BAJOPMAS JOURNAL 16(2): 26 – 38
Printed in Bayero University, Kano - Nigeria.

EFFICIENT COMPRESSED STRUCTURAL PATTERN MATCHING OF RNA STRUCTURE USING WAVELET TREE AND BIT-PARALLELISM

Muhammad, Y. M.
Department of Computing, Faculty of Computing, Bayero University Kano, Nigeria
mymuhammad.cs@buk.edu.ng

ABSTRACT
Traditional string pattern matching is well defined as finding similarities between two strings in either in compressed or uncompressed form. These techniques have good application in natural language processing, plagiarism detection, information retrieval etc. However, these algorithms cannot be directly applied to Biological string sequences without modification, this due to the inherent characteristics of these types of strings. For example, in RNA sequences, each string (nucleotide) is complementary to one another as such traditional matching algorithms will yield false results i.e. two physically similar sequences might not be structurally similar. To address this problem, this research proposes a Structural matching (s-matching) algorithm to achieve finding true similarity between RNA sequences. It is technique that finds true similarities between RNA string sequences taking into cognizant the complementary base pairing characteristics of the RNA nucleotides. Also, due to the exponential increased in the size of uncharacterized RNA sequences traditional matching algorithms yielded low optimal performances. To further enhance the optimality of the results, this research uses data compression techniques to compress these sequences before applying the structural matching algorithm. The proposed algorithms will help in finding the structure of the newly discovered RNA sequences that is critical for its functional inference with more optimal performances.

Key words: Biological string sequences, inherent characteristics, S-matching and RNA

1.0 INTRODUCTION
Compressed string pattern matching is defined as finding all the occurrences of an uncompressed string pattern P in a given compressed text string T_c using an arbitrary compression scheme. The aim is to minimize the search time and space optimization. This is applicable in bioinformatics where we wish to detect similarities between a new RNA sequence and RNA sequences in a database from which an ancestral relationship between them can be inferred. Myriad of algorithms were proposed for compressed pattern matching problem i.e. In [1] addressed the parameterized matching (p-matching) problem in term of fully compressed run length encoding and later proposed an alternative in [2] in $O(r_p \times r_T)$ time where r_p and r_T are run numbers in encoding for P and T_c. Compressed p-matching in [3] using the concept of parameterized compression called partial decomposition method. Navarro in [4] perform a compressed exact matching on LZ compression scheme in $O(\min\{n, m_{nc}\} + n_{occ})$ time. A compressed approximate matching problem was addressed by [5] in $O(mk_{nc}+n_{occ})$ time, where k is the number of permitted edits. For biological sequences, structural similarity gives better inference than sequence similarity. Though, there are few or no researches on compressed structural matching (s-matching), compressed parameterized matching can be extended to structural matching (s-matching). This can formalized as follows.

An s-string composed of constant and parameter symbols from finite sets $\sum$ and $\prod$ respectively where each symbol in $\prod$ may be a complement of one another. The idea of matching these symbols and complements is referred to as s-matching problem. This problem was initially addressed by indexing approach using data structures such as in [6] using data structures such as structural suffix tree (s-suffix tree) and structural suffix array with longest common prefix array [3]. However, these approaches suffered from the problem of space and time complexities.

In this paper we have proposed two new compressed pattern matching algorithms to solve exact single pattern s-matching problem on RNA structure. The first algorithm uses an encoder technique called word based tagged code (WTBC) that allows pattern (word) searching directly from the compressed text string and wavelet tree data structure that facilitate the searching process by reducing the cost of explicit maintenance of the text. While the second algorithm uses the same encoder technique (WBTC) and Bit-parallelism technique with n-gram to enable fast searching.

Other sections of the paper are organized as follows. In section 2 we present the Preliminaries. Section 3 presents the related concepts, the proposed algorithms were presented in Section 4, Section 5 presents the results analysis and finally, discussion and conclusion were presented in Section 6.

Our Contribution

Two algorithms were proposed in this research that will aid characterizing newly discovered RNA sequences for its functional inference with more optimal performances (i.e. Addresses the problems of false matches and low optimal performances that are associated with other algorithms).

1. *Preliminaries*

A string T = T[1, 2, … n] of length n is a production over an alphabet $\sum$. The following notations are used. Ti refers to ith symbol of T, T[i, j] a substring of T[i, i+1, …j] and T[i, n] the ith suffix of T[i, i+1,…n]. Let $\sum$ and $\prod$ be a two finite sets of constant and parameter symbols such that $(\sum * \prod) = ^{\varnothing\varnothing}$ with $ attached to the string as a terminal symbol.

