Journal of Theoretical and Applied Information Technology
15th June 2020. Vol.98. No 11
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

APPROXIMATE MULTIPLE STRING MATCHING ALGORITHM

J UJWALA REKHA
Dept. of CSE, JNTUH College of Engineering Hyderabad, India
E-mail: ujwala_rekha@jntuh.ac.in

ABSTRACT

Approximate multiple string matching comprises of finding all appearances of a set of patterns within a larger text with at most k-mismatches or k-differences. It is a process that consumes large amounts of computational resources and has applications in many domains of computer science such as correction of spelling, spam filtering and computational biology to name a few. However, these applications require a fast algorithm for string matching which depends on the construction of an accurate shift table. When a mismatch occurs while searching a pattern, a shift table specifies the maximum number of characters that can be skipped in text while ensuring that a possible match is not overlooked. This study presents a simple and efficient algorithm that is a variant of Tarhio-Ukkonen algorithm generalized to approximate multiple string matching. The algorithm searches for patterns of different lengths with at most *k*-mismatches by skipping some of the characters, while not missing the occurrences of any pattern. While the space and time complexity for construction of shift table is $O(|S|*(k+1)\times c)$ and $O(M\times k\times c)$, the time complexity of search algorithm is $O(M\times n)$, where $|S|$ is the number of patterns, c is the number of characters in the alphabet, n is the length of the text string and M is the total length of all patterns. To evaluate the efficacy of the algorithm, experiments are conducted on random text and random patterns of different lengths and different values of k.

Keywords: *Approximate String Matching, Multiple String Matching, Tarhio-Ukkonen Algorithm*

1. INTRODUCTION

The string matching commonly referred to as string searching is a most common issue in computer science with applications in text processing, file comparison, spam filters, information retrieval systems, and DNA sequence analysis to name a few. In the recent years, the range of string searching applications is extended to big data and social media analysis as well [1]. Due to the ever increasing scope of string matching applications, a suitable technique that can be customized or employed for latest applications and datasets ought to be chosen [2].

There are various classifications of algorithms pertaining to string matching but primarily they fall into two categories: exact and approximate string matching. While exact string matching finds the instances of a single pattern in a larger text ([3],[4],[5],[6]), approximate string searching finds the substrings within the text that are closer to the pattern ([7],[8],[9],[10],[A]). Approximate string searching problems can be categorized into: *k*- mismatches and *k*-differences problem. While approximate string searching with *k*-mismatches tries to search all the appearances of a pattern with not more than *k*-mismatches, *k*-difference approximate string matching searches substrings within the text string whose edit-distance is not more than *k* with the pattern. A simple edit distance is the minimum number of insertions, deletions and substitutions required to transform one string into another. If the cost of all edit-operations is equal, then we say *simple edit distance*. On the other hand, if different edit-operations have different costs and/or the cost of edit-operation is dependent on the characters involved, we say *general edit distance*. String matching algorithms that consider simple edit distance can be easily extended to consider general edit distance. Similarly, multiple string matching searches more than one pattern instead of a single pattern within a given text ([11],[12]); and approximate multiple string matching constitutes of searching all appearances of a set of patterns within a larger text with less than or equal to *k*-mismatches or *k*-differences ([13], [14],[15],[16]).

The efficiency of the string searching algorithms depends on the preprocessing time required for construction of a shift table. A shift

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

table enumerates the maximum number of characters that can be skipped in text while not overlooking a possible match. Generally, a string matching algorithm that employs a shift table works as follows: Initially, the pattern is aligned at the beginning of the text and compared character by character tracking the number of matches. If a match occurs, it is reported and the text is skipped according to the distance specified in the shift table to align the pattern at the location of next possible occurrence. Subsequently, the pattern is compared against the text from the aligned position. Shifting and matching is repeated until the pattern reaches the end of the text.

