Hashing based approach
Franek-Jennings–Smyth Algorithm [83]
Crochemore et al. Algorithm [84]
G. Navarro Algorithm [86]
Genomic –Oriented Rapid Algorithm [100]
S. Kumar Pandey et al. Algorithm [102]
Wuu Algorithm [87]
N. Huang et al. Algorithm [98]
SBNDM and LBNDM Algorithms [11, 79]
Two-way Non-Deterministic DAWG Matching Algorithm [93]
B. Yuebin Algorithm [88]
Nebel Algorithm [94]
Hudaib et al. Algorithm [99]
AKC Algorithm [89]
SSABS and TVSBS Algorithms [95]
Cho et al. Algorithm [101]
Fast-Search Algorithms [90, 91]
Z. Alqadi Algorithm [97]
A. M. Al-Ssulami Algorithm [103]

FIGURE 11. Classification of Hybrid Approaches.

required to convert non-deterministic automata to deterministic automata.

Wu and Manber [56] proposed the Shift-AND algorithm, which does not convert NFA to deterministic finite automaton (DFA) but uses NFA directly for performing operations in parallel. For NFA, the automaton is given by $\{Q, \Sigma, \delta, q_0, F\}$ for the language P in which all words are recognized having ending occurrence of p. Here, $Q = \{q_1, q_2, q_3 \ldots . q_m\}$, where q_0 is an initial state, $F = (q_m)$ is set to final states, and $\delta : Q \times \Sigma \rightarrow P(Q)$ is the transition function. The key idea is to keep a record of all prefixes that matched the suffix of the text read by creating a table that holds bit mask. In a bit mask, the set of prefixes is kept and updated using bit parallel. Thus, the algorithm builds a table and updates bit mask in scanning the pattern.

Bit-parallel algorithms usually do not keep a record of previous alignments that have been checked. Shift-Vector matching [79] is the first algorithm to introduce partial memory for transferring information to subsequent alignments. A bit vector S is maintained, which provides information regarding the occurrence of the pattern at certain positions [69]. In the searching phase, the algorithm takes OR operation with bit vector and updates Shift-Vector corresponding to text character, which is aligned to the rightmost character of the pattern.

The bit-parallel algorithm for small alphabets [82] is based on the principle of matching matrix of the pattern and the text. For matrix matching, a 2-base logarithm table is used to locate the leftmost ''1'' bit. This bit indicates the recent probable occurrence of the pattern in the text. The bad character approach of BM algorithm and 2-base logarithm table value of the current flag are used to obtain shifts.

A single algorithm may not work well for different applications because each application may pose different challenges. Therefore, the strengths of different algorithms should be determined to integrate their advantages in a hybrid approach for solving complex issues.

E. HYBRID APPROACH

The hybrid approach operates well on a complex problem because it combines the advantages of different algorithms and is better than individual algorithms [83]. Many approaches use the hybrid concept. The approaches that combine one or more character-based methods are placed under character-based approaches, and methods that use one or more methods from automata-based approaches are placed under automata-based approaches. The approaches that use character- and automata-based methods are placed in between the two approaches, as shown in Figure 11.

Crochemore *et al.*'s algorithm [84] is a combination of Aho-Corasick [85] and DAWG [41] and is also known as the reverse factor algorithm. This algorithm makes at least 2n comparisons. Two theoretical tools, namely, Aho-Corasick machine and DAWG, are used for implementation purposes. Two processes, namely, PROCESS1 and PROCESS2, are used in this algorithm. PROCESS1 scans the text from left to right with a shift of ending position of the pattern by $m/2$ (m denotes the length of the shortest pattern). PROCESS1 remembers the position of each character i in the string (Υ) and passes control to PROCESS2, which starts searching the string backward from $i + SHIFT(SHIFT = length\,of \;\; shortest\,pattern - |\Upsilon|)$. The feature of this algorithm is that it can make long jumps compared with the Aho-Corasick algorithm [85].

Navarro's algorithm [86] is a modification of BDM algorithm and skips characters using suffix automaton. The modification allows errors in searching the pattern in reverse order.

Wuu's algorithm [87] is an extension of KMP algorithm and uses tree-based approach for pattern matching. The main difference from KMP algorithm is that the shift is moved horizontally right or left similar to KMP and vertically up and down. The traversal process is bounded for each node within a subject tree. The time complexity of the algorithm is $O(n \times log\, n)$, where n denotes the number of nodes in the tree.

Yuebin's algorithm [88] is a modification of Boyer-Moore-Horspool algorithm [38]. The pattern is scanned from right to left. In the pre-processing phase, an array NEXT is generated to compute the shift position. The information in this array is used to determine the number of characters to be skipped. AKC algorithm [11], [89] is a modification of the algorithms in [39]. Characters within windows are scanned from right to left. For each search, the information regarding the factors that match the suffix of the pattern is stored. Once pattern is shifted after each search, this algorithm ensures that the previously matched suffix and text factor remain the same.

Fast-search (FS) algorithms [11], [90], [91] constitute the family of variant algorithms from BM. The basic mechanism of all these algorithms is nearly the same as that of BM, that is, shift is computed using bad character rule only during comparison mismatch in the first attempt; otherwise, good suffix rule is applied for other cases. FS is the first algorithm of this family. The comparison of pattern with the window is performed from right to left. At each attempt, the pattern and current window is compared, and shift is computed using bad character rule in case a mismatch occurs during the first character comparison. In other cases, the suffix rule is used. Backward FS is another algorithm of this family. It combines bad character rule with good suffix rule to obtain backward good suffix rule. Forward FS algorithm uses look-ahead character to compute large shift advancements.

Simplified BNDM (SBNDM) and long BNDM (LBNDM) [11], [79] are proposed by the same authors and are based on BNDM algorithm [92]. The main loop in SBNDM is made faster than that in BNDM without memorizing the longest prefix. This process makes its shift computation lighter than that of BNDM. If current alignment position is A in the text with u denoting updates are done, then the starting position for next aligment will be $A + m - u + 1$. The functionality of LBNDM to handle long-length patterns is improved compared with that of BNDM. In LBNDM, pattern is partitioned into subpatterns. The leftmost subpattern is scanned first. Then, all the remaining subpatterns are examined when a match is found.

Two-way non-deterministic DAWG matching (TNDM) algorithm is also a variant of the BNDM algorithm. Backward and forward searches are made alternately. Once the pattern is aligned with text window and a mismatch occurs, TNDM initializes state vector D in accordance with the two rightmost characters compared with 1^m in BNDM and scans in the forward direction to examine text characters after the alignment to look for any conflicting characters within the pattern. Further improvement is made on TNDM in the form of forward non-deterministic DAWG matching. When finding the suffix using FNDM, BNDM backward check is substituted with a naive check of occurrence [11], [27], [93].

Nebel [94] modified Horspool string matching algorithm by increasing the searching speed using probabilities of different symbols. This algorithm is similar to Sunday's algorithm [43] except in the way symbols in the pattern are compared with symbols in text. The relative number of occurrences of different symbols is used to represent probabilities. Pre-processing is divided into two phases. In the first phase, the positions of pattern P in which same symbols occur are determined. In the second phase, the values of V (V denotes an array) are set using min-heap depending on needs. The algorithm takes $O(mlog\,(n))$ overall running time.

SSABS and TVSBS algorithms [95] are examples of a hybrid algorithm. SSABS is hybrid of quick-search and Raita algorithms [39]. The comparison is carried out using Raita algorithm. First, the rightmost character is compared. Then, a further comparison is carried out using the leftmost character once a match is found. In this way, the resemblance is established between the given window and pattern. The remaining characters are compared from right to left until a complete match occurs, or vice versa [95]. Thathoo *et al.* [96] later improved this algorithm using a shift method from Berry-Ravindran bad character rule [42]. This algorithm requires a small number of character comparisons due to its great shift advancements.

FJS algorithm [83] is a mixture of linear worst case time complexity and sublinear average behavior of KMP and quick-search algorithms, respectively [26]. The algorithm has two phases. The first phase involves the use of two steps for each attempt of comparison. The first comparison between the given window and pattern is performed using the quick-search algorithm and starts from the rightmost character of the pattern. In the case of a mismatch, quick-search shift is implemented; otherwise, FJS invokes the second step. In the second phase, KMP pattern matching is used, which starts from the leftmost character, and shift is performed accordingly followed by a return to first step [11], [27].