A p-string is a production T of length n from $(\sum * \prod)*\$$. For example:
Two p-strings S and T are said to be parameterized matched (p-matched) if and only if, constant symbols from S and T exactly matched with a bijection between parameter symbols. This problem can be easily solved with the help of previous encoding (prev-encoding) defined as:

Definition 1: Prev-encoding. Given a p-strings S, then the prev-encoding(S) encodes all constant/terminal symbols with the same symbol and each parameter symbol with the distance from its previous occurrence. As defined below.

$$\text{Prev-encoding(S)} = \begin{cases} Ti & if\ Ti\ is\ a\ constant\ symbol \\ 0 & if\ it\ is\ the\ first\ occurance\ of\ Ti \\ i - \max[j \mid Ti = Tj], 1 \le j \le i & if\ otherwise \end{cases}$$

Example 1: Let the strings S= BxAzzywv, T= BwAyyzxv and U= BwAyyxzv be p-strings over two finite sets of fixed symbols $\sum$ = {A, B} and parameter symbols $\prod$ = {v, w, x, y, z} then the *prev(*S) = B0A01000, *prev* (T) = B0A01000 and prev (U) = B0A01000.

Definition 2: Complement encoding (Compl-encoding). Given a p-string T then compl-encode(T) encodes constant symbols from ∑ with the same symbol and parameter symbol from ∏ to the distance of its previous occurance or complementary symbol as formally defined.

$$\text{Compl-encoding (T)} = \begin{cases} T_i, & if\ T_i = C \in \sum \\ 0, & if\ T_i \in \pi\ \ and\ T_i \neq compl(T_j)\ for\ 1 \leq j \leq i \\ i - \max[j | Ti = compl(Tj)], & if\ otherwise \end{cases}$$

Example 2: Given two finite sets of fixed symbols ∑ = {A, B} and parameter symbols ∏ = {v, w, x, y, z} with complementary parameter mapping 𝕀 = {(v↔v), (w↔ x), (y↔ z)} and let the strings S=AxBzzywv, T=AwByyzxv, and U=AwByyxzv be p-strings, then the compl*(*S) = A0B00150, compl (T) = A0B00150 and compl (U) = A0B00420.

Definition 3: Structural encoding (S-encoding). Given the prev(S) and Compl(S) of an s-string S then the s-encoding(S) encode constant symbols say C with same symbol C and encodes parameter symbols Max[$\pi_1 \in \text{prev(S)}, \pi_2 \in$ compl(S)] as formally defined.

$$\text{S-encode (T)} = \begin{cases} T_i, & if\ T_i = C \in \sum \\ \dfrac{prev(T_i)}{compl(T_i)} & \begin{matrix} if\ prev(T_i) \geq compl(Ti) \\ if\ compl(T_i) > prev(T_i) \end{matrix} \\ 0, & otherwise \end{cases}$$

Example 3: Given two finite sets of fixed symbols ∑ = {A, B} and parameter symbols ∏ = {v, w, x, y, z} with complementary parameter mapping 𝕀= {(v↔v), (w↔ x), (y↔ z)}. Let the strings S=BxAzzywv, T=BwAyyzxv, and U=BwAyyxzv be s-strings and *prev(*S) = B0A01000, *prev* (T) = B0A01000, prev (U) = B0A01000, compl*(*S) = A0B00150, compl (T) = A0B00150, compl (U) = A0B00420 then the s-encode*(*S) = B0A01**15**0, s-encode(T) = B0A01**15**0 and s-encode(U) = B0A01**42**0.

2. RELATED CONCEPT

2.1 *Data Compression Techniques*

As a result of geometric increase in volume of biological data, storing, processing and retrieval of such becomes a costly and cumbersome affair, this lead to higher demand for larger storage and efficient techniques to handle such huge volume of data. An example of such biological data is RNA structural sequence represented as string sequence formed from the set {A, C, G, U} each representing a nucleotide base Adenine, Cytosine, Guanine and Uracil that form the RNA molecule. During the process of forming RNA structure (RNA folding) nucleotides (A, U) and (G, U) form what is called complementary base pairs. One of the techniques used to efficiently address the problem of larger storage requirement, optimizing the processing and retrieval time is data compression, which is a process of representing a data in a form that will occupy less space. The idea behind data compression is minimizing storage requirement and making data processing and retrieval easier. This has a good application in Bioinformatics where pattern matching technique can be used to find sequence / or structural similarity between RNA query sequence and RNA sequences in the database. Two compression techniques are used namely, lossless compression and lossy compression. In lossless compression both input and output data are identical as no single bit is lost during compression e.g. Run length encoding, Huffman coding, Lempel ziv etc. In lossy compression the output data cannot produce exactly the same input data e.g. Jpeg, Mp3 etc.

