ISSN 2515-8260 Volume 07, Issue 06, 2020

Enhanced Tunneled Burrow Wheeler Transform to reduce decoding time with minimum space consumption

Ranjitha S[1], Robert L[2]

[1]*Research Scholar, Department of Computer Science, Government Arts College (Autonomous), Coimbatore, Tamilnadu, India*
[2]*Associate Professor, Department of Computer Science, Government Arts College (Autonomous), Coimbatore, Tamilnadu, India*

E-mail: ranjithabiju@gmail.com[1] , robertatgac@gmail.com[2]

Abstract:DNA or genomic sequences compression and indexing using the standard algorithms are facing a high complexity as massive datasets grow rapidly.To avoid this problem, a Tunneled Run-Length Encoded (RLE) Burrows-Wheeler Transform (BWT)-based encoding with Improved Index (TBWT-II) algorithm has been proposed that uses Text-Label index (TL_{BW}-index) for counting and discovering the labeled patterns. However, the reduction on global space consumption of the TL_{BW}-index was not effective. Also, the classic MTF in TBWT-II has a specific local property that can be leveraged during encoding time and the decoded i^{th} character was a series function of the decoded values of prior characters. Therefore in this article, an Enhanced TBWT-II (ETBWT-II) algorithm is proposed to effectively reduce the global space consumption of TL_{BW}-index. The major goal of this algorithm is to avoid the need of local searching capabilities within the compressed database and minimize the space consumption during retrieval of characters. As a result, a locally-decodable Move-To-Front (MTF) encoding is used instead of standard MTF in TBWT-II for reducing the decoding time of a single character with the minimum space consumption. Finally, the experimental results on SCOPe 1.67 dataset show the performance efficiency of proposed ETBWT-II algorithm compared to the existing compression algorithms.

Keywords:DNA sequence compression, Text indexing, BWT, TBWT-II, Move-To-Front encoding

1. INTRODUCTION

Normally, DNA sequences are structured by using a group of four bases, viz. A, G, T and C. Every set of three symbols like TGC, ACT, and so on in the DNA sequences are called as codons. Also, the genomes size can be varied. Typically, the nucleotide sequences are accumulated by some large archival databases like GenBank at the National Center for Biotechnology Information (NCBI) [1], the DNA Data Bank of Japan (DDBJ) [2], the Short Read Archive (SRA) [3], etc.By using such databases, the sequenced data are classified, preserved and assigned. These databases size has been two or three percentage higher in recent years. As a result, most of the conventional compression algorithms are not adopted to compress the DNA sequences in an efficient manner. To avoid this problem, many advanced algorithms have been suggested for DNA sequences compression.

ISSN 2515-8260 Volume 07, Issue 06, 2020

Among many compression algorithms, BWT algorithm [4] is mostly favoured rather than other algorithms for compressing DNA sequences because of its efficiency on both compression and indexing of DNA sequences.On the other side, the computational cost of this algorithm is high while compressing high amount of DNA sequences. Therefore, a TBWT algorithm [5] is introduced that minimizes the cost of RLE-BWT by means of its combinatorial properties. This algorithm is executed only on width-maximal run-blocks with higher height and width.

The encoding of a TBWT needs the residual BWT with the bit vectors. In contrary, the TBWT inversion has no ability for perfectly matching the tunnel start or end to the respective tunnel because of the interfaces of start or end intervals of blocks while treating critical collisions.As a result, a TBWT-II algorithm [6] is suggested that simplifies the TBWT by efficiently indexing the labeled texts. This TBWT-II considers a novel index for a text with non-overlapping labels that can accumulate the text in a BWT. The label sequence is considered in the format of TL_{BW}-index that facilitates the effective text-label queries for counting and discovering the labeled patterns. These patterns can be utilized on any labeled DNA sequences. But, the global space consumption of the TL_{BW}-index is reduced only at the minimal level. Also, the classic MTF in TBWT-II has a specific local property that can be leveraged during encoding time and the decoded i^{th} character is a series function of the decoded values of prior characters.

Hence in this article, an ETBWT-II algorithm is proposed for further reducing the global space consumption in an effective way. The main aim of ETBWT-II algorithm is to deal with the requirement of local searching capabilities in the compressed database and reduce the space consumption during retrieval of each character. For this purpose, a locally-decodable MTF method is applied in this ETBWT-II algorithm instead of classical MTF in TBWT-II. Based on this method, the decoding time of a single character of the actual text and the space redundancy are minimized.