Alqadi *et al.*'s algorithm [97] is a multiple skip pattern matching and is a modification of BM algorithm. To the best of our knowledge, this algorithm performs skips depending on index values. In this case, the comparison is based on the index values of all substring occurrences of given text that is equal to pattern p of length m. The skip value is calculated using ranges from $1-(m-1)$.

Huang *et al.*'s algorithm [98] aims to reduce the memory requirements of Aho-Corasick algorithm [85]. The approach of magic states is used in DFA. The algorithm rearranges states in two steps: magic states are found in the first step, and the transition matrix is partitioned depending on a threshold in the second step. The magic state receives the same input character resulting in the same next state. The transition matrix has two submatrices: one with smaller state values than the threshold, and the other is used to generate bitmap matrix and state list matrix. In the search phase, all elements in the second matrix are identified and bitmap matrix is set to 1 in case no magic state is found; otherwise, it is set to 0. Next state is inserted to the state list matrix.

Hudaib *et al.*'s algorithm [99] is a modification of Berry *et al.*'s algorithm [42]. The algorithm divides the text into two equal parts and scans two parts by using two windows simultaneously. The left window scans from the left part of the text, whereas the right window scans from right to left in parallel. This process makes this algorithm suitable for parallel processors [27]. This algorithm uses shift rule of Berry *et al.*'s algorithm to compute shift values during the searching phase.

Genomic-oriented rapid algorithm [100] uses Horspool algorithm and filtering approach based on hash function. In the pre-processing phase, the position of each character c of the alphabet within the pattern is stored with the rightmost character preceded by character c. In the searching phase, Horspool bad character rule concept is implemented; in the concept, fast-based loop is used to locate the occurrence of the rightmost character of the pattern [27].

Cho *et al.* [101] used the bad character rule of Horspool along with factorial number system to compute the shift table for text search and found all the substrings with same relative orders as pattern p within a given text t. The proposed algorithm is more efficient than KMP with time complexity of $O(n+mlogm)$ in average case and $O(n+mlogm)$ in worst case.

Shivendra Kumar *et al.* [102] proposed a hashing with chaining-based hybrid string matching algorithm to reduce the time complexity of string matching algorithms. The idea of hashing with chaining is combined in the proposed algorithm. The proposed algorithm involves two phases: preprocessing and searching phases. In the first phase of preprocessing, the given string is divided into substring, and each substring has a size equal to the pattern. After division, each substring is assigned with a unique integer (ASCII value), and the substrings are stored in a hash table along with their location using a hash function. In the second phase of searching, the integer hash value of the pattern is calculated, and the hash value of the pattern and integer hash values of substrings are compared in the hash table. If the hash values of pattern and substring are matched, then the location of the substring is returned. The proposed algorithm cannot reduce the time complexity in most of the cases and requires $O(n-m)$ extra memory because it stores substrings in a hash table.

Al-Ssulami [103] proposed a hybrid algorithm for string matching called simple string matching. The proposed algorithm is a modified version of Horspool algorithm with additional string matching conditions for scanning and matching the text (from left to right) and string pattern (from right to left). The proposed algorithm operates in two steps. In the first step of pre-processing, the pivoting character in the pattern is searched by computing the character distance and its maximum safe shift. In the second step, the algorithm compares the pivot character of the pattern with characters of the text. If the pivoting character matches with the character in the text, then the algorithm starts matching the pattern with text from the right most character until the end of the text. If the matching fails, then hybrid Horspool shift is used for matching. If the pattern and text with equal character *(bi)* length is mismatched $(where\, 0 <= i < k$ and $b_j \neq b_j + 1)$ at any given positions $(0 \leq k < m\, or\, j)$ during the matching process, then the position of the pattern must be shifted to exactly $(k-i+1)$ or $(k-j+1)$ positions to accelerate the searching process. The proposed algorithm achieves good performance for pattern matching on human proteins, text of natural languages (e.g., Arabic, English, Italian, French, and Chinese), and E. coli genome. However, the efficiency of the proposed algorithm decreases when the mismatches of the patterns do not occur at the rightmost end.

In summary, we discuss and analyze different algorithms of software-based pattern string matching in the past two decades. The discussion and analysis indicate that each algorithm has its own merits and demerits in terms of suitability to applications and capability to handle complexity of problems. We summarize the above-mentioned discussion with different parameters in the following section.

IV. COMPARISON ANALYSIS OF SOFTWARE-BASED PATTERN STRING MATCHING ALGORITHMS

The literature review reveals that time complexity, limitation, and dataset are important in deciding the performance of the methods. We analyze the methods in terms of the three parameters and show the results in Tables 1–5.

On the basis of the analysis reported in Tables 1–5, we summarize the advantages and disadvantages of the methods from 1980 to 2018 in Table 6. This summary helps readers understand the strengths and weaknesses of the methods. Accordingly, a hybrid method or a unified method with advantages of the methods can be developed to address new challenges. Appropriate methods can also be easily selected depending on strengths and requirements.

TABLE 1. **Comparative analysis of character based string matching algorithms.**

Algorithms	Time complexity	Limitations	Data
Horspool [38]	Pre-processing:$O(m+\sigma)$ for time complexity and $O(\sigma)$ for space complexity Searching: $O(m \times n)$ for time complexity	This algorithm is good for short-length patterns but not for long patterns such as full sentences [109].	Not specified
Apostolico-Giancarlo [39]	Pre-processing : $O(m+\sigma)$ Searching: $O(n)$	The performance of this method depends on the pre-processing step [104].	English text
Sunday [44]	Pre-processing: $O(m+\sigma)$ Searching: $O(m \times n)$	This algorithm is good for short-length patterns but not for long-length patterns including sentences [10].	English text
BMS [34]	Pre-processing: $O(m+\sigma)$ Searching: $O(m \times n)$	The searching of this algorithm misses characters in some cases. [77]	English and DNA alphabets
Colussi [45]	Pre-processing: $O(m)$ Searching:$O(n)$	This method is time consuming [110].	Random sequences of English text
Raita [40]	Pre-processing: $O(m+\sigma)$ Searching: $O(m \times n)$	This algorithm is inefficient for complex datasets [110].	English text
Turbo-BM [41]	Pre-processing: $O(m+\sigma)$ Searching: $O(n)$	Given that this algorithm considers small length patterns for searching, it consumes considerable time for large datasets such as DNA sequences [101].	Random English text
Berry-Ravindran [42]	Pre-processing: $O(m+\sigma)$ Searching: $O(m \times n)$	This method is inefficient for complex datasets [111].	English text
Cao et al. [47]	the matching time is $\theta(n/\lambda m)$ when $\lambda m \geq 1$	this algorithm requires more comparisons then BM	English alphabets in NL texts
Hakak et al. [48]	The given input is divided into two halves, and the right half is proposed first to improves the memory and time complexity.	The time complexity of this algorithm may increase in searching for multiple patterns in a given text.	Arabic, English, Chinese, Italian, and French texts
Hakak et al. [49]	Time complexity = $O(n)$	This algorithm maybe time consuming for uni-code based texts because of the use of brute force approach.	Arabic text.

TABLE 2. **Comparative analysis of automata-based string matching algorithms.**

Algorithms	Time complexity	Limitations	Data
Forward DAWG	Searching: $O(n)$ for worst case	This method is complex to implement and understand [113].	Small alphabets of English text
BNDM [67]	$O(n[m/\omega])$ for time complexity, and $O(\sigma[m/\omega])$ for space complexity	This method has poor performance on natural texts. [76]	English texts
BOM [68]	Pre-processing: $O(m)$ Searching: $O(m \times n)$	This method is unsuitable for long length patterns [110].	Random texts of 10 MB size
Double forward DAWG matching [69]	$O(m + nlog\|\Sigma\|m/m)$, where n is the size of the text and m is the size of the searched word.The condition is linear in the worst case $(O(m+n))$.	This method is limited to a specific dataset.	Not specified
Linear DAWG matching	$(O(nlogm/m))$ for average case, and $(O(n))$ for text characters in worst case.	This method is time consuming for large datasets [109].	English text (from the TREC Wall Street Journal collection) and network traffic (from the 1999 DARPA Intrusion Detection Evaluation Dataset)
Liu et al. [71]	$O(n)$ for worst case, and $O(n/m)$ for the best case.	The performance of this method depends on the length of patterns [71].	English texts 10MB
Forward BOM	$(O(m \times \Sigma)$ time is consumed to construct the forward factor oracle of a word. Pre-processing: $(O(m \times \sigma^2)$ for time and space.	this algorithm involves a costly pre-processing step.	DNA sequences composed of four nucleotides adenine, cytosine, guanine and thymine, protein sequence, natural English text
Extended BOM [72]	$(O(\sigma^2))$ time and space are consumed to construct a two-dimensional table.	This method is good for short length patterns.	Rand problem and genome sequences
Simplified extended BOM [73] (fan)	This algorithm is 12% faster than the original extended BOM on average.	Given that this involves 2-grams algorithms, has difficulty for long pattern searching.	DNA sequences composed of four nucleotides adenine, cytosine, guanine and thymine, protein sequence, natural English text
Waga et al. [74]	This method has no linear time complexity in worst case.	Skipping value algorithms help in unnecessary matching executions and fast online pattern matching.	Online Montre.
Fan et al.	NA	The additions of integer comparison and multi-window functionality result in fast and cost effective string matching for short length patterns.	DNA sequence and English text and NL.