1. ***Huffman Coding***

This is the first data compression methods proposed by [8] that uses binary code for compression. This was later extended to use a sequence of byte assigned to each text word. The compression can either be bit oriented or byte (word) oriented. In bit oriented compression, statistical analysis of the data is used to generate variable length code for the symbol. This method is greedy toward the symbols frequencies where minimum number of bit are used to represent high frequency symbols and higher number of bits are used to represent low frequency symbols. Direct searching in a compressed text is faster with this method. In word oriented Huffman compression, words are used as basic element of the vocabulary. This method is based on space less word model where the word is encoded as it is if followed by a space else the word is encode first and then the separator. So, the possibility of false matches occurrences are easily avoided by disallowing direct searching, hence this method perform decoding more efficiently and gives better compression than bit wise method. For example: Consider the RNA sequence T = AUAGCAGCU over the alphabets ∑ = {A, C, G, U}

Table 1: Huffman coding vocabulary

Symbols	A	G	C	U
Frequency	3	2	2	2
Huffman code	00	01	10	11

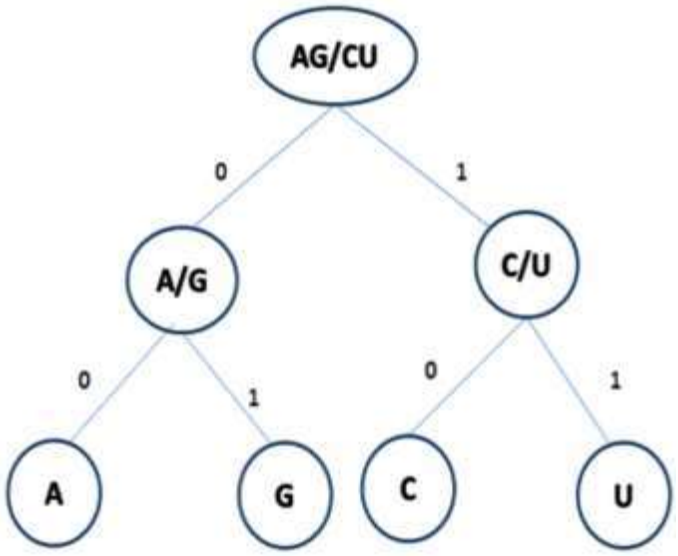

Figure 1: Huffman Tree

The compressed text Tc of T = 000000010110101111 = 18 bits
Number of ASCII characters in T = 9 × 8 = 72 bits
Compression ratio = (18/72)*100 = 25%

2. ***Word Based Tagged Code***

This is an efficient compression technique for dynamic data proposed in [9] to the problem false matches associated with other text compression techniques like, Huffman coding. This makes more versatile in areas of natural language processing, bioinformatics etc. In this method words are treated as a basic unit of compression. It generates word code using two combinations of (00, 01, 10 and 11). The algorithm starts by passing the source text then

generating the vocabulary and saving it in decreasing order of frequencies. For example: We consider the RNA sequence T= AGAGCAUCU of length m over the alphabet ∑ = {A, C, G, U}, depending on the length of the pattern we split the text into words each of pattern's length. Supposed the length of the pattern is 2 for this example. The Vocabulary for T and WBTC codes are given in the table and Table respectively.

Table 2: A & B: WBTC vocabulary tables

A

Words	**AG**	**CA**	**UC**	**U**
frequency	2	1	1	1

B

Words	**AG**	**CA**	**UC**	**U**
frequency	2	1	1	1
WBTC	01	10	0001	1101

The compressed text Tc of T = 01011000011101= 14 bits

Number of ASCII characters in the text = 9x8 = 72 bits

The compression ratio = (14/72)*100 =19.44%

3.0 WAVELET TREE (WT)

An indexed data structure introduced in [10] that explicitly reduces the cost of maintaining text by making it easier to track and retrieve text. It implement three basic text operations i.e. finding the frequencies of a symbol in the given text (COUNT), finding an occurrence of a symbol in the given text (LOCATE) and obtaining a symbol at any position of the source text (DISPLAY) using RANK and SELECT operation [11]. WT can be formed from a compressed text as a binary tree where symbols are present at leaf node as in [12] and [13].