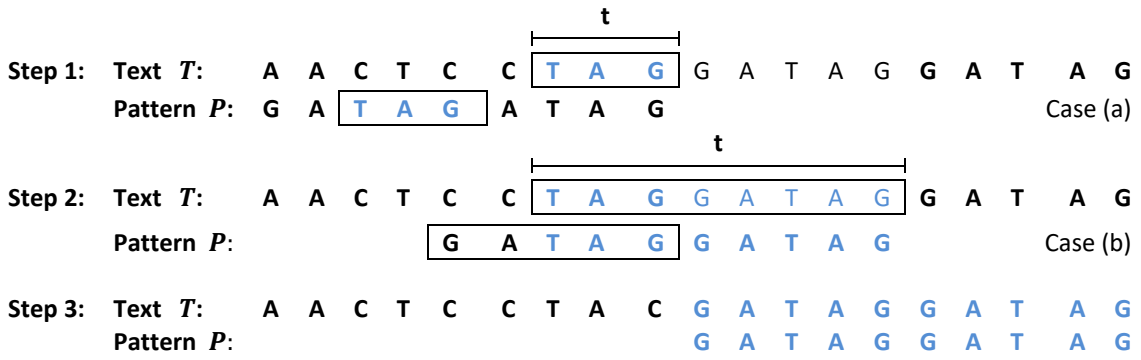

Figure 1(a). Illustration of Good Suffix Rule

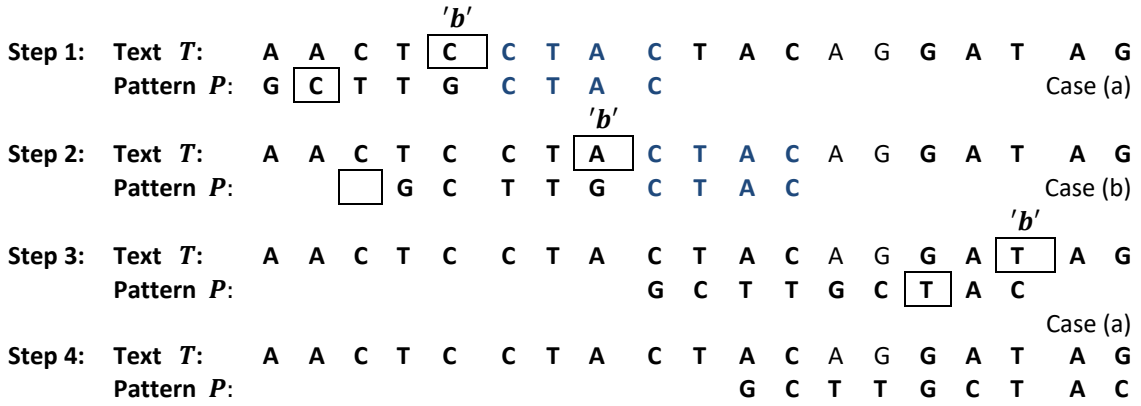

Figure 1(b). Illustration of Bad Character Rule

Approximate string matching algorithms for k-mismatches problem are proposed by Baeza-Yates-Gonnet [17] and Tarhio-Ukkonen [8] which are generalizations of Boyer-Moore algorithm [4]. While Baeza-Yates-Gonnet computes shift distances based on good suffix rule, Tarhio-Ukkonen algorithm is derived from the bad character rule. While the good suffix rule uses the knowledge of the matched characters, the bad character rule uses the knowledge of mismatched characters in order to skip the characters in the text that certainly do not match. The good suffix rule and the bad character rule are defined below and illustrated in Figure 1(a) and Figure 1(b) respectively.

i. **Good Suffix Rule:** If $'t'$is the substring of text $'T'$matching a suffix of pattern $'P'$, then the pattern $'P'$ can be shifted to the right until (a) the substring $'t'$ in the text matches the previous occurrence of $'t'$ in the pattern, if it exists, or (b) a prefix of $'P'$ matches a suffix of substring $'t'$, or (c) $'P'$moves past $'t'$in the text.

ii. **Bad Character Rule:** For a particular alignment of $'P'$against text $'T'$, let $'b'$ be the rightmost mismatched character in $'T'$, then shift $'P'$ to the right until (a) the mismatched character $'b'$in text matches a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

previous occurrence of $'b'$ in the pattern, or (b) $'P'$ moves beyond $'b'$ in the text.

A bad character rule prevents repeated unsuccessful comparisons. Consequently, in this study, Tarhio-Ukkonen algorithm generalized to approximate multiple string searching with k-mismatches is presented. In Tarhio-Ukkonen algorithm, the pattern is aligned with the text string by skipping the letters such that the k+1 letters at the end within the text string of the previous alignment contains more than one character match with the pattern. In contrast to Tarhio-Ukkonen algorithm, the proposed algorithm computes the shift distance by taking into account multiple patterns rather than a single pattern. The patterns are aligned with the text such that the k+1 letters at the end within the text string of the previous alignment contains more than one character match with any of the patterns in the set.