As shown in Table 6, hybrid approaches are more flexible than a single approach and can be extended to solve any string matching problem. However, hybrid approaches require comprehensive knowledge of different algorithms and need to combine their features to obtain the optimal output. Prior to designing an algorithm, other factors

TABLE 3. Comparative analysis of bit parallelism string matching algorithms.

Algorithms	Time complexity	Limitations	Data
Shift-And [56]	$O(n[m\omega])$ for worst case and $O(\sigma[m/\omega])$ for extra space	This method is time consuming for natural texts [11].	Random English texts
SO [76]	Pre-processing: $O(m \times \sigma)$ Searching: $O(n)$	this method is good for a specific dataset [11].	English text of 50,000 characters (a legal document)
Bit-parallel length invariant matcher [79]	$O(n[m/\omega](\omega + m - 1))$ for worst case time complexity and $O(\sigma \times (\omega + m - 1))$ for extra space	This method is limited to few data. [81].	Natural language, DNA sequence, and binary alphabet random text.
Fredriksson and Grabowski [77]	the pattern representation of optimal average and worst case running times is fitted in a single computer word $O(n)$. Total average time: $O(nlog_\sigma m/m)$	This algorithm is good for long-length patterns [55].	DNA sequence, natural text, binary input
Streaming SIMD extension filtering [80]	$O(m \times \sigma)$ for worst case and $O(n/m + nm/2^{16})$ for average case	This method is good for DNA sequence searching but not for general texts [104].	Genome sequence
Shift-Vector matching [79]	$O(n[m \times \omega])$ for worst case and $O(\sigma[m/\omega])$ for extra space.	This algorithm works well when searching finds a complete match [79].	DNA and natural English text
Bit-parallel Algorithm for small alphabets [82]	Pre-processing: $O(m\sigma)$, Searching: $O(nm)$	The performance of this method degrades when the length of the patterns varies [11].	DNA sequence
Bit-parallel for short-length patterns [81]	$O(n[m/\omega] + [n/m]log_2(\omega))$ for worst case	The performance of this method degrades when the length of the patterns varies [11].	Rand problem

TABLE 4. Comparative analysis of Hashing based string matching algorithms.

Algorithms	Time complexity	Limitations	Data
Wu and Manber [56]	Pre-processing $O(n + m \times T(h))$, where T(h) is the time required to compute the hash value, space complexity for storing shift table: $O(n)$, searching: $O(n(T(h) + m))$	This algorithm is sensitive to the length of patterns [56].	Text files from Wall Street Journal
S. Kim & Y. Kim [58]	Time complexity is not mentioned.	This method is good for constant-length patterns.	English text and DNA sequence
Lecroq [55]	The pre-processing phase of this algorithm compute a shift for all the possible strings of length q	This method is good for short-length patterns.	English text and genome sequence
BMHq [57]	$O(nm)$ for worst case and $O(m + \sigma)$ for space	This algorithm works well when the same patterns are exactly repeated.	DNA sequence
Faro [59]	$O(n/(m-q))$ for best case and $O(nm)$ for worst case	The performance of this algorithm is poor when the length of the pattern is short	Genome sequences, protein sequences and natural language texts.

that can affect its performance must be determined and addressed.

V. CURRENT TRENDS, APPLICATIONS, CHALLENGES AND FUTURE SCOPE

Analysis of the performance of string matching algorithms until 2018 shows that they are popular and required in several applications. Most of the algorithms are developed to improve time efficiency because matching in string matching algorithms involves complex operations. Thus, new algorithms or extensions of string matching concepts are limited to few applications and solving issues. However, the use of string matching algorithms has increased because they work well regardless of databases, scripts, and applications. In this section, we explore the current trend of string matching algorithms, new applications that require improved strengths of these algorithms, new challenges, and the possible future scope.

A. CURRENT TREND

We use a Web of Science citation report (from 1991 to 2018) to determine the current trend of string matching algorithms, as shown in Figure 12. The trend is increasing because the scope of string matching is extended to big data analysis, parallel computing, and distributed network [109]. The graph is plotted by using various keywords, such as the application of string matching algorithms, exact matching algorithms, and other related keywords. The citation report is created on the basis of a Web of Science tool. Our intention is to show that string matching is still vital for new concepts, such as big data and social media data. Figure 12 shows the trend or importance of string matching over time. String matching plays an important role, and an appropriate method that can be modified or used for new applications and datasets should be selected.

B. APPLICATIONS

As discussed earlier, the scope of string matching algorithms is limited to few applications. Based on the literature review, we investigate the applications where exact matching algorithms can be applied for deeper analysis. Figure 13 shows the applications, namely, multimedia, networking, forensic, and search engines. From this figure, additional applications or sub-applications can be identified in the future.

TABLE 5. **Comparative analysis of hybrid string matching algorithms.**

Algorithms	Time complexity	Limitations	Data
Crochemore et al. [84]	$O((n/m).logm)$ is consumed for an average number of comparisons.	This method is sensitive to the length of patterns.	English text
BNDM for long-length patterns	$O(nm)$ for worst case and $O(nlogm/m)$ for average case.	This method works well for a particular length of patterns.	Not specified
Wuu [87]	The expected running time is less than the size of the text $O(n \times logn)$	This method is good for long-length patterns.	Text files with pattern ranging from 5 to 15.
Yuebin [88]	This method Performs $O(n/m)$ comparisons and has worst case complexity of $O(n)$	This method is efficient for limited lengths of patterns.	Different text strings
AKC [89]	Pre-processing: $O(m + \sigma)$, Searching: $O(nm)$	The method is limited to a particular dataset.	Not specified
SBNDM and LBNDM [11,79]	$O(nm)$ and $O(\sigma)$ for extra space when $O(m \leq \omega)$	The performance of these algorithms depends on the length of patterns.	English text, DNA sequence and binary data.
Fast search [90, 91]	$O(m.max(m, \sigma))$ is consumed to construct tables for backward and forward good suffix rules with table size $O(m.\sigma)$	The performance of this method is poor for short-length patterns [72].	Natural English text and Rand problem
TNDM [93]	Pre-processing: $O(m)$ for time and space complexity, Searching: $O(n[m/\omega])$ for time complexity and $O(\sigma[m/\omega])$ for extra space.	It is expensive for longer patterns [27].	English text and genome sequence
Horspool with probabilities [94]	Pre-processing: $O(mlogm)$ for time and $O(m)$ for space.	This method is limited to specific datasets.	DNA sequence
SSABS and TVSBS [95]	$O(nm)$ for worst case and $O(\sigma)$ and $O(\sigma^2)$ spaces for computing shift tables.	This method is good for DNA data [106].	Genome sequence
FJS [83]	Pre-processing: $O(m)$ for computing failure function and $O(m+\sigma)$ for computing quick search bad character, Searching: $O(n)$	This method is sensitive to the length of patterns [27].	Texts published by Project Gutenberg and DNA sequence
Alqadi et al. [97]	$O(m \times n)$ but moves to $O(n)$ due to skips.	This method is good for uniform patterns.	DNA sequence
Huang et al. [98]	This method guarantees linear time complexity in worst case situation.	This method is designed for network applications.	2389 patterns and Defcon9 traces as input strings.
Hudaib et al. [99]	Pre-processing: $O(m)$ for time and space Searching: $O(nm^2)$ for worst case complexity and $O(m)$ for additional space	The performance of this method depends on the location of text patterns [100].	Calgary corpus of text
Genomic oriented rapid [100]	$O(nm)$ for worst case and $O(n + \sigma)$ for space	This method is sensitive to the length of patterns [104].	Genome sequence
Cho [101]	$O(n + mlonm)$ for average case and $O(n + mlonm)$ for worst case	This algorithm is good for long-length patterns [101].	Random English text
Pandey et al. [102]	Pre-processin: $O(m \times (n - m) + m)$, Searching O(1) for best case and $O((n - m) + 1)$ for worst case	Given that the substrings are stored in a hash table, this algorithm requires $O(n - m)$ extra memory.	Not specified.
Simple string matching	Pre-processing $O(m)$ for space	The efficiency of this algorithm decreases if mismatches of patterns do not occur at the rightmost end.	Arabic, English, Italian, French and Chinese texts.

