Received: May 18, 2023. Revised: September 16, 2023. 108

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Enhanced Index Based DNA Sequence Compression Algorithm

Arunachalaprabu Gurunathan[1]* **Fathima Bibi Kaja Moideen[1]**

[1]*Department of Computer Science, Thanthai Periyar Government Arts and Science College (Autonomous), Affiliated to Bharathidasan University, Tiruchirappalli, India*
* Corresponding author's Email: guruarun12@gmail.com

Abstract: Biological data analyses involve researchers from several fields. To store and manipulate the huge volume of biological data obtained from different aspects is difficult. Compression algorithms considerably increase the storage medium's capacity while reducing the number of bits required representing the sequence. The core concept behind EIBDNASCA involves creating an optimized index file, which stores the non-repetitive bases (run length less than 8). This index file plays a crucial role in swiftly retrieving and reconstructing specific segments of the DNA sequence during the decompression process. In addition to the index file, EIBDNASCA incorporates a work file, which stores the repetitive bases (run length above 8) and represented in binary form. This work file allows the algorithm to perform various pre-processing and transformation tasks on the DNA sequence before generating the final compressed output. Finally, an enhanced Huffman coding technique is applied to the symbols present in the index file, optimizing the encoding process for more efficient compression. The proposed algorithm is examined using a variety of different GenBank database sources. Compression ratio, compression gain, and time required to compress and decompress the sequences are the metrics used to assess the performance. The experimental findings indicate that EIBDNASCA attains an average compression ratio of 1.23 bpb (bits per base) with an average compression gain of 84.52%. The average compression time is recorded at 0.590 seconds, and decompression is completed in 0.625 seconds.

Keywords: Compression ratio, Deoxyribonucleic acid, Huffman coding, Index file, Run length encoding.

1. Introduction

Three kinds of molecules are involved in the story of genetic material namely proteins, deoxyribonucleic acid (DNA) and Ribonucleic acid (RNA). Proteins: a set of molecules whose behavior, concentration and shape determine the cell's properties. The cells of hair, nerve or blood are distinct because they are composed of different kinds of proteins. DNA: another molecule that defines the specific proteins and in turn depends on proteins to become active. RNA: Its structure is similar to DNA. The nucleus of a cell contains the genetic material DNA that is transmitted from one generation to the next.

The DNA molecule are chains of long strands composed of four different kinds of bases Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). Two strands are joined together as bases can bind to each other (For example: Adenine – Thymine and Cytosine – Guanine). A DNA sequence is a long string that describes the accurate ordering of these four bases.

Growth in bioinformatics allows fast collection and interpretation of biological datasets. In the field of data processing and interpretation, biologists encountered similar type of information overload and faced a lot of challenges. One of the important challenges was to collect biological information of individual DNA sequences and determine the order of the sequences that make up human DNA. Recent genome projects generate trillions of base pairs of biological data at rapid rate constantly. The Release 254(February 2023) of GenBank contains 1731302248418 bases [1].

Table 1 shows the number of bases as a pointer of the augmentation of GenBank database.

Table 1. Statistics of GenBank
(From February 2022 to February 2023)

Release	Month Year	Number of Bases	Sequences
248	02 / 2022	1173984081721	236338284
249	04 / 2022	1266154890918	237520318
250	06 / 2022	1395628631187	239017893
251	08 / 2022	1492800704497	239915786
252	10 / 2022	1562963366851	240539282
253	12 / 2022	1635594138493	241015745
254	02 / 2023	1731302248418	241830635

Storing large volume of DNA sequences will effect in memory overflow. The amount of available storage space is classically the bottleneck. Another interesting and important setback is transmitting the DNA sequences from one place to another. Network traffic is generated when datasets are transferred between databases. In such cases, accessing DNA sequences is a difficult problem in many scientific applications. As a result, several algorithms are proposed to compress the size of sequences.