The remaining of the study covers the following topics: Section 2 presents the applications of approximate string matching. While in Section 3, the Tarhio-Ukkonen algorithm is explained along with an illustration, Section 4 explains generalized Tarhio-Ukkonen algorithm for approximate multiple string searching with an illustration. Section 5 portrays the results of the experiments conducted, and finally the conclusions are given in Section 6.

2. Applications of Approximate String Matching

The main motivation for approximate string matching is derived from computational biology, signal processing and text retrieval.

Computational Biology:

A DNA sequence is a long text over alphabet $\{A, C, G, T\}$. Searching a specific sequence in another sequence can be modeled as a string matching problem and is fundamental in medical diagnosis, comparing healthy and mutated DNA sequences and characterization of antibody repertoire to name a few. However, exact string searching is not practical since DNA patterns seldom match absolutely due to mutations and evolutionary alterations. Consequently, finding similar sequences rather than exact matches is more promising. This gave rise to motivation of "approximate string matching" that allows errors in searching. Two sequences are similar if the distance between the sequences is less than some threshold where the distance between two sequences is defined as the minimum number of operations required to transform one sequence into another. The operations can be assigned a cost depending on its likelihood. For instance, a more likelihood operation is cheaper and vice versa.

Signal Processing:

Speech recognition constitutes of determining a textual message being transmitted in a speech signal. For instance, it consists of detecting boundaries between words in a speech signal and then determining what word each segment represents. Even for a small vocabulary, automatic speech recognition is a complex problem due to variations in accent, pronunciation, articulation, volume and speed by various persons. Consequently, exact matching of signals is impractical and the text closest to the received message is obtained.

Text Retrieval:

Information retrieval is the activity of finding information that is relevant from a large collection of information resources and string searching is one of the fundamental tools. In an information retrieval system, a user enters a query and the system searches for relevant information matching the query, where a query is a statement that specifies the information needs. Generally, typographical errors are present in both the query and the documents being searched. Consequently, classical string matching is not sufficient and approximate string matching is a better option. Other text processing applications of approximate string matching are correction of spelling, document classification and text categorization to name a few.

3. TARHIO-UKKONEN ALGORITHM [8]

Let $P = p_1 \ldots p_m$ represent the pattern and $T = t_1 \ldots t_n$ represent the string of text over an alphabet Σ and k an integer. Approximate string searching of $k-$mismatches problem constitutes of searching for all appearances of pattern P within the text string T with less than or equal to $k-$ mismatches.

Let the pattern terminate at position j in the string of text then the rightmost $k+1$ letters in the text aligned above the pattern are $t_{j-k}, t_{j-k+1}, \ldots, t_j$. Subsequently, the pattern must be shifted towards right such that there is more than one character match within $t_{j-k}, t_{j-k+1}, \ldots, t_j$. The

Journal of Theoretical and Applied Information Technology
15th June 2020. Vol.98. No 11
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

upper limit on this shift distance is $m-k$. Let $d_k[t_{j-k},...,t_j]$ represent the amount of shift distance, then $d_k[t_{j-k},...,t_j]$ is the least of the elemental shifts for the letters t_{j-k}, t_{j-k+1}, ... , t_j. The elemental shift for character t_h denoted as $d_k[i,t_h]$ is calculated from the position i of character t_h above the pattern where $j-k \le h \le j$. Consequently, the possible positions for character t_h are $m-k$, ... , m; furthermore for all $m-k \le i \le m$ and $a \in \Sigma$, $d_k[i,a]$ is the least distance between $p_i = a$ and the next occurrence $p_s = a$ for $1 \le s < i$ defined as follows:

$$d_k[i,a] = \min\left\{\min_{1\le s<i}\{\, s \mid p_{i-s} = a\,\} \cup \{\, m-k\,\}\right\} \quad (1)$$

Thus, we can now define $d_k[t_{j-k},...,t_j]$ formally as follows:

$$d_k[t_{j-k},...,t_j] = \min_{0\le i\le k} d_k[m-i,t_{j-i}] \quad (2)$$

Algorithm 1: Calculation of Shift Table d_k

```
1.  for all a ∈ Σ do
    //For each letter in the alphabet
2.    for i ← m down to m − k do
      //For k+1 positions
3.      s ← i − 1; d_k[i,a] ← m − k;
        chg ← 0;
4.      while (i − s < m − k) && chg != 1
5.        if (p_s = a) then
6.          chg ← 1; d_k[i,a] ← i − s;
7.        end of if
8.        else s ← s − 1;
9.      end of while
10.   end of for
11. end of for
```

The computation of elemental shifts $d_k[i,a]$ for all $m-k \le i \le m$ and $a \in \Sigma$ is presented in Algorithm 1. Subsequently, approximate string searching process with $k-$ mismatches is presented in Algorithm 2.