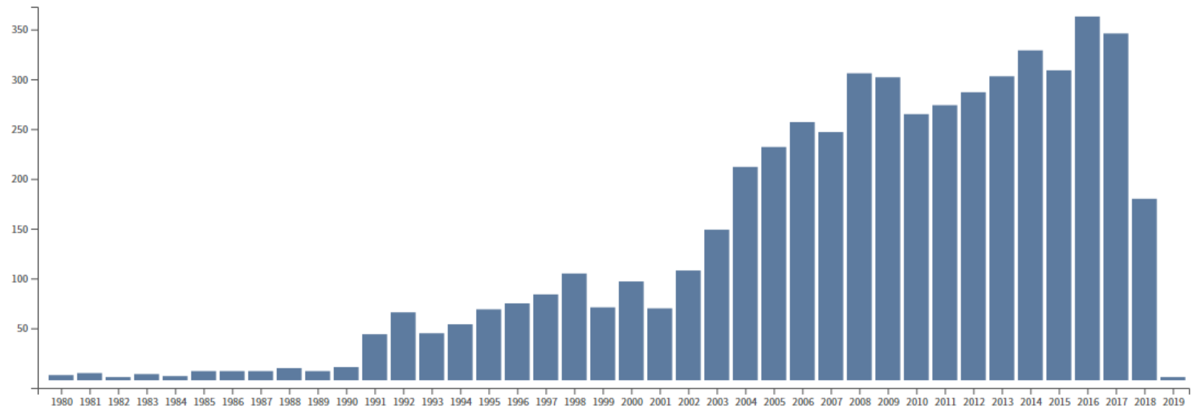


FIGURE 12. **Web of Science Citation Report [110].**

1) MULTIMEDIA APPLICATIONS

Social media use multimedia information, such as texts, videos, audios, images, and different scripts. Our analysis reveals that most of the string matching algorithms focus on English text, biological data but not different script data, video data, and image data. The reason is that most of these algorithms involve exact or approximate matching. This matching is insufficient to handle multivalued and multivariate data. Therefore, string matching algorithms should be extended to the above-mentioned applications

TABLE 6. Summary of string matching approaches.

Approaches	Advantages	Limitations	Applications
Character based	Long length of texts requires long shifts. thus, these methods are suitable for applications that search long pattern of texts	The sliding window system of these methods is slower than that of other string matching approaches.	Natural text
Automata-based	These approaches avoids backtracking (examining each character only once under constant time per character). Thus, these methods are suitable for web related applications or real-time systems.	These methods consume much time in construction, such as the construction of flow chart and computation of transition table. Building automaton is also time consuming if input symbol is large.	Security related applications and real-time applications with optimal input size.
Bit parallel based	Bit-parallel algorithms are extremely fast when a pattern fits in a computer word.	The performance of these algorithms degrades considerably as m number of bits per word sizeâŇĽ increases.	These methods are suitable for applications in which parallelism can easily be performed, such as plagiarism checking, spell checking, data mining and bio-informatics.
Hashing based	The speed of processing is high for these methods due to quadratic comparisons. However, they are suitable for pre-processed texts and hash tables with large entries.	The cost of a good hash function can be significantly higher than the inner loop of the lookup algorithm for a sequential list or search tree. Tese approaches are ineffective when the number of entries is small and can cause long delays due to micro-processor cache misses induced by poor quality of reference.	These apporaches are suitable for texts pre-processed with large table entries such as spell checking, imaging applications and bio-informatics. These methods can be extended to multi-dimensional pattern matching.
Hybrid based	The combination of more than two approaches results in improved solutions.	Most hybrid methods have overheads.	The suitability of these methods depends on the application.

because they require complex matching procedure [111], [112], [117].

2) NETWORKING APPLICATIONS

The rapid advancements in technology have made security a primary concern in all networks at present. The threats of hacking and intruder attacks constantly exist. To solve security issues, several methods have directly and indirectly proposed encoding-decoding and encryption-decryption to secure data. For example, most prominent methods include encryption, virtual private networks, and firewalls. Among these techniques, network intrusion detection (NID) is a new technique that is used to detect suspicious activities at the network and host level. NID systems are used to capture data packets that travel through network media, such as cables and wireless. In these cases, string matching algorithms can be investigated to verify the packets. In other words, signature-based intrusion detection and anomaly detection systems can be introduced using string matching algorithms.

3) FORENSIC SCIENCE

String matching algorithms are explored for network security and multimedia applications and can be extended to forensic science applications that require matching to authenticate data with the original data, such as blood and DNA samples for large data. In the future, DNA sequence can be used as a passport to identify persons in the centralized database of the world.

4) SEARCH ENGINES

Indexing and retrieval methods may fail to retrieve actual information based on user interest through recognition of text, sound, image, or video. String matching plays a vital role in solving such issues because it does not require recognition of each character in the text or content in image or video. Instead, string matching considers the entire input as one pattern to find a match. This advantage is inherent to string matching algorithms.

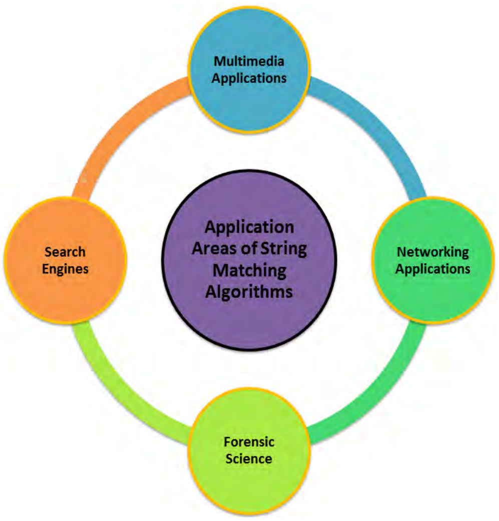


FIGURE 13. The possible new applications of string matching algorithms.

C. CHALLENGES

From the analysis of the literature review, it can be observed that there are numerous challenges within the domain of Exact matching algorithms that need to be addressed. We list the challenges as follows:

1) REFINEMENT OF PROPOSED CLASSIFICATION

The performance of string matching algorithms varies in different areas of applications, such as molecular biology, network intrusion detection, and text processing. Some algorithms perform well for only short-length patterns, only long-length patterns, and only average-length patterns. A string matching algorithm that performs well for English text may behave differently in DNA matching. Thus, the performance of string matching algorithms must be analyzed on the basis of different patterns and text formats. Analyzing the performance of these algorithms on the basis of different applications (e.g., DNA sequencing, fingerprint detection, and text processing) and classifying them depending on performance in respective areas rather than methodologies used are difficult. However, these tasks help researchers implement and optimize only specific algorithms pertaining to specified area. For example, only algorithms that perform well in DNA sequencing can be targeted for optimization rather than selecting algorithms randomly.

2) PERFORMANCE ANALYSIS USING DIFFERENT ENCODING TECHNIQUES

Encoding is the basis for string matching algorithms. Encoding techniques have different types, such as ASCII, UTF-8, and UTF-16. The performance of different string matching algorithms can be checked using different encoding techniques. This way is useful for texts that require more than 1 bit for a single alphabet, such as Arabic, Chinese, and Persian texts.

3) ANALYSIS TOOL

Considerable simulation, networking, and programming tools are available, which aid in determining the behavior of a particular model or application. However, no effective tool exists that can aid in determining the performance of algorithms. Although many mathematical models are available, not everybody can efficiently determine the performance of algorithms through those models. Those models depend on the hypothesis. The main parameter for determining the performance of the algorithm is execution time. However, other factors, such as hardware and text size to be searched, play a crucial role in determining the performance of algorithms. The importance of these factors for an algorithm should be determined.

4) EXECUTION TIME

Considerable overheads are associated with string matching algorithms, such as pre-processing time followed by searching time. The time and space complexities of these algorithms increase with the increase in data size. Thus, efforts are required to optimize these algorithms for fast execution with minimum overhead.

5) LIBRARY

No authenticated library is available for string matching algorithms to determine their efficiency. Although few libraries, such as Faro and Lecroq [11], are used as a smart tool, considerable efforts are needed to include many and existing algorithms. Accordingly, future researchers can optimize the previous algorithms without considerable efforts by keeping time constraint in mind.