The rest of the sections are prepared as follows: Section II surveys the related works on DNA sequences compression algorithms. Section III describes the ETBWT-II algorithm whereas Section IV presents its efficacy. Also, Section V summarizes the entire discussion.

2. RELATED WORKS

Sardaraz et al. [7] proposed a DNA sequence compression algorithm, namely SeqCompress that copes with the space complexity of biological sequences. This algorithm was based on the lossless data compression and the statistical model with an arithmetic coding was used for compressing DNA sequences. However, the decompression time was high.

Kimura & Koike [8] proposed BWT of reads with three basic functionalities for detecting genomic rearrangements. Initially, breakpoints regions were detected from the discordant pairs based on the conjugate gradient method. Then, the reads partially matching the breakpoints regions were collected and identified as the branching points among the collected reads. But, it needs an improved compression technique to compress the large-scale datasets.

Eric et al. [9] proposed an optimal seed based compression algorithm using a substitution mechanism for DNA sequences. In this algorithm, the repetition structures were exploited by generating the offline dictionary which has all such repeats with the details of mismatches. But, the compression ratio was not effective and also the computational complexity was high.

Sandhya et al. [10] proposed a novel two-stage algorithm by combining the characteristics of both Lempel-Ziv-Welch (LZW) and Huffman coding algorithms. Initially, the input sequence was given to the LZW algorithm for obtaining the coded output for compressed sequence.

ISSN 2515-8260 Volume 07, Issue 06, 2020

After that, it was converted into the binary form and the likelihood of symbols was computed. These likelihoods were assigned in decreasing manner and given as input to the Huffman algorithm. Moreover, Huffman encoding was achieved and the compressed sequence was acquired. However, the complexity was high and ineffective for more number of sequences.

Saada& Zhang [11] proposed an algorithm for compressing the DNA sequences on the basis of binary representations of DNA sequences. Initially, a new method was proposed for compressing the DNA sequence and converting it into the binary representation. After that, the resulting DNA was compressed by using the extended-ASCII encoding. Nonetheless, compression ratio was less.

Punitha&Murugan [12] proposed a novel algorithm, namely compressBest for compressing the DNA sequences. This algorithm was based on the utilization of 2-bits encoding mechanism includes split into segments, appropriate matching and the decompression matching. The 2-bit encoding mechanism was performed while bases were distributed randomly. On the other hand, the compression ratio was less than the lossless compression algorithm.

3. PROPOSED METHODOLOGY

In this section, the proposed ETBWT-II algorithm is briefly described for compressing and indexing the DNA sequences.Consider the alphabet $\Sigma = \{c_1, c_2, \dots, c_{|\Sigma|}\}$ where $c_1 < c_2 < \cdots < c_{|\Sigma|}$ according to the lexicographical ordering on Σ and $S = (a_1, a_2, \dots, a_{|\Sigma|})$is the MTF stack with a_1 at the top and $a_{|\Sigma|}$ at the bottom. For $j \in [|\Sigma|]$, consider $S[j]$ is the character at location j in S, initiating from the top. Set a string $x = (x_1, x_2, \dots, x_n) \in \Sigma^n$ and $m = MTF(x) = \in (m_1, m_2, \dots, m_n) \in \{0,1, \dots, |\Sigma| - 1\}^n$ is the MTF encoding of x with the primary MTF stack$S_0 \coloneqq (c_1, c_2, \dots, c_{|\Sigma|})$.

Given a MTF stack $S = (a_1, a_2, \dots, a_{|\Sigma|})$ and a permutation $\pi \in S_{|\Sigma|}$, consider $S' = \pi \circ S$ is the stack such that $S'[\pi(j)] = S[j] = a_j$ for all $j \in [|\Sigma|]$. Also, S is associated with the permutation $\pi(S)$ which transforms the primary stack S_0 to S, i.e., $S = \pi(S) \circ S_0$. For $i \in [n]$, consider $S_i$is the stack induced by simulating the MTF decoder on $m[1: i]$, initiating fromS_0. As well, S_i is the stack induced by $MTF(x[1: i])$, i.e., the stack after encoding the initial i characters of x, initiating fromS_0. For $0 \le i < j \le n$, consider $\pi_{i,j} \in S_{|\Sigma|}$ is the common permutation induced by simulating the MTF decoder on $m[i + 1: j]$, initiating fromS_i.