The time complexity of Tarhio-Ukkonen algorithm for $k-$mismatches problem is $O\left(kn\left(\frac{1}{m-k}+\frac{k}{c}\right)\right)$; where c is the number of characters in the alphabet Σ [8].

Algorithm 2: Approximate String Matching

```
1.  j ← 1;
    while (j ≤ (n − m + 1))
2.
3.      h ← j; i ← 1; e ← 0
        // h index of text and i index of pattern
4.      while (h ≤ j + m − 1)
5.        if (i ≤ |P|) and (t_h ≠ p_i) then e++;
6.        end of if
7.        i++; h++;
8.      end of while
9.      if e > k then
10.       j = j + d_k[t_{j−k},...,t_j] computed
          according to (2)
12.     else
13.       report the occurrence of pattern P at
          position j
14.     end of if else
15. end of while
```

An illustration of the computation of shifts and the working of Tarhio-Ukkonen algorithm is presented for pattern $P = AATGCCTTAC$, text string $T = AACTGCCTAACCCGCACAAC$ and $k = 2$. The elemental shifts $d_k[i,a]$ are calculated according to (1) and presented in Table 1, and the shifts corresponding to approximate string matching for P are computed according to Algorithm 2 and are described in Table 2.

Table 1. Shift table for $P = AATGCCTTAC$ *&* $k = 2$

Position	A	C	G	T
8	6	2	4	1
9	7	3	5	1
10	1	4	6	2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2. Illustration of Tarhio-Ukkonen Algorithm 2 for $k=2$, $m=10$ and $n=20$

Text:	**A**	**A**	**C**	**T**	**G**	**C**	**C**	**T**	**A**	**A**	C	C	C	G	C	A	C	A	A	C
Pattern:	**A**	**A**	**T**	**G**	**C**	**C**	**T**	**T**	**A**	**C**										
Shift 1		A	A	T	G	C	C	T	T	A	C									
Shift 2						A	A	T	G	C	C	T	T	A	C					
Shift 3								A	A	T	G	C	C	T	T	A	C			
Shift 4										A	A	T	G	C	C	T	T	A	C	
Shift 5											A	A	T	G	C	C	T	T	A	C

Shift 1:	$d_k[TAA] = \min\{ d_k[8,T], d_k[9,A], d_k[10,A]\} = 1$ MOVE
Shift 2:	$d_k[AAC] = \min\{d_k[8,A], d_k[9,A], d_k[10,C]\} = 4$ MOVES
Shift 3:	$d_k[CGC] = \min\{d_k[8,C], d_k[9,G], d_k[10,C]\} = 2$ MOVES
Shift 4:	$d_k[CAC] = \min\{d_k[8,C], d_k[9,A], d_k[10,C]\} = 2$ MOVES
Shift 5:	$d_k[CAA] = \min\{d_k[8,C], d_k[9,A], d_k[10,A]\} = 1$ MOVE

* It can be noted that the text contains one instance of pattern at position 2 in the text string with two mismatches.

4. PROPOSED ALGORITHM FOR APPROXIMATE MULTIPLE STRING MATCHING

Let $S = \{P^1, P^2, ...\}$ be a finite set of patterns (where $P^r = p_1^r p_2^r ...$) and $T = t_1 ... t_n$ be the string of text defined over a character set Σ, then the approximate multiple string searching problem constitutes of finding all appearances of each of the patterns of set S within the string of text T with a maximum of $k-$ mismatches.