6) DEVELOPMENT OF EFFICIENT DATA STRUCTURES

Different data structures are used in string matching algorithms. Some data structures use trees and arrays based on suffix or prefix approach. Efficient data structures should be developed, which can perform better than previously used data structures regardless of applications and data.

7) OPTIMAL FILE SIZE

All algorithms do not perform well depending on different text and pattern sizes. The performance of some algorithms either linearly or exponentially decreases. The optimal file size or number of words should be determined to enable an algorithm to provide an efficient performance. This task is challenging due to a large number of available algorithms to be implemented and evaluated.

8) BENCHMARK STANDARD

Different developers implement string matching algorithms depending on their logic. If two same algorithms are compared for a given data set, then both algorithms will behave differently depending on the different platforms used and logics used. In this case, some benchmark methods are needed to help researchers determine whether the algorithm developed by the developer is working in accordance with the standard and is the correct version. This way helps developers identify the correct version of algorithms with ease of implementation.

D. FUTURE SCOPE

Here, we discuss the possible future scope of software-based pattern string matching algorithms.

1) FACTORIAL ANALYSIS OF STRING MATCHING ALGORITHMS

The effect of different factors, such as text size, RAM, and IDE, can be determined on string matching algorithms by designing a factorial model based on factorial design. This procedure helps determine the effect of different factors on the execution time of an algorithm.

2) IMPLEMENTATION OF STRING MATCHING ALGORITHMS IN MAPREDUCE ENVIRONMENT

MapReduce is a parallel programming paradigm used in Hadoop for big data analysis. Current string matching algorithms can be optimized for MapReduce framework. This process increases the execution time of these algorithms using parallel processing. The trend has already started. Abdulrazzaq *et al.* [113] developed a new algorithm for DNS sequence matching by using an MPI technique. The proposed approach uses multicore processors for parallel processing. The obtained DNA sequence shows the highest performance among those of other serial versions of algorithms after parallelization [113].

3) USE OF GPU AND FIELD PROGRAMMABLE GATE ARRAY (FPGA)

CPU is optimized for sequential serial processing with few cores. GPU comprises thousands of small, efficient cores that can handle multiple tasks simultaneously [114], [115]. GPU is programmed using either CUDA or OpenGL. String matching algorithms are usually written in C or C + + programming languages. Thus, another future direction is to implement string matching algorithms in GPUs and FPGAs using CUDA or OpenGL or any other GPU/FPGA-supported language. This way increases the efficiency of string matching algorithms by manifolds.

4) SURVEY OF APPROXIMATE AND HARDWARE-BASED APPROACHES

Another future work is to carry out a survey based on approximate and exact matching algorithms implemented in hardware devices. The proposed work will focus on evaluating their weaknesses and strengths on the basis of a specific parameter.

5) ARABIC PATTERN MATCHING ALGORITHMS

Arabic is the second most spoken language in the world after English [116], [117], [118]. In Arabic language, connected and unconnected words exist, which take considerable bytes and processing time. Thus, development of multilingual exact matching algorithms with suitable encoding techniques is a promising and interesting future work.

6) MEMORY ANALYSIS OF STRING MATCHING ALGORITHMS

Existing string matching algorithms have been analyzed in terms of time complexities and performance [66]. Analysis of memory requirements of existing string matching algorithms on heap during runtime is an interesting topic. Optimization based on memory consumption on heap during runtime can be a future research area as well.

7) OTHER FUTURE RELATED WORKS

Other possible future works related to string matching include classification of multiple pattern-based algorithms depending on an application with performance analysis and survey on image-based matching algorithms with a focus on time complexities of algorithms and possible areas of application. Determining whether a particular algorithm can be used in multiple applications is also an interesting endeavor. The need of proposing new algorithms even if the present hardware can solve time complexity issues should be supported as well.

VI. CONCLUSION

The field of string matching is vast due to the development of numerous algorithms, and studying methodology, complexity, and limitations of all those algorithms is a tedious task. This work focuses only on software-based pattern string matching algorithms and their applications. However, other category of string matching algorithms can also be explored. In this work, we analyze more than 50 string matching methods in terms of strengths, weaknesses, and efficiency with respect to applications, which can help future researchers identify suitable string matching algorithms depending on their application and complexity of the problems. On the basis of the analysis, we compare string matching algorithms of respective category based on time complexity, limitation, and databases. Furthermore, we identify new challenges, applications, and directions to expand the scope of string matching algorithms. To the best of our knowledge, this comprehensive survey on single-pattern exact string matching algorithms is the first to discuss future directions, challenges, possible applications, and new taxonomies. We extend the review on approximate string matching algorithms in detail by considering implementation issues, real-time application, and future vector-based matching algorithms.

REFERENCES

[1] A. A. AbdulRazzaq, N. A. Rashid, A. A. Hasan, and M. A. Abu-Hashem, "The exact string matching algorithms efficiency review," presented at the 3rd World Conf. Innov. Comput. Sci., 2013. [Online]. Available: http://www.world-education-center.org/index.php/P-ITCS/article/view/2668/2228

[2] S. Hakak, A. Kamsin, P. Shivakumara, M. Y. I. Idris, and G. A. Gilkar, "A new split based searching for exact pattern matching for natural texts," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0200912.

[3] K. M. Alhendawi and A. S. Baharudin, "String matching algoritms (SMAs): Survey & Empirical analysis," *J. Comput. Sci. Manage.*, vol. 2, no. 5, pp. 2637–2644, May 2013.

[4] W. M. Szeto and M. H. Wong, "Stream segregation algorithm for pattern matching in polyphonic music databases," *Multimedia Tools Appl.*, vol. 30, no. 1, pp. 109–127, Jul. 2006.

[5] S. Shrivastav, S. Kumar, and K. Kumar, "Towards an ontology based framework for searching multimedia contents on the Web," *Multimedia Tools Appl.*, vol. 76, no. 18, pp. 18657–18686, Sep. 2017.

[6] Y.-H. Kim, H.-J. Kwon, J.-G. Kang, and H. Chang, "The study on content based multimedia data retrieval system," *Multimedia Tools Appl.*, vol. 57, no. 2, pp. 393–405, Mar. 2012.

[7] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, Mar. 2001.

[8] P. D. Smith, "Experiments with a very fast substring search algorithm," *Softw., Pract. Exper.*, vol. 21, no. 10, pp. 1065–1074, Oct. 1991.

[9] P. D. Michailidis and K. G. Margaritis, "On-line string matching algorithms: Survey and experimental results," *Int. J. Comput. Math.*, vol. 76, no. 4, pp. 411–434, Mar. 2001.

[10] C. Charras, and T. Lecroq, *Handbook of Exact String Matching Algorithms*. London, U.K.: H. King's College, 2004.

[11] S. Faro and T. Lecroq, "The exact online string matching problem: A review of the most recent results," *ACM Comput. Surv.*, vol. 45, no. 2, Feb. 2013, Art. no. 13.

[12] G. F. Ahmed and N. Khare, "Hardware based string matching algorithms: A survey," *Int. J. Comput. Appl.*, vol. 88, no. 11, pp. 16–19, Jan. 2014.

[13] L. Otero-Cerdeira, F. J. Rodríguez-Martinez, and A. Gómez-Rodríguez, "Ontology matching: A literature review," *Expert Syst. Appl.*, vol. 42, no. 2, pp. 949–971, Feb. 2015.

[14] S. Wu and U. Manber, "Fast text searching: Allowing errors," *Commun. ACM*, vol. 35, no. 10, pp. 83–91, Oct. 1992.

[15] D.Sankoff, *Common Subsequences and Monotone Subsequences*. Reading, MA, USA: Addison-Wesley, 1983.

[16] V. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," in *Problems of Information Transmission*. Cham, Switzerland: Springer, 1965, p. 196.

[17] V. SaiKrishna, A. Rasool, and N. Khare, "String matching and its applications in diversified fields," *Int. J. Comput. Sci. Issues*, vol. 9, no. 1, pp. 219–226, 2012.

[18] M. Farach-Colton, G. M. Landau, S. C. Sahinalp, and D. Tsur, "Optimal spaced seeds for faster approximate string matching," in *Proc. Int. Colloq. Automata, Lang., Program. (ICALP)*, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds. Berlin, Germany: Springer, Jul. 2005, pp. 1251–1262.