At first, a single B-tree $\mathcal{T}$is built over the entire MTF encoding $m = MTF(x) \in \{0,1, \dots, |\Sigma| - 1\}^n$ which supports local decoding queries. Assume $B \ge 2$ is the branching factor. Each node v is augmented with the permutation$\varphi_\pi(v) = \pi_{i-1,i}$. Assume v is an internal node with its children being $v_1, v_2, \dots, v_B$ in order from left to right. After that, v is augmented with the composition of permutations of its children, i.e.,

$$\varphi_\pi(v) = \varphi_\pi(v_B) \circ \varphi_\pi(v_{B-1}) \circ \dots \circ \varphi_\pi(v_1) \quad (1)$$

It is observed that the node v whose subtree $\mathcal{T}_v$ is constructed over the sub-array $m[i + 1: j]$ is augmented with the value$\varphi_\pi(v) = \pi_{i,j}$. The query algorithm maintains the MTF stack S which is initialized for identity stack S_0 at the starting of the array. Consider $i \in [n]$ is the query index. The algorithm traverses down the tree, updating S at each level. It maintains the invariant that whenever it enters a node v whose sub-tree includes $m[j + 1: k]$, it updates S to the true stack S_j before the starting of $m[j + 1: k]$.

To maintain this invariant recursively, the base is taken at the root whose sub-tree has the entire array m. As a result, the query algorithm starts $S = S_0$, which corresponds to the true

ISSN 2515-8260 Volume 07, Issue 06, 2020

primary MTF stack. Assume v is the node at depth d whose sub-tree $\mathcal{T}_v$includes $m[j+1:k]$. If the query algorithm has entered v and S is the true MTF stack S_j. By hypothesis, $j+1 \leq i \leq k$, consider $v_1, v_2, \dots, v_B$ is the children of v in order from left to right and v_{β^*} is the child of v whose sub-tree encompasses i. After that, S is updated as:

$S \leftarrow \varphi_\pi(v_{\beta^*-1}) \circ \varphi_\pi(v_{\beta^*-2}) \circ \dots \circ \varphi_\pi(v_1) \circ S$ (2)

It indicates the update rule which maintains the invariant at the node at depth $d+1$, considering the invariant is maintained at a node at depth d. Therefore, the proof that the invariant is maintained follows by induction on d. Finally, the algorithm reaches the leaf node corresponding tom_i. At this moment, the MTF stack S is the true stackS_{i-1}. Thus, it reports $x_i = S[m_i]$ and run time is $t = O(\log_B n)$.

For simplifying, the update rule (2) is stated in terms of forward compositions of permutations$\pi_{i,j}$. Practically, if$\beta^* > {}^{B}/_{2}$, one can identically update S by initiating from $\varphi_v \circ S$ and composing the inverse permutations $\varphi_\pi^{-1}(v_\beta)$ for $\beta \geq \beta^*$:

$S \leftarrow \varphi_\pi^{-1}(v_{\beta^*}) \circ \varphi_\pi^{-1}(v_{\beta^*+1}) \circ \dots \circ \varphi_\pi^{-1}(v_\beta) \circ \varphi_\pi(v) \circ S$ (3)

To support rank queries under the MTF encoding, consider v is an internal node with children$v_1, v_2, \dots, v_B$.Set a character$c_\sigma \in \Sigma$.Generally, the MTF stack at the starting of the sub-array rooted at $\mathcal{T}_v$ is varied from the MTF stack at the starting of the sub-array $\mathcal{T}_{v\beta}$ rooted at each child$v_\beta, \beta > 1$. Therefore, $\varphi_{rk}(v, \sigma)$ is defined in terms of the values of its children by including the entry of the vector $\varphi_{rk}(v_\beta)$ which corresponds to c_σ, for each $\beta \in [B]$. For $\beta \in [B]$, the true MTF stack at the beginning of the sub-array rooted atv_β, considering the MTF stack at the beginning of the sub-array rooted at $v$is S_0 is given by Eq. (2). Thus,

$\varphi_{rk}(v, \sigma) = \sum_{\beta=1}^{B} \varphi_{rk}\left(v_\beta, \varphi_\pi(v_{\beta-1}) \circ \varphi_\pi(v_{\beta-2}) \circ \dots \circ \varphi_\pi(v_1)(\sigma)\right)$ (4)

Consider $\Phi_{rk} = \{0,1, \dots, n\}^{|\Sigma|}$ and each node v is augmented with$\varphi_{rk}(v) \in \Phi_{rk}$. Since the permutation $\varphi_\pi(v)$ is encoded, the value at each internal node is a function of the values of its children and thus this is a legitimate B-tree.