Assume that the patterns are aligned with the text beginning at text position j. Let m_r be the length of the pattern P^r, then the pattern P^r ends at text position $j+m_r-1$. Consider the rightmost $k+1$ text characters $t_{j+m_r-k-1}, ..., t_{j+m_r-1}$ aligned above the pattern P^r. Similar to Tarhio-Ukkonen algorithm [8], let $d^r[t_{j+m_r-k-1} ... t_{j+m_r-1}]$ represent the length of the shift corresponding to pattern P^r, then $d^r[t_{j+m_r-k-1} ... t_{j+m_r-1}]$ is the least of the elemental shifts for each of $t_{j+m_r-k-1}, ..., t_{j+m_r-1}$. The elemental shift for t_h is calculated from character t_h itself and its position above the pattern P^r and is denoted as $d_k^r[\, i, t_h]$, where i is the position of t_h above the pattern and $j+m_r-k-1 \le h \le j+m_r-1$. Consequently, the possible positions for t_h are m_r-k, ... , m_r; and for all $m-k \le i \le m$ and $a \in \Sigma$, $d_k^r[i,a]$ can be defined as follows:

$$d_k^r[i,a] = \min\left\{\{m_r - k\} \cup \min_{1 \le s < i}\left\{s \mid p_{i-s}^r = a\right\}\right\} \quad (3)$$

Algorithm 3: Computation of d_k^r

```
1.  for r ← 1 to |S| do    //For each pattern
      for a in Σ do
2.     //For each letter in the alphabet
3.      for i ← m_r down to m_r − k do
        //For k+1 positions
4.        s ← i − 1;
          d_k^r[i,a] ← m_r − k; chg ← 0;
5.        while i − s < m_r − k && chg ! = 1
6.          if p_l^r = a then
7.            chg ← 1; d_k[i,a] ← i − s;
8.          else s ← s − 1;
9.        end of while
10.     end of for
11.   end of for
12. end of for
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The details for computation of shift table $d_k^r[i,a]$ according to (3) are presented in Algorithm 3. Furthermore, an illustration for the computation of shift table for patterns $P = \{AATGCCTTAC, ATGCC, CGTAAC\}$ is presented in Table 3.

Table 3. A Shift table for $k = 2$ for patterns $P = \{AATGCCTTAC, ATGCC, CGTAAC\}$

Pattern *AATGCCTTAC*				
Position	**A**	**C**	**G**	**T**
8	6	2	4	1
9	7	3	5	1
10	1	4	6	2
Pattern *ATGCC*				
Position	**A**	**C**	**G**	**T**
3	2	3	3	1
4	3	3	1	2
5	3	1	2	3
Pattern *CGTAAC*				
Position	**A**	**C**	**G**	**T**
4	4	3	2	1
5	1	4	3	2
6	1	4	4	3

While matching, each of the patterns P^r is compared with the text string T. The index of the text character being scanned is denoted by h and the corresponding index of the character in the pattern P^r is denoted by i. All the patterns are aligned with the text starting at text position j. Subsequently, the length of the shift $d_k^r[t_{j+m_r-k-1}...t_{j+m_r-1}]$ corresponding to pattern P^r is computed from the elemental shifts of each of $t_{j+m_r-k-1},...,t_{j+m_r-1}$ as follows:

$$d_k^r[t_{j+m_r-k-1}...t_{j+m_r-1}] = \min_{i\in[0,k]}\left\{d_k^r[m_r-i, t_{j+(m_r-i)-1}]\right\} \quad (4)$$

Let $A \subseteq S$ denote a set of patterns that satisfy the condition $j \le n - m_r + 1$ (i.e., patterns that span within the text when aligned at position j in the text); then the minimum shift required to align the patterns in A such that at least one of the patterns has more than one character match in $t_{j+m_r-k-1},...,t_{j+m_r-1}$ is computed as follows:

$$d[t_j] = \min_{1\le r\le|S|}\left\{d_k^r[t_{j+m_r-k-1}...t_{j+m_r-1}]\right\} \quad (5)$$

The details of the approximate multiple string matching process is presented in Algorithm 4. The working of the proposed algorithm is illustrated in Table 4 for patterns $P = \{AATGCCTTAC, ATGCC, CGTAAC\}$ and the text $T = AACAGATGAGCCCGA$ with $k = 2$.

Algorithm 4: Approximate Multiple String Matching