[19] J. Kärkkäinen and J. C. Na, "Faster filters for approximate string matching," in *Proc. 9th Workshop Algorithm Eng. Exp. (ALENEX)*, 2007, pp. 84–90.

[20] G. Kucherov, L. Noe, and M. Roytberg, "Multiseed lossless filtration," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 2, no. 1, pp. 51–61, Jan. 2005.

[21] G. Kucherov, K. Salikhov, and D. Tsur, "Approximate string matching using a bidirectional index," in *Theoretical Computer Science*, vol. 638. Amsterdam, The Netherlands: Elsevier, 2016, pp. 145–158.

[22] G. Navarro and R. Baeza-Yates, "A hybrid indexing method for approximate string matching," *J. Discrete Algorithms*, vol. 1, no. 1, pp. 205–239, 2001.

[23] D. Belazzougui, F. Cunial, J. Karkkainen, and V. Makinen, "Versatile succinct representations of the bidirectional burrows-wheeler transform," in *Algorithms—ESA*, in *Proc. Eur. Symp. Algorithms*, H. L. Bodlaender, G. F. Italiano, Eds. Berlin, Germany: Springer, Sep. 2013, pp. 133–144.

[24] T. W. Lam, R. Li, A. Tam, S. Wong, E. Wu, and S. M. Yiu, "High throughput short read alignment via Bi-directional BWT," in *Proc. IEEE Int. Conf.Bioinf. Biomed.*, Nov. 2009, pp. 31–36.

[25] L. M. Russo, G. Navarro, A. Oliveira, and P. Morales, "Approximate string matching with compressed indexes," *Algorithms*, vol. 2, no. 3, p. 1105, Sep. 2009.

[26] T. Schnattinger, E. Ohlebusch, and S. Gog, "Bidirectional search in a string with wavelet trees and bidirectional matching statistics," *Inf. Comput.*, vol. 213, pp. 13–22, Apr. 2012.

[27] A. A. AbdulRazzaq, N. A. Rashid, A. A. Hasan, and M. A. Abu-Hashem, "The exact string matching algorithms efficiency review," *Global J. Technol.*, vol. 4, no. 2, pp. 576–589, 2013.

[28] V. Alfred, "Algorithms for finding patterns in strings," in *Algorithms and Complexity*, vol. 1. Amsterdam, The Netherlands: Elsevier, 2014.

[29] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, Jul. 1977.

[30] D. Suleiman, "Enhanced berry ravindran pattern matching algorithm (EBR)," *Life Sci. J.*, vol. 11, no. 7, pp. 395–402, 2014.

[31] D. Suleiman, A. Hudaib, A. Al-Anani, R. Al-Khalid, and M. Itriq, "ERS-A algorithm for pattern matching," *Middle East J. Sci. Res.*, vol. 15, no. 7, pp. 1067–1075, 2013.

[32] S. Arudchutha, T. Nishanthy, and R. G. Ragel, "String matching with multicore CPUs: Performing better with the Aho-Corasick algorithm," in *Proc. IEEE 8th Int. Conf. Ind. Inf. Syst.*, Dec. 2013, pp. 231–236.

[33] A. N. M. E. Rafiq, M. W. El-Kharashi, and F. Gebali, "A fast string search algorithm for deep packet classification," *Comput. Commun.*, vol. 27, no. 15, pp. 1524–1538, Sep. 2004.

[34] R. S. Boyer, and J. S.Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[35] M. A. Hernández, "A taxonomy of some right-to-left string-matching algorithms," in *Int. Workshop Funct. Constraint Logic Program.*, Heidelberg, Germany: Springer, 2010, pp. 79–95.

[36] G. Baeza-Yates and G. H. Gonnet, "A new approach to text searching" *Commun. ACM*, vol. 35, no. 10, pp. 74–82, Oct. 1992.

[37] D. Breslauer, L. Colussi, and L. Toniolo, "Tight comparison bounds for the string prefix-matching problem," *Inf. Process. Lett.*, vol. 47, no. 1, pp. 51–57, Aug. 1993.

[38] R. N. Horspool, "Practical fast searching in strings," *Softw., Pract. Exper.*, vol. 10, no. 6, pp. 501–506, Jun. 1980.

[39] A. Apostolico and R. Giancarlo, "The Boyer–Moore–Galil string searching strategies revisited," *SIAM J. Comput.*, vol. 15, no. 1, pp. 98–105, Feb. 1986.

[40] T. Raita, "Tuning the boyer-moore-horspool string searching algorithm," *Softw., Pract. Exper.*, vol. 22, no. 10, pp. 879–884, Oct. 1992.

[41] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter, "Speeding up two string-matching algorithms," *Algorithmica*, vol. 12, nos. 4–5, pp. 247–267, Nov. 1994.

[42] T. Berry and S. Ravindran,"A fast string matching algorithm and experimental results," in *Proc. Prague Stringology Club Workshop*, 1999, pp. 16–26.

[43] M. K. Ahmad, "An enhanced Boyer-Moore algorithm," Ph.D. dissertation, Dept. Comput. Sci., Fac. Inf. Technol., Middle East Univ., Amman, Jordan, 2014.

[44] D. M. Sunday, "A very fast substring search algorithm," *Commun. ACM*, vol. 33, no. 8, pp. 132–142, Aug. 1990.

[45] L. Colussi, "Correctness and efficiency of pattern matching algorithms," *Inf. Comput.*, vol. 95, no. 2, pp. 225–251, Dec. 1991.

[46] H. Xian-feng, Y. Yu-bao, and X. Lu, "Hybrid pattern-matching algorithm based on BM-KMP algorithm," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng. (ICACTE)*, vol. 5, pp. V5-310–V5-313, Aug. 2010.

[47] Z. Cao, Z. Yan, and L. Liu, "A fast string matching algorithm based on lowlight characters in the pattern," in *Proc. 7th Int. Conf. Adv. Comput. Intell. (ICACI)*, pp. 179–182, Mar. 2015.

[48] S. Hakak, A. Kamsin, P. Shivakumara, M. Y. I. Idris, and G. A. Gilkar, "A new split based searching for exact pattern matching for natural texts," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0200912.

[49] S. Hakak, K. Amirrudin, P. Shivakumara, and M. Y. I. Idris, "Partition-based pattern matching approach for efficient retrieval of arabic text," *Malaysian J. Comput. Sci.*, vol. 31, no. 3, pp. 200–209, 2018.

[50] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Develop.*, vol. 31, no. 2, pp. 249–260, Mar. 1987.

[51] W. S. Dorn, "Generalizations of Horner's rule for polynomial evaluation," *IBM J. Res. Develop.*, vol. 6, no. 2, pp. 239–245, Apr. 1962.

[52] J. Lee, "Analysis of fundamental exact and inexact pattern matching algorithms," Standford Univ., Stanford, CA, USA, Project Rep. BIOC 218, Jun. 2004.

[53] S. Fide and S. Jenks, "A survey of string matching approaches in hardware," Dept. Elect. Eng. Comput. Sci., Univ. California, Irvine, Irvine, CA, USA, Tech. Rep. TR SPDS 06-01, 2008.

[54] A. A. Abdulrazzaq, N. A. Rashid, and H. Y. Hamdani, "Influenced factors on computation among quick search, two-way and Karp-Rabin algorithms," in *Proc. 3rd Int. Conf. Inform. Technol.*, Oct. 2009, pp. 81–87.

[55] T. Lecroq, "Fast exact string matching algorithms," *Inf. Process. Lett.*, vol. 102, no. 6, pp. 229–235, Jun. 2007.

[56] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Dept. Comput. Sci., Univ. Arizona, Tucson, AZ, USA, Tech. Rep. TR-94-171994, May 1994.

[57] P. Kalsi, H. Peltola, and J. Tarhio, "Comparison of exact string matching algorithms for biological sequences," in *Proc. 2nd Int. Conf. Bioinf. Res. Develop. (BIRD)*, 2008, pp. 417–426.

[58] S. Kim and Y. Kim, "A fast multiple string-pattern matching algorithm," in *Proc. 17th AoM/IAoM Conf. Comput. Sci.*, 1999, pp. 44–49.

[59] F. Simone, "A very fast string matching algorithm based on condensed alphabets," In *Proc. Int. Conf. Algorithmic Appl. Manage.* Cham, Switzerland: Springer, 2016, pp. 65–76.

[60] W. Yang, "Mealy machines are a better model of lexical analyzers," *Comput. Lang.*, vol. 22, no. 1, pp. 27–38, Apr. 1996.