Example: Consider the RNA sequence T=AGAGCAUCU over the alphabet ∑ = {A, C, G, U}. We construct the wavelet tree as in **Figure 2.**

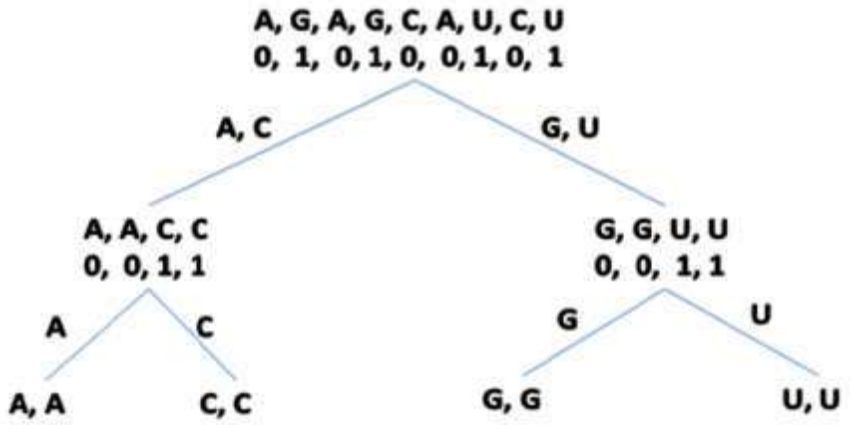

Figure 2: Wavelet Tree

3.1 Bit-Parallelism

This was first introduced in [14] is a technique that performs paralleled bit operations in a computer word. With this many value can be stored in a word and be updated in single operation. It facilitates searching by reducing the number of operations by a factor of say W, where W is the computer word size usually 32 or 64 bits. Many articles are published on using bit-parallelism to solve pattern matching problem. The bit parallel operations such as AND, OR are used to solve sequence alignment problem on DNA/RNA sequences as Introduced in [18].

Standard shift-or algorithm was one of the early bit-parallel algorithms proposed for exact string pattern matching. It consists of two stages, the pre-processing stage and searching stage. At the pre-processing stage, algorithm:

1. Builds a non-deterministic finite automaton (NFA) for a pattern P[0…m-1] of size m
2. The automaton is constructed with states 0(initial state) to m (final or accepting state).
3. for a character P_i , where i = 0…m-1,there exist a transition from a state i to state i+1
4. For every $c \in \sum$ there is a transition from initial state back to it thereby making it non-deterministic process.
5. Forms a table with bit mask entry say $c \in \sum$.
6. The bit mask B[c] has the i^{th} bit set to 0 if and only if P[i] = c for $0 \leq i \geq m-1$ i.e. If bit i in B[c] is 0 then there exist a transition from state i to i+1with character c.

At searching stage:

1. The automaton presented in a bit vector D that encodes the state of operation.
2. i^{th} bit of the vector sets to 0 only when state i is active.
3. At initial stage all bits of the vector are set to 1
4. The state vector D is updated by $D \leftarrow (D<<1) \mid B[T[i]]$ where T[i] is the ith text.
5. When all the ith text characters are processed and the least bit of D is 0 then a match is encountered at the shift 1-m

3.2 Compressed Pattern Matching (CPM)

Pattern matching is task of finding all the occurrences of a string pattern inside large body of text. It is one of the good application areas of data compression. As a result of geometric increase in the volume of the daily generated data, data compression became highly desirable for its' efficient utilization.

Compressed pattern matching is a process of finding all occurrences of string pattern P (compressed or uncompressed) inside the body of large compressed text Tc. This technique can be applied on either text or image data, and has good application in areas like, plagiarism detection, signal processing, text mining, computational biology etc.

Myriad of algorithms have been proposed on compressed pattern matching to solve different pattern matching problem on both text and image data using different compression techniques such as methods using indexing approach [15], [16], [17], coding methods [18], [19], [20], [21], [22], [23] and [24].

However, these algorithms are associated with problem of false matches and their approaches are sometimes not suitable for the analysis of biological data because of its cumbersome nature [24]. Hence, this paper will propose an alternative approach that will help in addressing these inadequacies.