Compression is the task of minimizing the bits to represent a base. Two types of compression techniques are: Lossy and lossless. Data compressed using lossy techniques cannot be recovered exactly. The reconstructed data differs from original one. DNA variant, called mutation occurs when (1) one base being replaced incorrectly by another base (2) addition of an incorrect base to the sequence (3) deletion of a correct base from the sequence. In such cases the sequence gets altered and lead to malfunctioning. For this reason a lossy compression technique is not usually applied to text compression, specifically DNA sequence compression. A lossless compression is the task to reduce the size of file and restore the original data in decompression. Lossless compression improves storage capacity and exchange of data worldwide. The data is uploaded and downloaded accurately and quickly at any instant. Researchers believe that the quality of compression largely depends on the repetitive and/or non-repetitive bases within the DNA sequence. Therefore, repetitive/non-repetitive bases have received a considerable amount of attention in recent times. The work addresses a solution to this problem with the help of lossless compression. Several such lossless DNA sequence compression algorithms have been developed to reduce the need for large amounts of storage and to increase transmission.

Some of the major compression algorithms are BioCompress [2], BioCompress2 [3], Gen Compress [4], DNACompress [5], Normalized Maximum Likelihood (NML) [6], GeNML [7] and DNA Sequence Compressor (DNASC) [8]. These compression algorithms are suitable for more repetitive similar DNA sequences but stand infeasible for non-repetitive DNA sequences.

In bioinformatics, the invention of lossless compression algorithms might be crucial to the understanding and analysis of DNA sequences. Improving storage device capacity and reducing DNA sequence memory occupancy are two specific objectives of the research work. In order to compress DNA sequences, an EIBDNASCA based on run length encoding and improved huffman coding methods is introduced in the work. On both repetitive and non-repetitive bases, the basic RLE method is used. Non-repetitive bases take up more storage space and reduce the storage medium's capacity. Thus, the improved huffman coding method is used after moving the non-repetitive bases to an index file. The proposed approach seems to be a promising way to compress DNA sequences, even in high throughput situations. The paper is organized into five sections and the general information of each section is given as follows. Section 2 reviews the recent studies made in DNA compression. Performance Metrics are defined in section 3. Section 4 describes the working of EIBDNASCA and section 5 gives the experimental results. To conclude, Section 6 briefs the view of the proposed work.

2. Related works

Although compression methods for DNA sequences in bioinformatics are relatively a novel development, the field is strongly associated with various other fields. The following section discusses the literature review for latest DNA compression algorithms, its experimental methodology and results.

Krishnamoorthy and Karthikeyan, (2022) proposed a novel hybrid streamlining of hospitalization–subordinate DNA pressure (HOARDNAComp) compression technique based on auto regression and firefly algorithm. The technique utilizes an even mode factual strategy associated with auto regression. Modified firefly computation set the model boundary values and addresses the issue during presentation of computation in a particular cycle. Experiments have showed that an average compression ratio of 1.39 bpb was obtained. The performance of HOARDNAComp when compared with DNA compression using particle swarm optimization (DCPSO) [9] resulted in minimum compression ratio [10].

Rosario Gilmary and Murugesan (2021) described a bit reduction method to compress the DNA sequence. The method consists of three stages, namely, a) reduce the number bits of DNA sequences

b) encode binary format to hexadecimal c) apply Huffman coding to hexadecimal values. This study was also compared with existing algorithms. The result showed better compression ratio and high saving storage. Multiple transformations and conversions applied to the DNA sequence, leads to increased complexity in compression [11].

Mansouri et al., (2020) used a DNA compression algorithm with single-block encoding (DNAC-SBE). In One-Bit Method, the location of bases with high frequencies is substituted by 1s and others by 0s. The SBE encodes the streams produced. Each block is assigned a short codeword dynamically. DNAC-SBE discovers the unidentified bases. The proposed algorithm achieved high saving percentage when compared with other algorithms The drawback of DNAC-SEB is its rigid use of fixed block size (seven bits) for encoding, which lead to average compression results against diverse genomic data. [12].

Murugesan, (2020) designed a codon based compression algorithm (CBCA) to compress the DNA sequences [13]. It compresses and decompresses the data without using dictionary which reduces the requirement of additional memory. The method resulted with an improved compression ratio of 1.59 bpb when compared to existing algorithms. Experimental results have shown that 0.18 seconds is required to compress the sequences. The drawback pertains to utilizing fixed-length binary values (1 bit, 2 bits, 4 bits, 5 bits, or 6 bits) for the encoding process.