[61] R. Navarro and M. Raffinot, "Bit-parallel approach to suffix automata: Fast extended string matching," in *Proc. Annu. Symp. Combinat. Pattern Matching*, Berlin, Germany: Springer-Verlag, 1998, pp. 14–33.

[62] K. R. Rasool and N. Khare, "Parallelization of KMP string matching algorithm on different SIMD architectures: Multi-core and GPGPU&;" *Int. J. Comput. Appl.*, vol. 49, no. 11, pp. 26–28, Jul. 2012.

[63] J. A. Joseph, K. Reeba, and S. Salivahanan, "Efficient string matching FPGA for speed up network intrusion detection," *Appl. Math. Inf. Sci.*, vol. 12, no. 2, pp. 397–404, Mar. 2018.

[64] M. Aldwairi, Y. Flaifel, and K. Mhaidat, "Efficient WU-Manber pattern matching hardware for intrusion and malware detection," in *Proc. Int. Conf. Elect., Electron., Comput., Commun., Mech. Comput. (EECCMC)*, Tamil Nadu, India, Jan. 2018, pp. 1–6.

[65] X. Wang and D. Pao, "Memory-based architecture for multicharacter Aho–Corasick string matching," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 1, pp. 143–154, Jan. 2018.

[66] S. Hakak, A. Kamsin, P. Shivakumara, O. Tayan, M. Y. I. Idris, and G. A. Gilkar, "An efficient text representation for searching and retrieving classical diacritical arabic text," *Procedia Comput. Sci.*, 142, pp. 150–157, Jan. 2018.

[67] B. Commentz-Walter, *A String Matching Algorithm Fast on the Average*. Berlin, Germany: Springer, 1979, pp. 118–132.

[68] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: A new structure for pattern matching," in *SOFSEM, Theory and Practice of Informatics*. Berlin, Germany: Springer, 1999, pp. 295–310.

[69] R. Allauzen and M. Raffinot, "Simple optimal string matching algorithm," *J. Algorithms*, vol. 36, no. 1, pp. 102–116, Jul. 2000.

[70] L. He, B. Fang, and J. Sui, "The wide window string matching algorithm," *Theor. Comput. Sci.*, vol. 332, nos. 1–3, pp. 391–404, Feb. 2005.

[71] C. Liu, Y. Wang, D. Liu, and D. Li, "Two improved single pattern matching algorithms," in *Proc. 16th Int. Conf. Artif. Reality Telexistence–Workshops (ICAT)*, Hangzhou, China, Nov./Dec. 2006, pp. 419–422.

[72] S. Faro and T. Lecroq, "Efficient variants of the backward-oracle-matching algorithm," in *Proc. Prague Stringology Conf.*, Czech Republic, U.K., 2008, pp. 146–160.

[73] H. Fan, N. Yao, and H. Ma, "Fast variants of the backward-oracle-marching algorithm," in *Proc. 4th Int. Conf. Internet Comput. Sci. Eng.*, Washington, DC, USA: 2009, pp.56–59.

[74] W. Masaki, I. Hasuo, and K. Suenaga, "Efficient online timed pattern matching by automata-based skipping," in *Proc. Int. Conf. Formal Modeling Anal. Timed Syst.* Cham, Switzerland: Springer, 2017, pp. 224–243.

[75] F. Hongbo, S. Shupeng, Z. Jing, and D. Li, "Suffix type string matching algorithms based on multi-windows and integer comparison," in *Proc. Int. Conf. Inf. Commun. Secur.*. Cham, Switzerland: Springer, 2015, pp. 414–420.

[76] Y. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," *Commun. ACM*, 1992, pp. 74–82.

[77] G. Fredriksson and S. Grabowski, "Practical and optimal string matching," in *Proc. Int. Symp. String Process. Inf. Retr. (SPIRE)*, 2005, pp. 376–387.

[78] Intel.com. *XML Parsing Accelerator with Intel Streaming SIMD Extensions 4 (Intel-SSE4)*. Accessed: Apr. 5, 2015. [Online]. Available: https://software.intel.com/en-us/articles/xml-parsing-accelerator-with-intel-streaming-simd-extensions-4-intel-sse4

[79] H. Peltola and J. Tarhio, "Alternative algorithms for bit-parallel string matching," in *Proc. 10th Int. Symp. String Process. Inf. Retr. (SPIRE)*, 2003, pp. 80–93.

[80] K. Ulekci and M., "Filter based fast matching of long patterns by using SIMD instructions," in *Proc. Prague Stringology Conf.*, Prague, Czech Republic, Jan. 2009, pp. 126–280.

[81] M. O. Külekci, "A method to overcome computer word size limitation in bit-parallel pattern matching," in *Proc. 19th Int. Symp. Algorithms Comput., (ISAAC)*, 2008, pp. 496–506.

[82] G. Zhang, E. Zhu, L. Mao, and M. Yin, "A bit-parallel exact string matching algorithm for small alphabet," in *Frontiers in Algorithmics*. Berlin, Germany: Springer, 2009, pp. 336–345.

[83] F. Franek, C. G. Jennings, and W. F. Smyth, "A simple fast hybrid pattern-matching algorithm," *J. Discrete Algorithms*, vol. 5, no. 4, pp. 682–695, Dec. 2007.

[84] M. Crochemore, A. Czumaj, L. Gąsieniec, T. Lecroq, W. Plandowski, and W. Rytter, "Fast practical multi-pattern matching," *Inf. Process. Lett.*, vol. 71, nos. 3–4, pp. 107–113, Aug. 1999.

[85] M. C. A. V. Aho, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[86] G. Navarro, "Nrgrep: A fast and flexible pattern matching tool," Dept. Comput. Sci., Univ. Chile, Santiago, Chile, Tech. Rep. TR/DCC-2000-3, Aug. 2000. [Online]. Available: ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/nrgrep.ps.gz

[87] H.-T. Lu and W. Yang, "A simple tree pattern-matching algorithm," in *Proc. Workshop Algorithms Theory Comput.*, 2000, pp. 1–8.

[88] Y. Bai and H. Kobayashi, "New string matching technology for network security," in *Proc. 17th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2003, pp. 27–29.

[89] M. Ahmed, M. Kaykobad, and R. A. Chowdhury, "A new string matching algorithm," *Int. J. Comput. Math.*, vol. 80, no. 7, pp. 825–834, Jul. 2003.

[90] D. Cantone and S. Faro, "Fast-search: A new efficient variant of the Boyer-Moore string matching algorithm," in *Experimental and Efficient Algorithms* (Lecture Notes in Computer Science), vol. 2647, K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim, Eds. Berlin, Germany: Springer, 2003.

[91] D. Cntone and S. Faro, "Searching for a substring with constant extra-space complexity," in *Proc. 3rd Int. Conf. Fun Algorithms*, 2004, pp. 118–131.

[92] G. Navarro and M. Raffinot, "A bit-parallel approach to suffix automata: Fast extended string matching," in *Combinatorial Pattern Matching* (Lecture Notes in Computer Science), vol. 1448, M. Farach-Colton, Ed. Berlin, Germany: Springer, 1998.

[93] B. U. J. Holub, "Fast variants of bit parallel approach to suffix automata," in *Proc. 2nd Haifa Annu. Int. Stringol. Res. Workshop Israeli Sci. Found.*, 2005, pp. 1–20.

[94] M. E. Nebel, "Fast string matching by using probabilities: On an optimal mismatch variant of Horspool's algorithm," *Theor. Comput. Sci.*, vol. 359, nos. 1–3, pp. 329–343, Aug. 2006.

[95] S. S. Sheik, S. K. Aggarwal, A. Poddar, N. Balakrishnan, and K. Sekar, "A FAST pattern matching algorithm," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 4, pp. 1251–1256, Jul./Aug. 2004.

[96] R. Thathoo, A. Virmani, S. S. Lakshmi, N. Balakrishnan, and K. Sekar, "TVSBS: A fast exact pattern matching algorithm for biological sequences," *Current Sci.*, vol. 91, no. 1, pp. 47–53, Jul. 2006.

[97] Z. A. Alqadi, M. Aqel, and I. M. El Emary, "Multiple skip multiple pattern matching algorithm (MSMPMA)," *IAENG Int. J. Comput. Sci.*, vol. 34, no. 2, pp. 14–20, Dec. 2007.

[98] N.-F. Huang, Y.-M. Chu, C.-Y. Hsieh, C.-H. Tsai, and Y.-J. Tzang, "A deterministic cost-effective string matching algorithm for network intrusion detection system," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2007, pp. 1292–1297.