4.0 COMPRESSED S- MATCHING ON RNA STRUCTURE

In finding similarities between biological sequences, large sequences are easily handle by compressing it and performing a pattern matching between compressed sequence Tc of string T and a pattern P. RNA structural sequence is form from a sequence of four nucleotide bases represented with this string letters (A, C, G, U) where (A, U) and (G, C) are complementary to one another. However, this behavior of RNA molecule during the structure formation has made compressed s-matching problem not to be easily handled using the current compressed pattern matching techniques. Therefore, we can formally define the compressed s-matching problem as:

Definition 4: Let $\sum$ and $\prod$ be two finite sets of fixed and parameter symbols, also let $\mathbb{\Gamma}$ be a set of complementary base pairs (x_1, x_2) such that $x_1, x_1 \in \prod$. A production over $(\sum * \prod)^*$ is an s-string. The CSMP can be easily handled, given a compressed form T_c of an length **n** string **T** and an length **m** string pattern **P** such that, for each **i** an array s-encode[p_i] where $1 \leq i \leq m$ is formed either from prev-encode[pi] when prev-encode[pi] ≥ compl-encode[pi] or compl-encode[pi] otherwise where $(\sum \cap \prod) = \varnothing$. Then the pattern P and T_c are said to be s-matched at position i if s-encode(Tc[1…n-1]) = s-encode(P[1…m-1]), where $1 \leq i \leq n$.

In this paper we are proposing two algorithms for compressed s-matching on RNA structure. The first algorithm will proposed the use of WBTC compression technique for RNA structural sequence compression and the matching will be perform using the s-WT.

4.1 WBTC and WT Algorithm for RNA s-matching

WBTC is an efficient word encoding method that offers higher compression ratio. However, the efficiency of this coding method was hampered by the possibility of having a false matches, this is because it does not encode text in free prefix form. Therefore, the proposed algorithm will use s-WT to organize the WBTC coded text into a form that will eliminate the possibility of having the false matches thereby improving the efficiency search operation. The algorithm will runs at both pre-processing and matching (searching) phases.

The Pre-processing phase

At this stage, s-encoded words of a given RNA s-string sequence are compressed by assigning a WBTC and WT is constructed from the WBTC of the s-encoded words. During the pre-processing, the RNA sequence is grouped into words of length m (number of nucleotide base in RNA molecule). If any of the words of length is greater than the pattern m then, is grouped into m-gram because a pattern may s-match word's substrings of the s-string sequence as demonstrated in the example below:

Example 4: Let P=CAAC be a pattern to be search from a string sequence T over the alphabet $\sum$ = {A, C, G, U} and complementary mapping A↔C, G↔U for a word "AGUUGA" ∈ T. It is clear that the word length < m so, the word is splitted using 4-gram as: AGUU, GUUG and UUGA. Hence, the pattern p = CAAC s-matched the word substring GUUG based on the mapping.

After the words formation, s-encodings (as in **Example 3**) for each word is determine by taking the bit of higher value from either the prev-encoding (as in **Example 1**) or compl-encoding (as in **Example 2**) according to **Definition 3.** The s-encodings for each word together with their corresponding frequencies (arranged in decreasing order) are stored in the vocabulary table as demonstrated in the example below:

Example 5: Let T = AGAAGAGAAGUCCAUGG be an RNA string sequence build over an alphabet ∑ = {A, C, G, U} and complementary mappings A↔C, G↔U.

Here m=4, so we group the string text into words of length 4 each as: AGAA, GAGA, AGAA, G UCC, AUGG. Where, GAGAA is splitted as in example 4. The encodings are computed as given in the table below:

Table 3: Encodings

Word	**AGAA**	**GAGA**	**AGAA**	**GUCC**	**AUGG**
Prev-encoding	0021	0022	0021	0001	0001
Compl-encoding	0021	0022	0021	0021	0101
S-encoding	0021	0022	0021	0021	0101

Therefore, the s-encoded string sequence T = 0021 0022 0021 00210101
The vocabulary table for T is formed and WBTC is assigned for each word as in the table below:

Table 4: Vocabulary for T

Index	**0**	**1**	**2**
S-encoding	0021	0022	1010
Frequency	3	1	1
WBTC	01	10	0001

Therefore, the compressed from of string sequence T is Tc = 010101100001 after which the wavelet (WT) is constructed from WBTC of the s-encoded words.

Structural Wavelet tree (s-WT) construction

The proposed algorithm constructs a wavelet tree (WT) from the WBTC s-encoded words of the s-string sequence using the bits pairs {00, 01, 10, 11} where, {01, 10} are used as flags that marked the end of the coded word and {00, 11} as prefix of the flag bit. Each node of WT is an array of bits pair of a coded word where the root node is at level one (1) with the starting bits pairs for each coded word as in the original string sequence. While the bits pairs of all other coded words are at the level of its parent node + 1. The root node is assigned with 00 for the left pair and 11 for the right pair. The same process is done for all the remaining bits pairs of the coded words. To process a coded word, move from the node to its children and observe if bit pair of the child is 00 then move to left, if is 11 then move right. The coded word is processed when a flag pair 01 or 10 is encountered. The WT construction algorithm is given below.