```
1.  j ← 1;
2.  while (j ≤ (n − s + 1))
    // s is the length of the shortest pattern
3.    for r ← 1 to |S| do
4.      e_r = 0;
5.    end of for
6.    h ← j; i ← 1;
      // h index of text and i index of pattern
7.    while (h ≤ j + l − 1)
      // l is the length of the longest pattern
8.      for r ← 1 to |S| do
9.        if (j ≤ n − m_r + 1) and (i ≤ |P^r|)
          and (t_h ≠ p_i^r) then e_r ++;
10.       end of if
11.     end of for
12.     i++; h++;
13.   end of while
14.   for r = 1 to |S| do
15.     if e_r ≤ k then
16.       report the occurrence of pattern
          P^r at position j
17.     end of if
18.   end of for
19.   j = j + d[t_j] computed according to (5)
20.   end of while
```

4.1. Time Complexity

The time complexity for calculation of table $d_k[i,a]$ for $m-k \le i \le m$ and $a \in \Sigma$ is $O(m \times k \times c)$, where c is the number of characters in the alphabet Σ. Consequently, the preprocessing time required for multiple patterns is $O(M \times k \times c)$; where M is the total length of all patterns. Meanwhile, the time complexity of matching process of the proposed algorithm is $O(M \times n)$ in the worst case.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 4. Illustration of the proposed algorithm for $k = 2$, $n = 15$ and $k = 3$ patterns

Text:	A	A	C	A	G	A	T	G	A	G	C	C	C	G	A	
Pattern 1:	A	A	T	G	C	C	T	T	A	C						$d_2^1[GAG] = 4$
Pattern 2:	A	T	G	C	C											$d_2^2[CAG] = 2$
Pattern 3:	C	G	T	A	A	C										$d_2^3[AGA] = 1$
Shift 1 1 move		A	A	T	G	C	C	T	T	A	C					$d_2^1[AGC] = 4$
		A	T	G	C	C										$d_2^2[AGA] = 1$
		C	G	T	A	A	C									$d_2^3[GAT] = 1$
Shift 2 1 move			A	A	T	G	C	C	T	T	A	C				$d_2^1[GCC] = 3$
			A	T	G	C	C									$d_2^2[GAT] = 3$
			C	G	T	A	A	C								$d_2^3[ATG] = 2$
Shift 3 2 moves				A	A	T	G	C	C	T	T	A	C			$d_2^1[CCG] = 2$
				A	T	G	C	C								$d_2^2[TGA] = 1$
				C	G	T	A	A	C							$d_2^3[GAG] = 1$
Shift 4 2 moves					A	A	T	G	C	C	T	T	A	C		
					A	T	G	C	C							$d_2^2[GAG] = 2$
					C	G	T	A	A	C						$d_2^3[AGC] = 3$
	Report occurrence of pattern "ATGCC" at position 6 in the text with two mismatches															
Shift 5 2 moves							A	T	G	C	C					$d_2^2[GCC] = 1$
							C	G	T	A	A	C				$d_2^3[CCC] = 3$
	Report next occurrence of pattern "ATGCC" at position 8 in the text with two mismatches															
Shift6 1move								A	T	G	C	C				$d_2^2[CCC] = 1$
								C	G	T	A	A	C			$d_2^3[CCG] = 3$
	Report next occurrence of pattern "ATGCC" at position 9 in the text with two mismatches															
Shift 7 1 move									A	T	G	C	C			$d_2^2[CCG] = 2$
									C	G	T	A	A	C		

*It can be noted that there are three overlapping instances of "ATGCC" at positions 6, 8, and 9 in the text string.

5. EXPERIMENTAL RESULTS

In order to evaluate the approximate multiple string searching algorithm presented in this study, experiments are conducted extensively on random texts of 2 million characters with alphabets of different sizes, and then random patterns of different lengths and different values of k. Two sets of experiments are conducted: in one set of experiments approximate string matching algorithm presented in the study is considered and in another set of experiments Tarhio-Ukkonen algorithm is executed on each pattern separately. The experiments are conducted on Intel i3 CPU @ 2.6 GHz and 2GB RAM. For each set of tests, average shift distance and the number of character comparisons are reported in Table 5.

The average shift distance is as small as 1.9 characters and as large as 4.7 characters for the

proposed algorithm. On the other hand, the least average shift distance is 2.8 characters and the highest is 5.4 characters when Tarhio-Ukkonen algorithm is executed separately on each pattern. Furthermore, it can be noted that the average shift distance is more in the second set of experiments when compared to the proposed algorithm. Across two sets of experiments, the number of character comparisons done is more for smaller average shift distances and lesser for larger average shift distances. An important observation is, "in spite of larger shift distances, the number of character comparisons in the second set of experiments is more when compared to the number of character comparisons in the proposed algorithm.