Hui Chen, (2020) developed a context modeling tool named entropy coding technique (ECT) for compressing the genomic data. The input sequence is alienated into coding sequence, intron, RNA and residual clusters. It amalgamates the features of entropy coding technique. Initially, all sets will be arranged associated to the specific sequence features and ECT will encode these sequences. The ECT achieved an average of 1.72 bpb compression ratio. Disadvantage: it needs long computation time [14].

3. Performance metrics

The following are the metrics to measure the efficiency of a compression algorithm:

3.1 Compression ratio (CR)

Compression ratio binds the size of original file to the size of reduced file. It is articulated in bits per base (bpb) or bits per character (bpc).

CR = Compressed file size / Original file size

3.2 Compression factor (CF)

Compression factor depicts the ratio of original file size to the compressed file size. The compression factor is defined as

CF = Original file size / Compressed file size

3.3 Saving percentage (SP)

Saving percentage depends on the value of original file size minus compressed file and the original file size. It is expressed in terms of percentage.

SP = (Original file size -Compressed file size) / Original file size

3.4 Compression time

Compression time is the time needed to compress a file.

3.5 Decompression time

The time needed to rebuild the file to its original is termed as decompression. Compression and decompression time are formally expressed in seconds.

4. Proposed algorithm

Current DNA sequencing technologies generate large volume of genomic data (DNA Sequences). Rapid growth of these sequences has become extremely high. They are readily available for sequence mining, homology searches and modeling. Effective and scalable storage medium that analyze and store these genomic datasets is required.

Among the issues faced by the researchers of bioinformatics community, few of them stand out.

i) Increase in DNA sequences led to tremendous need of disk storage capacity.
ii) Transferring genomic data from one point to another was generally difficult and took more time.
iii) DNA sequences contain repetitive and non-repetitive bases. It is known that non-repetitive bases occupy more space in compression. Inference of novel pathway needs lossless compression algorithms to compress such genomic data.

Some of the major goals associated with these issues are:

i) A novel compression algorithm is required to reduce the size of the genomic data and achieve better compression ratio specifically in DNA

sequences containing both repetitive and non-repetitive bases.

ii) The algorithm must provide an efficient way to store the vast amount of genomic data without affecting the effectiveness of the storage medium. The compression algorithm must increase the capacity of storage medium.

iii) The algorithm should be scalable and able to work well for various sizes.

iv) The algorithm must be capable of transferring the genomic dataset from one node to another as fast as possible and also reduce network traffic.

The need for lossless algorithms to compress the data is highly essential. In this research work, a novel algorithm has been proposed that achieves good compression gain, better compression ratio and reduced time taken for compression as well as decompression. In particular, the EIBDNASCA compresses both repetitive R and non-repetitive NR bases in a sequence. The EIBDNASCA combines both the features of RLE and improved Huffman coding methods. At first, apply basic RLE method to find repetitive R and non-repetitive NR bases in a sequence D. Now, read the repetitive segment R (r, b1; r, bn) and write the base(s) with run length > 5 in work file. At the work file R(r, b1; r, bn-1), the repetitive bases are represented in binary form while R(r, bn) alone represented uniquely. Then, read the non-repetitive bases NR (r, b1; r, bn) and write the base(s) with run length ≤ 5 in index file. Here, the non-repetitive bases NR (r, b1; r, bn-1) are written to the index file as symbols and NR(r, bn) alone represented by unique symbol. Now apply improved Huffman coding to the symbols in index file. Usually compression algorithms use a flag bit to transfer the base pair (runlength, base) into index file. That is, a flag bit set in work file (to reconstruct the data) and pair (runlength, base) written in index file. However, such algorithms have a major drawback: if the work file has n number of flag bits, it needs (n x s) bits to represent the flag bits where s is length of flag bit. This can occupy more storage space. To overcome this issue, the EIBDNASCA transfers a base pair to another file by representing the last base with a unique symbol instead of flag bit. Another important advantage is that the algorithm compresses the file based on the bases in the DNA sequence, not the file size. This resolves the scalability issue.