The query algorithm, given$(c_\sigma, i) \in \Sigma \times [n]$, initializes a rank counter $rk = 0$ and traverses similar root-to-leaf path as before. Set an internal node v with children $v_1, v_2, \dots, v_B$ in its path. Consider $\beta^* \in [B]$ is such that the sub-array rooted at v_{β^*} has the index i. The algorithm updates rk as:

$rk \leftarrow rk + \sum_{\beta=1}^{\beta^*-1} \varphi_{rk}\left(v_\beta, \varphi_\pi(v_{\beta-1}) \circ \varphi_\pi(v_{\beta-2}) \circ \dots \circ \varphi_\pi(v_1)(\sigma)\right)$ (5)

After that, it recurses tov_{β^*} and executes this process until it reaches the leaf and returns$rk_x(c_\sigma, i)$. This B-tree $\mathcal{T}$ is compressed for supporting rank queries under the MTF encoding with respect to the desired space bound$H_0(MTF(x))$. Consider $r = B^t$ and m is split into ${}^{n}/_{r}$ sub-arrays $A_1, A_2, \dots, A_{{}^{n}/_{r}}$ of size r and a B-tree is constructed over each sub-array. For each $j \in [{}^{n}/_{r}]$, an accurate MTF stack is accumulated at the starting of the sub-arrayA_j, the occurrence of each character $c \in \Sigma$ in the prefix $x[1:(j-1)r]$ and its index in memory.

For a MTF character $\sigma \in \{0,1, \dots, |\Sigma|-1\}$, consider f_σ is the occurrence of σ in m and each occurrence of σ in m is encoded by $\log \frac{n}{f_\sigma}$ bits. A zero-order entropy constraint is exploited by augmenting each node v with an additional value $\varphi_0(v)$ which is the sum of the entropy of the symbols in its sub-tree. So,

$H_0(m) = \sum_{\sigma=0}^{|\Sigma|-1} f_\sigma \log \frac{n}{f_\sigma} = \sum_{i=1}^{n} log \frac{n}{f_{m_i}} = \sum_{j=1}^{{}^{n}/_{r}} \sum_{i \in A_j} log \frac{n}{f_{m_i}} = \sum_{j=1}^{{}^{n}/_{r}} H_0(A_j)$ (6)

ISSN 2515-8260 Volume 07, Issue 06, 2020

In Eq. (6), $H_0(A_j)$ denotes the sum of entropy of the symbols inA_j. Assume Φ_0 is an alphabet of these values$\Phi_0(v)$. Since the entropy of each occurrence of a character can acquire one of $O(r \log n)$ values and the sub-tree of each node have at most $r$leaves,$|\Phi_0| = O(r^2 \log n)$. As a result, for each node v, the vector of values is encoded as $\varphi(v) = (\varphi_\pi(v), \varphi_{rk}(v), \varphi_0(v))$. The additional space needed for accumulating the accurate MTF stack and the rank of each character $c \in \Sigma$ at the beginning of each sub-array $A_j, j \in [n/r]$ is at most$\frac{n}{r}(|\Sigma| \log n + |\Sigma| \log|\Sigma|)$. Also, the space needed for the look-up tables is analyzed by considering the alphabet size $|\Phi| = |\Phi_\pi| \cdot |\Phi_{rk}| \cdot |\Phi_0| \leq O(|\Sigma|! \cdot (r+1)^{|\Sigma|} \cdot r^2 \log n)$with $r = B^t$. Therefore, the look-up tables occupy the space as:

$O(|\Phi|^{B+1} + B \cdot |\Phi|^B) = 2^{O(B|\Sigma| \log|\Sigma| + t|\Sigma| \cdot B \log B + B \log \log n)} = 2^{O(\varepsilon \log n)} = n^{O(\varepsilon)}$(7)

Here, the penultimate equality follows by using the value of B in two cases:

- If $t|\Sigma| > \log \log n$, then $B \log B = {\varepsilon \log n}/{t|\Sigma|}$. Therefore,$t|\Sigma| \cdot B \log B = \varepsilon \log n$and $B \log \log n \leq B \cdot t|\Sigma| \leq \varepsilon \log n$.
- Or else, $B \log B = {\varepsilon \log n}/{\log \log n}$. Thus, $B \log \log n \leq \varepsilon \log n$and $t|\Sigma| \cdot B \log B \leq \log \log n \cdot B \log B = \varepsilon \log n$.