Table 5.Comparison of the results of the proposed approximate multiple string matching and Tarhio-Ukkonen algorithm executed separately on each pattern

c	m_a	k	$\|S\|$	**Proposed Algorithm**		**Tarhio-Ukkonen Algorithm**	
				Average of Shift Distance	**Number of Character Comparisons (×2M)**	**Average of Shift Distance**	**Number of Character Comparisons (×2M)**
3	**8**	**3**	**2**	2.6	5.8	3.5	8.2
3	**16**	**3**	**2**	3.4	5.8	4.2	7.5
3	**32**	**3**	**2**	4.7	4.5	5.4	6.2
4	**8**	**3**	**3**	3.2	5.6	3.2	8.7
4	**8**	**6**	**3**	2.5	6.1	3.4	8.2
4	**8**	**8**	**4**	1.9	6.2	2.8	10.2

6. CONCLUSIONS

A simple and efficient method for approximate multiple string matching that is a variant of Tarhio-Ukkonen algorithm is proposed. The proposed algorithm finds all occurrences of the patterns that are of different lengths with at most $k-$ mismatches. To evaluate the effectiveness of the algorithm, experiments are run on random patterns and random texts. Proposed algorithm for approximate multiple string matching is compared against approximate string matching performed on each pattern separately. The results of the proposed algorithm are more promising when compared to performing approximate string matching separately on each pattern in the set. In the future, algorithms that search each pattern with different number of errors must be studied.

REFERENCES:

[1] Kambatla K, Kollias G, Kumar V, Grama A. Trends in big data analytics. Journal of Parallel and Distributed Computing. 2014 Jul 1; 74(7):2561-73.

[2] Hakak SI, Kamsin A, Shivakumara P, Gilkar GA, Khan WZ, Imran M. Exact String Matching Algorithms: Survey, Issues, and Future Research Directions. IEEE Access. 2019 Apr 30; 7:69614-37.

[3] Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. "Fast pattern matching in strings." *SIAM journal on computing* 6.2 (1977): 323-350.

[4] Boyer, Robert S., and J. Strother Moore. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

[5] Horspool, R. Nigel. "Practical fast searching in strings." *Software: Practice and Experience* 10.6 (1980): 501-506.

[6] Crochemore, Maxime, et al. "Speeding up two string-matching algorithms." *Algorithmica* 12.4-5 (1994): 247-267.

[7] El-Mabrouk, Nadia, and Maxime Crochemore. "Boyer-Moore strategy to efficient approximate string matching." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 1996.

[8] Tarhio, Jorma, and Esko Ukkonen. "Approximate boyer-moore string matching." *SIAM Journal on Computing* 22.2 (1993): 243-260.

[9] Liu, Zheng, et al. "A fast algorithm for approximate string matching on gene

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

sequences." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 2005.

[10] Salmela, Leena, Jorma Tarhio, and Petri Kalsi. "Approximate Boyer-Moore string matching for small alphabets." *Algorithmica* 58.3 (2010): 591-609.

[A] Navarro G. A guided tour to approximate string matching. ACM computing surveys (CSUR). 2001 Mar 1;33(1):31-88.

[11] Crochemore, Maxime, et al. "Fast practical multi-pattern matching." *Information Processing Letters* 71.3-4 (1999): 107-113.

[12] Crochemore, Maxime, and Thierry Lecroq. "Multiple String Matching." (2015).

[13] Baeza-Yates, Ricardo, and Gonzalo Navarro. "Multiple approximate string matching." *Algorithms and Data Structures*. Springer Berlin Heidelberg, 1997. 174-184.

[14] Muth, Robert, and Udi Manber. "Approximate multiple string search." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 1996.

[15] Baeza-Yates, Ricardo, and Gonzalo Navarro. "New and faster filters for multiple approximate string matching." *Random Structures & Algorithms* 20.1 (2002): 23-49.

[16] Fredriksson, Kimmo, and Gonzalo Navarro. "Average-optimal single and multiple approximate string matching." *Journal of Experimental Algorithmics (JEA)* 9 (2004):1-4.

[17] Baeza-Yates RA, Gonnet GH. Fast string matching with mismatches. Inf. Comput.. 1994 Feb 1;108(2):187-99.