4.1 Run length encoding (RLE) method

The RLE method counts the occurrences k of the base b from the input sequence D and writes (runlength, base) – kb. The resulting k consecutive occurrence of a base is called runlength and is normally referred to as Run Length Encoding or RLE. The given sequence D has n bytes (30bases) 'TTTTAAGCTTTTTTTTTTTTTTTTTTG' taken from HUMGPRTB dataset. The runlength k of the sequence is computed by applying RLE and replacing k occurrences with the pair kb, '4T 2A 1G 1C 21T 1G' (length 12 bytes). The highlight about RLE is that, the method is easy to implement and requires less amount of processing power. Drawback: for more non-repetitive NR bases the method might not be appropriate.

4.2 Work file – bit representation for bases

Table 2 shows bit representation of the EIBDNASCA. Let D = {b1, b2, b3 … bm} be a DNA sequence, where bi ∈ A, and A is the set of all bases in the file. Read the repetitive segment R (r, b1; r, bn) where r ≥ 6. Set bit representation and base value for R (r, b1; r, bn-1) using Table 2 and for (r, bn) uniquely from Table 3. Table 2 offers a comprehensive and novel representation of bases in the work file using a four-bit encoding scheme. Notably, the first bit of this representation, denoted as "1" serves a dual purpose: it signifies that the control is encoding the current base while simultaneously transitioning to the next base within the work file. However, it is crucial to emphasize that this control is not transferred to the index file. The significance of this table lies in its provision of detailed bit patterns for representing bases, covering a run length ranging from 9 to 16. For instance, when dealing with a run length of 9 and the base "A," the Bit Representation (BR) corresponds to 000, while the base value is represented as 00. Table 3 presents a comprehensive and detailed representation of the last base of the segment in the work file, utilizing a four-bit encoding scheme. In this encoding scheme, the first bit, indicated as "0" serves as a crucial control signal that signifies the current base is encoded and subsequently transferred to the Index file. Importantly, this transfer ensures that the next base in the work file is not processed. The table covers a range of run lengths from 9 to 16, offering specific Bit Representation (BR) patterns for each base (A, C, G, T). For instance, when dealing with a run length of 10 and the base "G" the corresponding Bit Representation is denoted as 001, while the base value is represented as 10.

4.3 Huffman coding method

David Huffman designed a lossless compression method that reduces the size of the file. The idea is

that codes using this method are referred as Huffman codes [15].

Algorithm

1. Sort the bases based on their frequency in descending order.
2. Make each base as a leaf node. In the beginning, the least frequency base is taken.
3. Two minimum frequency nodes are extracted to construct a new node. Make the minimum frequency base as left node and second minimum frequency base as right node. The new node is the sum of the left and right nodes frequencies.
4. Now the new node is inserted to tree.
5. Repeat steps (3) & (4) until reaches the root.
6. Assign left edge ← 0 and right edge ← 1 for non-leaf node.
7. Now, determine codeword by traversing the tree.

4.4 Improved huffman coding method

The key concept of improved Huffman coding method is substituting a unique symbol to the bases contained in index file. This method can be used to further reduce memory space and time. For instance, consider a DNA sequence D = {b1, b2, b3 …bm} where bi ∈ A, A is the set of all bases in file. Read the non-repetitive segment NR (r, b1; r, bn) where r ≤ 5. Set the symbol representation for NR (r, b1; r, bn-1) and uniquely for (r, bn) from Table 4. The Improved Huffman coding method is used in symbols and not to bases.

The first part of the Table 4 is structured as an 8 x 8 matrix, with each cell containing a symbol that represents a specific control code. This control code is responsible for encoding the current base, and then the encoded information is transferred to the work file. The rows of the matrix correspond to different run lengths, ranging from 1 to 8. At run length 1, denoted by the symbol "!", the control code encodes the current base as 'A' and subsequently transfers it to the work file. Similarly, for run lengths 2 to 8, the symbols represent the bases 'C', 'G', and 'T', respectively, and their corresponding control codes facilitate encoding and transfer. In table 4, the printable ASCII values are assigned to the bases in a systematic manner. For instance, the ASCII value 33 is assigned to position a11 of the matrix, corresponding to the symbol "!". Similarly, ASCII value 34 is assigned to position a12, representing the symbol "". This pattern continues for the entire matrix, enabling a consistent and reliable mapping of symbols to their respective ASCII representations.