Algorithm 1: s-WT construction algorithm

Let $W=w_1 \ldots w_n$ be a sequence of words and $SW = sw_1 \ldots sw_n$ be the corresponding s-encoded words.

For *i ← 1 to n* ***do*** *// n = number of words in W*
    $P_{i,1}$ *← fist bit of the s-encoded word* SW_i
    Root ← $P_{i,1}$
    Current-node ← root
    J ← 2 // move to the next pair
    While (not end of SWi) do
        $P_{i,j}$ *← next pair from SWI*
        If ($P_{i,j-1}$ *← {00}) then*
            Current-node ← left-child
        Else
            Current-node ← right-child
            Insert $P_{i,j}$ *into a current-node of WT*
    J← J+1
    End while
End for

Words	AGAA	GAGA	AGAA	GUCC	AUGG
S-encode	0021	0022	0021	0021	0101
Index	0	1	2	3	4
B0	01	10	01	01	00

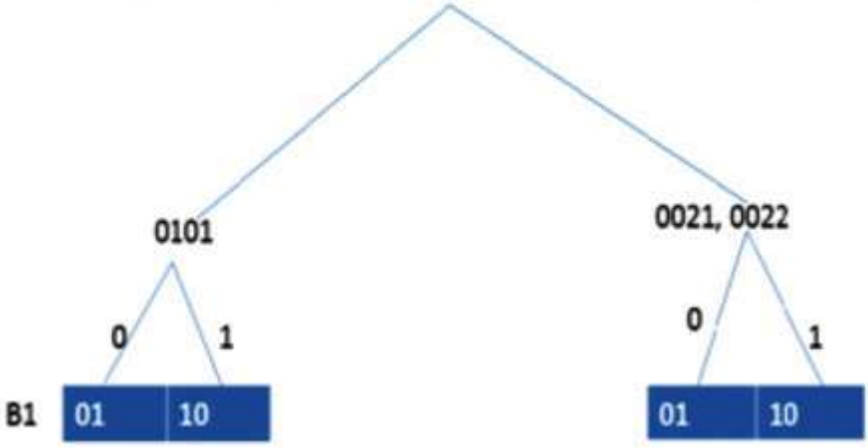

Figure 3: Structural Wavelet Tree (s-WT)

Searching phase (Compressed s-matching)

The s-WT constructed in the pre-processing phase will now be used to perform our s-matching on RNA structural string sequence i.e. finding the number and positions of all occurrences of a given pattern in a compressed RNA string sequence. Here the searching is implemented using two bit operations:

(i) Rank () = returns the number of occurrences of a pattern in the string sequence
(ii) Select () = returns the position at which a pattern occurs in the string sequence

Searching process is done as follows:

1. Find the pattern code from WBTC vocabulary table
2. Travers the WT to find a node with flag bits 01 or 10
3. Apply RANK () operation on the flag node to get the number of occurrences of a pattern
4. Apply SELECT () operation at flag node and traversing to root node to get the ith position of pattern occurrence.
5. Repeat 4 to find all occurrences of a pattern

Algorithm 2: s-WT algorithm for compressed s-matching

B_0 = root(T)
P_0 = get the 2 starting bits from WS
    i = 0
While *($P_i \neq\neq$ 01 and $P_i \neq\neq$10)* ***do*** *//locating the flag node*
    If (P_i == 00) then
        Bi+1 = left-child (B_i)
    Else
        Bi+1 = right-child (B_i)
    i = I + 1
    P_i = ith bit pair from WS
End while
NoOccurrence = $rank_{Pi}$ (B_i, $|B_i|$)
For *J = 1* ***to*** *No-Occurrence* ***do***
    indexNo = $select_{Pi}$ (B_i; j) *// index of jth occurrence*
    Level = i *// Level of the flag node*
    While *B_i = root (T)* ***do***
        Level = level – 1
        B_i = Parent (Bi+1)
        If *(B_{i+1} == left-child (B_i))* ***then***
            P_i = 00
        Else
            P_i = 11
        indexNo = $select_{Pi}$ (B_i, indexNo)
    End while
    Display the jth occurrence of a pattern at index
End for