This space consumption is insignificant for small enough constant $\varepsilon > 0$. But, as $B \geq 2$, the minimum redundancy is as:

$O(|\Phi|^3) = O_{|\Sigma|}(r^{3(|\Sigma|+2)}) = O_{|\Sigma|}(r^{3|\Sigma|+6})$ (8)

Moreover, the hypothesis$|\Sigma| = O(1)$ is used and t is adjusted by a constant factor for obtaining the overall space requirement as:

$s = H_0(m) + n/\left(\frac{\log n}{\max(t, \log \log n)}\right)^t + n^{1-\Omega(1)}$ (9)

Thus, this locally-decodable MTF encoding minimizes the space consumption and decoding time of a single character effectively.

4. EXPERIMENTAL RESULTS

In this section, the performance of proposed ETBWT-II algorithm is evaluated by using MATLAB 2017b as well as compared with the existing algorithms such as TBWT-II and TBWT. The comparison analysis is prepared in terms of compression ratio, computation time, encoding time and decompression time. In this experiment, DNA sequences are taken from the ASTRAL SCOPe 1.67 dataset which is obtained from the SCOPe website.Fig. 1 shows the example of two annotated sequences from the dataset.

```
>d1k3ta2 d.81.1.1 (A:165-333) Glyceraldehyde-3-phosphate dehydrogenase (GAPDH) {Trypanosoma cruzi}
SCTTNCLAPIVHVLVKEGFGVQTGLMTTIHSYTATQKTVDGVSVKDWRGGRAAAVNIIPSTTGAAKAVGMVIPSTQGKLTGMSFRVPTPDVSVVDLTFTAARDTS
IQEIDAALKRASKTYMKGILGYTDEELVSADFINDNRSSIYDSKATLQNNLPKERRFFKIVSWY>d2gsaa_ c.67.1.4 (A:) Glutamate-1-
semialdehyde aminomutase (aminotransferase) {Synechococcus sp., strain GR6}
FKTIKSDEIFAAAQKLMPGGVSSPVRAFKSVGGQPIVFDRVKDAYAWDVDGNRYIDYVGTWGPAICGHAHPEVIEALKVAMEKGTSFGAPCALENVLAEMVNDAV
PSIEMVRFVNSGTEACMAVLRLMRAYTGRDKIIKFEGCYHGHADMFLVKAGSGVATLGLPSSPGVPKKTTANTLTTPYNDLEAVKALFAENPGEIAGVILEPIVG
NSGFIVPDAGFLEGLREITLEHDALLVFDEVMTGFRIAYGGVQEKFGVTPDLTTLGKIIGGGLPVGAYGGKREIMQLVAPAGPMYQAGTLSGNPLAMTAGIKTLE
LLRQPGTYEYLDQITKRLSDGLLAIAQETGHAACGGQVSGMFGFFFTEGPVHNYEDAKKSDLQKFSRFHRGMLEQGIYLAPSQFEAGFTSLAHTEEDIDATLAAA
RTVMSAL>d1eysc_ a.138.1.2 (C:) Photosynthetic reaction centre (cytochrome subunit) {Thermochromatium
tepidum}
CEGPPPGTEQIGYRGVGMENYYVKRQRALSIQANQPVESLPAADSTGPKASEVYQSVQVLKDLSVGEFTRTMVAVTTWVSPKEGCNYCHVPGNWASDDIYTKVVS
RRMFELVRAANSDWKAHVAETGVTCYTCHRGNPVPKYAWVTDPGPKYPSGLKPTGQNYGSKTVAYASLPFDPLTPFLDQANEIRITGNAALAGSNPASLKQAEWT
FGLMMNISDSLGVGCTSCHNTRAFNDWTQSTPKRTTAWYAIRHVRDINQNYIWPLNDVLPASRKGPYGDPLRVSCMTCHQAVNKPLYGAQMAKDYPGLYK
```

Fig. 1: Two Annotated Sequences from the Dataset

4.1 Compression Ratio

ISSN 2515-8260 Volume 07, Issue 06, 2020

It defines the ratio of the compressed genomic sequence to the length of actual genomic sequence. Table 1 shows the compression ratio values for proposed and existing algorithms.