[99] A. Hudaib, D. Suleiman, M. Itriq, and A. Al-Anani, "A fast pattern matching algorithm with two sliding windows (TSW)," *J. Comput. Sci*, vol. 4, no. 5, pp. 393–401, 2008.

[100] S. Deusdado and P. Carvalho, "GRASPm: An efficient algorithm for exact pattern-matching in genomic sequences," *Int. J. Bioinf. Res. Appl.*, vol. 5, no. 4, pp. 385–401, 2009.

[101] S. Cho, J. C. Na, K. Park, and J. S. Sim, "A fast algorithm for order-preserving pattern matching," *Inf. Process. Lett.*, vol. 115, no. 2, pp. 397–402, Feb. 2015.

[102] S. K. Pandey, H. K. Tiwari, and P. Tripathi, "Hybrid approach to reduce time complexity of string matching algorithm using hashing with chaining," in *Proc. Int. Conf. ICT Sustain. Develop.* Singapore: Springer, 2016, pp. 185–193.

[103] A. M. Al-Ssulami, "Hybrid string matching algorithm with a pivot," *J. Inf. Sci.*, vol. 41, no. 1, pp. 82–88, Feb. 2015.

[104] S. Faro and T. Lecroq. (2010)."The exact string matching problem: A comprehensive experimental evaluation." [Online]. Available: https://arxiv.org/abs/1012.2547

[105] N. A. K. Ersin, "A research into string matching algorithms and a new string matching algorithm," M.S. thesis, Dept. Comput. Eng., Trakya Univ., Edirne, Turkey, 2008.

[106] P. Kalsi, H. Peltola, and J. Tarhio, "Comparison of exact string matching algorithms for biological sequences," in *Bioinformatics Research and Development* (Communications in Computer and Information Science), vol. 13. Berlin, Germany: Springer, 2008, pp. 1–10.

[107] T. Lecroq, "Experimental results on string matching algorithms," *Softw., Pract. Exper.*, vol. 25, no. 7, pp. 727–765, Jul. 1995.

[108] H. Zhang, "Parallelization of software based intrusion detection system," M.S. thesis, Univ. Canterbury, Christchurch, New Zealand, 2011.

[109] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *J. Parallel Distrib. Comput.*, vol. 74, no. 7, pp. 2561–2573, Jul. 2014.

[110] W. O. Science. (2016). *Web of Science*. [Online]. Available: http://isiknowledge.com/wos

[111] S. Hakak, A. Kamsin, O. Tayan, M. Y. I. Idris, and G. A. Gilkar, "Approaches for preserving content integrity of sensitive online Arabic content: A survey and research challenges," *Inf. Process. Manage.*, vol. 56, no. 2, pp. 367–380, Mar. 2017.

[112] S. Hakak, A. Kamsin, S. Palaiahnakote, O. Tayan, M. Y. I. Idris, and K. Z. Abukhir, "Residual-based approach for authenticating pattern of multi-style diacritical Arabic texts," *PLoS ONE*, vol. 13, no. 6, 2018, Art. no. e0198284.

[113] A. K. Abdulrazzaq, N. A. Rashid, and H. A. A. Alezzi, "Parallel processing of hybrid exact string," in *Proc. IEEE Int. Conf. Control Syst., Comput. Eng.*, Mindeb, Malaysia, Nov./Dec. 2013.

[114] Quora.com. (2015). *What-is-the-Difference-Among-CPU-GPU-APU-FPGA-DSP-and-Intel-MIC*. [Online]. Available: www.quora.co

[115] C.-L. Hung, C.-Y. Lin, and H.-H. Wang, "An efficient parallel-network packet pattern-matching approach using GPUs," *J. Syst. Archit.*, vol. 60, no. 5, pp. 431–439, May 2014.

[116] O. Tayan and Y. M. Alginahi, "A review of recent advances on multimedia watermarking security and design implications for digital Quran computing," in *Proc. Int. Symp. Biometrics Secur. Technol. (ISBAST)*, Aug. 2014, pp. 304–309.

[117] S. Hakak, A. Kamsin, O. Tayan, M. Y. I. Idris, A. Gani, and S. Zerdoumi, "Preserving content integrity of digital holy Quran: Survey and open challenges," *IEEE Access*, vol. 5, pp. 7305–7325, 2017.

[118] S. Zerdoumi *et al.*, "Image pattern recognition in big data: Taxonomy and open challenges: Survey," *Multimedia Tools Appl.*, vol. 77, no. 8, pp. 10091–10121, Apr. 2017.

SAQIB IQBAL HAKAK received the bachelor's degree in computer science engineering from the University of Kashmir, India, in 2010, the M.S. degree in computer and information engineering from IIUM, Malaysia, and the Ph.D. degree from the University of Malaya, Malaysia, under the faculty of Computer Science and Information Technology. His research areas include information security, natural language processing, cloud computing, cyber security, deep learning and wireless networks.

AMIRRUDIN KAMSIN received the B.I.T. degree in management the University of Malaya, in 2001, the M.Sc. degree in computer animation from Bournemouth University, U.K., and the Ph.D. degree from University College London (UCL), in 2014. He is currently a Senior Lecturer with the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. His research areas include human computer interaction (HCI), authentication systems, e-learning, mobile applications, serious game, augmented reality, and mobile health services.

PALAIAHNAKOTE SHIVAKUMARA received the B.Sc., M.Sc., M.Sc. Technology by research, and Ph.D. degrees from the University of Mysore, Mysore, Karnataka, India, in 1995, 1999, 2001, and 2005, respectively, all in computer science. He was a Research Fellow with the National University of Singapore, Singapore, from 2005 to 2007 and from 2008 to 2013. Besides, he was Research Consultant with Nanyang Technological University, Singapore, from 2007 to 2008. He is currently a Senior Lecturer with the University of Malaya (UM), Kuala Lumpur, Malaysia. Based on his work, he has published more than 190 research papers in national, international conferences and journals. His research interests include video text understanding, document analysis, image processing, and OCR related. He was a recipient of a prestigious Dynamic Indian of the Millennium Award by KG foundation, India, for his contributions to computer science field. He received the Top Reviewer Award from *Pattern Recognition Letters*. He has several international collaborators, namely, Nanjing University, China, Hohai University, China, Shantou University, China, Indian Statistical Institute, Kolkata, India, University of Essex, U.K., Assiut University, Egypt, and University of Technology Sydney, Australia. He has been serving as a Chair of different levels for International Conference, namely, ICDAR, DAS, ICFHR, and ACPR. He has been serving as an Associate Editor for *Transactions on Asian Language Information Processing* (TALLIP).

GULSHAN AMIN GILKAR received the bachelor's and master's degrees from India. She is currently a Lecturer with the Department of Computer Science and Information Technology, Faculty of Computer Science and Information Technology, Shaqra University, Saudi Arabia. She has vast teaching experience since she has worked in various educational institutions locally and abroad.

WAZIR ZADA KHAN (M'16–SM'17) received the B.S. and M.S. degrees in computer science from COMSATS University Islamabad, Wah Campus, in 2004 and 2007, respectively, and the Ph.D. degree from the Electrical and Electronic Engineering Department, Universiti Teknologi PETRONAS, Malaysia, in 2015. He is currently with the Farasan Networking Research Laboratory, Faculty of CS & IS, Jazan University, Saudi Arabia. His current research interests include wireless sensor networks, security and privacy, the Internet of Things.

MUHAMMAD IMRAN is currently an Associate Professor with the College of Applied Computer Sciences, King Saud University (KSU). He has published a number of research papers in refereed international conferences and journals. His research interest includes mobile and wireless networks, the Internet of Things, cloud/edge computing, and information security. His research is financially supported by several grants. He has been involved in more than 75 conferences and workshops in various capacities, such as a Chair, a Co-chair, and a Technical Program Committee Member. These include IEEE ICC, GLOBECOM, AINA, LCN, IWCMC, IFIP WWIC, and BWCCA. He has received number of awards such as the Asia Pacific Advanced Network Fellowship. Recently, European Alliance for Innovation (EAI) has appointed him as an Editor-in-Chief for the *EAI Transactions on Pervasive Health and Technology*. He also serves as an Associate Editor for the *IEEE Communications Magazine*, *Future Generation Computer Systems*, the IEEE ACCESS, *Wireless Communication and Mobile Computing Journal*, *Ad Hoc & Sensor Wireless Networks Journal*, *IET Wireless Sensor Systems*, the *International Journal of Autonomous and Adaptive Communication Systems*, and the *International Journal of Information Technology and Electrical Engineering*.