The working of algorithm will be demonstrated using the example, supposed we want search for a pattern **P=AUGG** form the RNA string sequence in example 5. The algorithm extract the WBTC of P from the vocabulary table, here the WBTC for **P** is 0001. The s-WT is traversed to get the node with flag bit **01.** Since, code 0001 is of two pairs, the algorithm moves to the next level from the root. The 1st pair is 00 so the algorithm moves to the left child from the root to $\mathbf{B_1}$ containing the flag bit **01.** Operation $\mathbf{rank_{01}}$ (B_1, $|B_1|$) = 1 is performed. This indicates that the pattern P occurs only once in the sequence. To determine the position of its occurrence, the algorithm traversed the s-WT in the reversed order i.e. from $\mathbf{B_1}$ to $\mathbf{B_0}$ by performing the operation $\mathbf{select_{01}}$ (B_1, 1) = 1, this means that bit pair **01** occurs at 1st position at node $\mathbf{B_1}$. The algorithm moves one level up to $\mathbf{B_0}$ and performs the operation $\mathbf{select_{00}}$ (B_0, 1) = 4 which means the word pattern **P** occurs at position 4 of the word sequence.

4.2 Bit-parallel compressed s-matching of RNA structure

The algorithm uses the WBTC assigned to s-encoded words of s-string sequence for compression and Bit-parallel shift-or to perform single pattern s-matching of RNA structure. The process is also done at both pre-processing and searching (or s-matching) phases.

The Pre-processing phase

The algorithm first encodes a given s-string into prev-encoding (as **in Example 1**), compl-encoding (as in **Example 2**) and s-encoding (as in **Example 3**). The s-encoding of the words are assigned WBTC as in **Table 4.** The algorithm uses the WBTC to compute the bit vector B of the corresponding word to be search as $B[C] = ((B[C_1] \& 01^m << 1) \mid (B[C_2] \& 01^m))$, where $C= \{c_1, c_2\}$.

Searching phase (s-matching)

In this phase, a shift-or with 2-gram is used to perform an s-matching of a word from an s-sting sequence. Here, we assumed the length of WBTC to be **m** (m=2, since 2-gram is used) and the maximum number of bit per encoded word as m+1. The vector D was initially set to 1^{m+1} = 111 and updated as D = ((D<<2) | B[C]). After processing if the main significant bit of D is zero then a match exists however, this match may be a false match. To confirm whether a match is actual or false we used $D \& B[c_1, c_2] \neq 0^{m+1}$ i.e. the bits of the previous 2-gram of the s-string are checked, if the previous bits are 00 or 11 then the match is false and if are 01 or 10 then the match is actual.

Example 6: Supposed we intend to search for a word pattern **P =AGAA** from an s-string **T = AGAA GAGA GUCC AUGG.** At the pre-processing stage, the algorithm extract the WBTC of **P** from the vocabulary table as **01** and compressed s-string $\mathbf{T_c}$ **= 011001010001.** The bit mask is computed as:

D= 111
B[0] = 110, B[1] = 101
B[00] = ((B[0] & 011 << 1) | (B[0] & 011)) = (110 & 011<<1) | (110 & 011) = 100 |010 = 110
B[01] = ((B[0] & 011 << 1) | (B[1] & 011)) = (110 & 011<<1) | (101 & 011) = 100 |001 = 101
B[10] = ((B[1] & 011 << 1) | (B[0] & 011)) = (101 & 011<<1) | (110 & 011) = 010 | 010 = 010
B[11] = ((B[1] & 011 << 1) | (B[1] & 011)) = (101 & 011<<1) | (101 & 011) = 010 | 001= 011

Searching

Input 01: D = (D<<2) | B[01] = 100 | 101 = 101, since the bit of D is zero there may be a match, we for check,
D & B[01] = 101 & 000 = 000, valid match exist
Input 10: D = (D << 2) |B[10] = 100 |010 = 110, **No match**
Input 01: D = (D << 2) | B[01] = 100 | 101 =101 , D & B[10] = 101 &010 =000, **valid match exis**t
Input 01: D = (D << 2) | B[01] = 100 |101 = 101, D & B[01] = 101 & 101 ≠ 000, **false match exist**
Input 00: D = (D << 2) | B[00] = 100 | 110 =110, **No match**
Input 01: D = (D << 2) | B[01] = 100 | 101 = 101, D & B[00] = 101 & 110 ≠000**, false match exist**

The total number of valid matches = **2**

5.0 CONCLUSION AND FUTURE WORK

The first algorithm word based tag code and wavelet tree uses self-indexed data structure which reduces the cost of explicit maintenance of the text thereby making the searching procedure easier. It also requires less bit-app operations to search because the wave let tree uses pair of bits to encode the text; this reduces the height of the WT. The use of bit parallelism techniques produces correct number of matches hence; problem of false matches is eliminated. However, searching for multiple patterns and addressing the problems of approximate pattern in structural matching can be considered in the future.

REFERENCES

[1]. A. Apostolico, P.L. Erdős, M. Lewenstein, Parameterized matching with mismatches, J. Discrete Algorithms 5(1) (2007) 135–140.