Table 1: Comparison of Compression Ratio (for 1000 Sequences)

Block Size	**Compression Ratio (%)**		
	TBWT	**TBWT-II**	**ETBWT-II**
2000	20.229	28.289	32.658
4000	21.624	29.098	33.077
6000	20.560	32.852	35.833
8000	19.907	27.902	34.400
10000	19.596	28.289	34.865

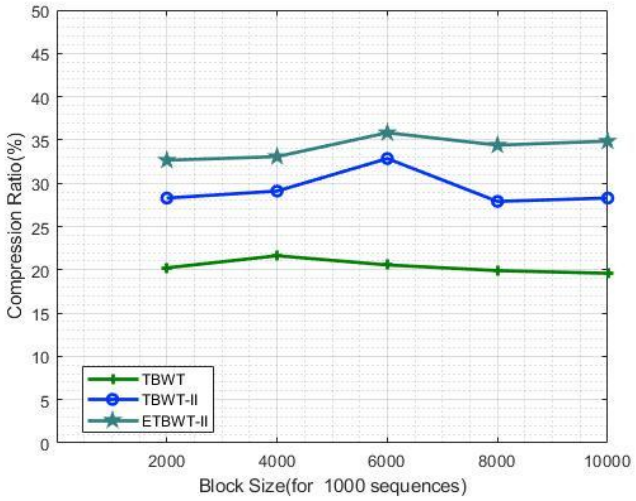

Fig. 2: Compression Ratio vs. Block Size

Fig. 2 shows the compression ratio for TBWT, TBWT-II and ETBWT-II algorithms under different block sizes for 1000 sequences. In this graph, x-axis denotes the block size for 1000 sequences and y-axis denotes the compression ratio in %. From this analysis, it is observed that the proposed ETBWT-II algorithm achieves higher compression than the other algorithms such as TBWT and TBWT-II. For example, if the block size is 2000, then the compression ratio of ETBWT-II algorithm is 32.66% which is higher than TBWT and TBWT-II algorithms whose compression ratio values are 20.23% and 28.29%, respectively.

4.2 Computation Time

It is the time taken for executing the TBWT, TBWT-II and ETBWT-II algorithms to transform the original DNA sequences into different blocks. Table 2 shows the computation time value for proposed and existing algorithms for 1000 sequences.

Table 2: Comparison of Computation Time (for 1000 Sequences)

Algorithms	**Computation Time (sec)**
TBWT	4695.9
TBWT-II	5678.0
ETBWT-II	6375.9

ISSN 2515-8260 Volume 07, Issue 06, 2020

Fig. 3 shows the computation time of TBWT, TBWT-II and ETBWT-II algorithms for 1000 sequences. In this graph, x-axis denotes the number of sequences and y-axis denotes the computation time in seconds. Here, consider 1000 sequences to evaluate the computation time. For 1000 sequences, the computation time of ETBWT-II algorithm is 35.78% higher than TBWT and 12.29% higher than TBWT-II algorithm. From this analysis, it is observed that the proposed ETBWT-II algorithm achieves a high computation time than the other algorithms.

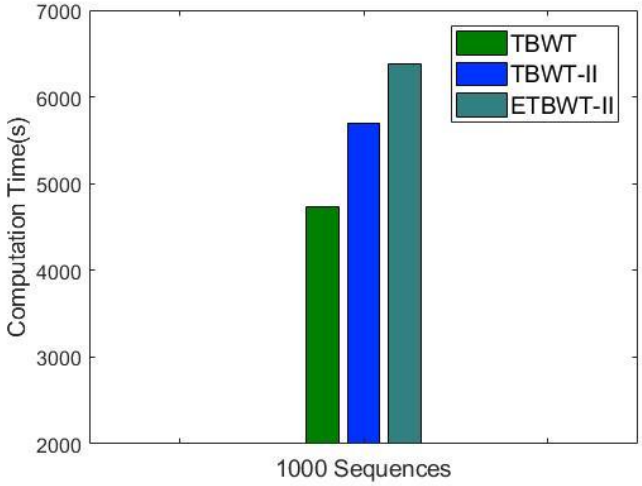

Fig. 3. Comparison of Computation Time

4.3 Encoding Time

It is the time taken for executing the RLE on the transformed DNA sequences using ETBWT-II, TBWT-II and TBWT algorithms. Table 3 shows the RLE time value for proposed and existing algorithms for 1000 sequences.

Table 3: Comparison of RLE Time (for 1000 Sequences)

Algorithms	**RLE Time (sec)**
TBWT	500.5580
TBWT-II	420.2851
ETBWT-II	380.3406

ISSN 2515-8260 Volume 07, Issue 06, 2020

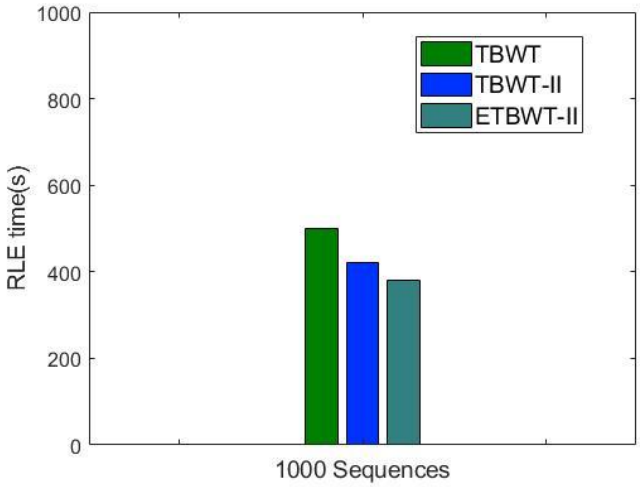

Fig. 4. Comparison of RLE Time

Fig. 4 shows the RLE time of TBWT, TBWT-II and ETBWT-II algorithms for 1000 sequences. In this graph, x-axis denotes the number of sequences and y-axis denotes the RLE time in seconds. Here, consider 1000 sequences to evaluate the RLE time. For 1000 sequences, the RLE time of ETBWT-II algorithm is 24.02% reduced than TBWT and 9.5% reduced than TBWT-II algorithm. From this analysis, it is observed that the proposed ETBWT-II algorithm achieves the reduced RLE time than the TBWT and TBWT-II algorithms.

4.4 Decompression Time

It is defined as the time taken to decompress the original DNA sequences using Inverse versions of ETBWT-II (IETBWT-II), TBWT-II (ITBWT-II) and TBWT (ITBWT) algorithms. Table 4 shows the decompression ratio values for proposed and existing algorithms.

Table 4: Comparison of Decompression Time (for 1000 Sequences)

Block Size	**Decompression Time (sec)**		
	ITBWT	**ITBWT-II**	**IETBWT-II**
2000	2297.2	1971.8	1398.4
4000	2397.0	1577.4	1320.7
6000	2506.0	2075.6	1485.8
8000	2120.5	1971.8	1584.9
10000	2205.3	1478.8	1251.2

ISSN 2515-8260 Volume 07, Issue 06, 2020

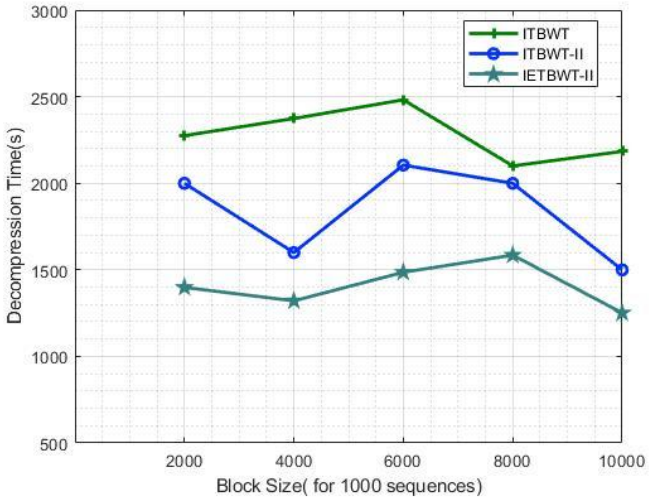

Fig. 5: Decompression Time vs. Block Size

Fig. 5 shows the decompression time for ITBWT, ITBWT-II and IETBWT-II algorithms under different block sizes for 1000 sequences. In this graph, x-axis denotes the block size for 1000 sequences and y-axis denotes the decompression time in seconds. From this analysis, it is observed that the IETBWT-II algorithm achieves less decompression time than the ITBWT-II and ITBWT algorithms. For example, if the block size is 2000, then the decompression time of IETBWT-II algorithm is 1398seconds which is lesser than the ITBWT and ITBWT-II algorithms whose decompression time values are 2297seconds and 1972seconds, respectively.

4.5 Decompression Memory

It is defined as the memory needed for decompression process using IETBWT-II, ITBWT-II and ITBWT algorithms. Table 5 shows the decompression memory values for proposed and existing algorithms.