[2]. A. Apostolico, P. L. Erdos, and A. Juttner, Parameterized searching with mismatches for run-length encoded strings, Theoretical Computer Science (in press), 2012.

[3]. Beal, R. and Adjeroh, D. A. (2013). Compressed parameterized pattern Matching. IEEE Data Compression Conference, pp. 461-470.

[4]. Navarro, G., Baeza-Yates, Ricardo., Sutinen, Erkki and Tarhio, Jorma. (2001). Indexing Methods for Approximate String Matching. *IEEE Data Engineering Bulletin* 24(4):19-27.

[5]. Kärkkäinen, Juha, Sanders, Peter (2003). Simple linear work suffix array construction. Automata, languages and programming. Lecture Notes in Computer Science. 2719. p. 943. doi:10.1007/3-540-45061-0_73 (https://doi.org/10.1007%2F3-540-45061-0_73). ISBN 978-3-540-40493-4.

[6]. T. Shibuya. Generalization of a suffix tree for RNA structural pattern matching. Algorithmica 39(1) (2004) 1– 9.

[7]. R. Beal, D. Adjeroh. Theoretical Computer Science 592 (2015) 59–71

[8]. Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. In proceedings of the IRE, 40(9), pp. 1098 -1101.

[9]. Gupta, A. and Agarwal, S. (2008). A Scheme That Facilitates Searching and Partial Decompression of Textual Documents. Intl. Journal of Advanced Computer Engineering, 1(2).

[10]. Grossi, R., Gupta, A. and Vitter, J. (2003). High-order Entropy Compressed Text Indexes. In Proceedings of 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 03), pp. 841–850.

[11]. Claude, F. and Navarro, G. (2008). Practical rank/select Queries over Arbitrary Sequences. In: Proc. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 5280, pp.176–187.

[12]. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G. (2007). Compressed Representations of Sequences and Full-text Indexes. ACM Transactions on Algorithms (TALG) 3(2): article, pp.20-24.

[13]. Navarro, G. and akinen, V. M. (2007). Compressed Full-text Indexes". ACM Comp. Survey, 39(1): article 2.

[14]. Baeza-Yates, R. (1989). Efficient Text Searching. Ph. D. Thesis, Department of Computer Science, University of Waterloo.

[15]. Grossi, R., Gupta, A. and Vitter, J. (2006). When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications. ACM- transactions on algorithms, 2(4):611-639.

[16]. Navarro, Gonzalo. (2014).Wavelet Trees for All. Journal of Discrete Algorithms 25:220.

[17]. Russo, Luís., Navarro, Gonzalo and Oliveira, Arlindo. (2011). Fully-Compressed SuffixTrees. *ACM Transactions on Algorithms* 7(4): article 53.

[18]. Brisaboa, Nieves., Iglesias, Eva., Navarro, Gonzalo and Paramá, José. (2003). An Efficient Compression Code for Text Databases. *Proc. ECIR'03*, pages 468-481. LNCS 2633.

[19]. Brisaboa, N. R., Cillero, Y., Farina, A., Ladra, S. and Pedreira, O. (2007). A New Approach for Document Indexing Using Wavelet Trees. Database and Expert Systems Applications, DEXA '07. 18th International Workshop, pp. 69-73.

[20]. Avendaño, Carlos., Feregrino, Claudia and Navarro, Gonzalo. (2005). Approximate Searching on Compressed Text. *Proc. CONIELECOMP'05*, pages 258-261. IEEE CS Press.

[21]. Gupta, A. and Agarwal, S. (2011). Searching a pattern in Compressed DNA Sequences. International Journal of Bioinformatics Research and Applications, vol. 7, no. 2, pp. 115-129.

[22]. Garg, R., Prasad, R., and Agarwal, S. (2014). Fast Parameterized Word Matching On Compressed Text. IEEE International Conference on ICCCT, MNNIT, Allahabad, India, pp. 317-321.

[23]. Khetan, R., Agarwal, S. and Prasad, R. (2016). An Efficient Approach towards Compressed Parameterized Word Matching Using Wavelet Tree. Journal of Information and Optimization Sciences, Vol. 37, no. 2, pp. 285-301.

[24]. Surya P. M. (2017). Enhanced Algorithms for Compressed Pattern Matching using Wavelet Tree and Bit-parallelism. A Thesis Submitted in Partial Fulfillment of the Requirement for the Award of PhD in Computer science & information technology faculty of engineering and technology Sam Higginbottom University of agriculture, Technology and sciences. Allahabad-211007