Table 5: Comparison of Decompression Memory (for 1000 Sequences)

Block Size	**Decompression Memory (Kb)**		
	ITBWT	**ITBWT-II**	**IETBWT-II**
2000	5902.1	5624.3	5000.1
4000	5557.1	5461.4	4994.5
6000	5090.7	4885.3	4465.2
8000	5282.7	4692.6	4289.9
10000	5826.2	5114.7	4678.4

ISSN 2515-8260 Volume 07, Issue 06, 2020

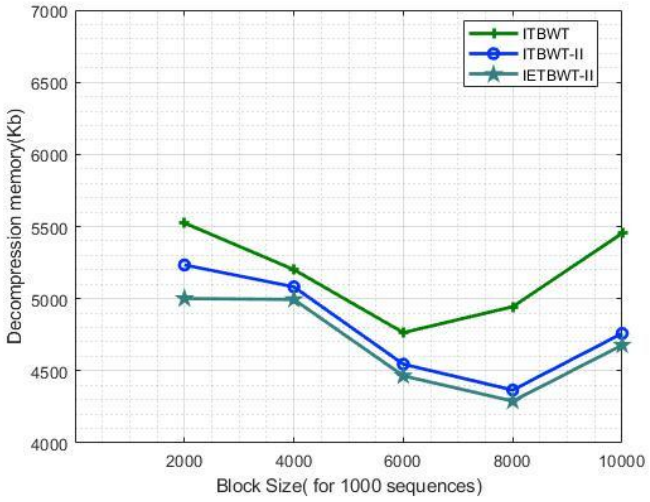

Fig. 6: Decompression Memory vs. Block Size

Fig. 6 shows the decompression memory for ITBWT, ITBWT-II and IETBWT-II algorithms under different block sizes for 1000 sequences. In this graph, x-axis denotes the block size for 1000 sequences and y-axis denotes the decompression memory in Kilobytes (Kb). From this analysis, it is observed that the proposed IETBWT-II algorithm achieves less memory than the ITBWT-II and ITBWT algorithms for decompressing the original DNA sequences. For example, if the block size is 2000, then the decompression memory of IETBWT-II algorithm is 15.28% less than the ITBWT and 11.1% less than the ITBWT-II algorithms.

5. CONCLUSION

In this article, an ETBWT-II algorithm is proposed with the objective of neglecting the local searching abilities within the compressed database and reducing the global space consumption during characters retrieval process. To achieve this objective, a locally-decodable MTF encoding is applied used instead of classical MTF in TBWT-II algorithm. Finally, the experimental results on SCOPe 1.67 dataset proved that the ETBWT-II algorithm has higher performance than the existing TBWT-II and TBWT algorithms.

6. REFERENCES

[1]. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Wheeler DL. GenBank. Nucleic Acids Res., 2005;33Suppl 1:34-38.
[2]. Sugawara H, Ogasawara O, Okubo K, Gojobori T, Tateno Y. DDBJ with new system and face. Nucleic Acids Res., 2007;36Suppl 1:22-24.
[3]. Shumway M, Cochrane G, Sugawara H. Archiving next generation sequencing data. Nucleic Acids Res., 2009;38Suppl 1:870-871.
[4]. Li C, Liu H, Liu J, Qin Y, Wang Z. A burrows-wheeler transform based method for DNA sequence comparison. Comput. Biol.Bioinform., 2014;2(3):33-37.

ISSN 2515-8260 Volume 07, Issue 06, 2020

[5]. Baier U. On undetected redundancy in the burrows-wheeler transform. arXiv preprint arXiv:1804.01937, 2018.
[6]. Ranjitha S, Robert L. Tunneled burrows-wheeler transform encoding with improved indexing for genomic sequences compression. Int. J. Adv. Sci. Technol., 2020;29(5):7682-7691.
[7]. Sardaraz M, Tahir M, Ikram AA, Bajwa H. SeqCompress: an algorithm for biological sequence compression. Genom., 2014;104(4):225-228.
[8]. Kimura K, Koike A. Analysis of genomic rearrangements by using the burrows-wheeler transform of short-read data. BMC Bioinform.,2015;16(18):S5.
[9]. Eric PV, Gopalakrishnan G, Karunakaran M. An optimal seed based compression algorithm for DNA sequences. Adv. Bioinform., 2016.
[10]. Sandhya, Kulkarni S, Kini Y. Pre equal architecture for lossless data compression and decompression using hybrid algorithm. Int. J. Adv. Electr. Electron. Eng., 2017;6.
[11]. Saada B, Zhang J. DNA sequence compression technique based on nucleotides occurrence. Proc.Int. Multi-Conf. Eng. Comput. Sci., Hong Kong, vol. 1, 2018.
[12]. Punitha K, Murugan A. A novel algorithm for DNA sequence compression. Emerg. Res.Comput. Inf.Commun. Appl., 2019;151-159.