The encoding scheme is based on an 8 x 8 matrix (second portion of Table 4), where each row corresponds to a specific run length (ranging from 1 to 8), and each column represents one of the four DNA bases (A, C, G, T). The main objective of this encoding is to transform long DNA sequences into more compact symbols, facilitating the storage and analysis of vast genetic data. The encoding process involves controlling the current base and then encoding the subsequent base directly within the index file and not transfers to work file. For instance, for run length 2, the symbols assigned to the bases are 'B' for A, 'J' for C, 'R' for G, and 'Z' for T. Moreover, the ASCII values are employed to assign specific numerical values to the bases in the matrix, streamlining the encoding process further.

Table 2. Work file – bit representation for bases

Run Length	Bit Representation (BR) – Base				
	BR	**A**	**C**	**G**	**T**
9	1000	00	01	10	11
10	1001	00	01	10	11
11	1010	00	01	10	11
12	1011	00	01	10	11
13	1100	00	01	10	11
14	1101	00	01	10	11
15	1110	00	01	10	11
16	1111	00	01	10	11

Table 3. Work file – bit representation for last base

Run Length	Bit Representation (BR)– Base				
	BR	**A**	**C**	**G**	**T**
9	0000	00	01	10	11
10	0001	00	01	10	11
11	0010	00	01	10	11
12	0011	00	01	10	11
13	0100	00	01	10	11
14	0101	00	01	10	11
15	0110	00	01	10	11
16	0111	00	01	10	11

Table 4. Symbols for bases in index file

Run Length	Symbol (Last base)				Symbol (Base)			
	A	C	G	T	A	C	G	T
1	!	)	1	9	A	I	Q	Y
2	"	*	2	:	B	J	R	Z
3	#	+	3	;	C	K	S	[
4	$	,	4	<	D	L	T	\
5	%	-	5	=	E	M	U	]
6	&	.	6	>	F	N	V	^
7	'	/	7	?	G	O	W	_
8	(	0	8	@	H	P	X	`

Illustration

The working principle of the EIBDNASCA is demonstrated as follows:
A sequence D with size 41 bytes is taken.

D={AAAAACCCTTAAAAAAAACCCCCCGGGTTTTTTTTGGGGG}, | D | = 41.
After applying RLE to D

5A 3C 2T 8A 6C 3G 9T 5G

Work file

9T
0000-
11

Index file

5A	3C	2T	8A	6C	3G	5G
E	K	Z	H	N	3	5

Table 5 illustrates the bits required to represent the bases after applying improved Huffman coding in index file.

Required bits = 2 x Frequency (5A) + 2 x Frequency (3C) + 3 x Frequency (2T) + 3 x Frequency (8A) + 3 x Frequency (6C) + 4 x Frequency (3G) + 4 x Frequency (5G)
= 2 x 1 + 2 x 1 + 3 x 1 + 3 x 1+ 3 x 1 + 4 x 1+ 4 x 1
=2+2+3+3+3+4+ = 21 bits

Size after Compression = Work file + Index file
= 06 bits + 21 bits
= 27 bits
= 3.37 bytes

Compression ratio is given by

$$Compression\ Ratio = \frac{Size\ after\ Compression}{Size\ before\ Compression} x\ 8$$

= (3.37 / 41) x 8
= 0.65 bpb

5. Results and discussion

5.1 Datasets

Six different standard benchmark datasets taken from GenBank database are used to experiment the proposed method. Table 6 gives the major descriptions of these datasets. The EIBDNASCA is implemented and executed in Java. Relationship between size of file before compression and after compression (original size and compressed size) is needed to be considered.

Table 5. Improved Huffman coding method

Bases	Symbols	Frequency	Code Word	Length of Codeword	Number of Bits Required
5A	E	1	00	2	2
3C	K	1	01	2	2
2T	Z	1	100	3	3
8A	H	1	101	3	3
6C	N	1	110	3	3
3G	3	1	1110	4	4
5G	5	1	1111	4	4
Required bits = 21 bits					

EIBDNASCA

Compute the run length of bases

```
for all bases b of D do
Compute the runlength of b
for i = 0 to |D|
        runlength ← 1
        while i+1 < |D| && D.base(i) =
        =D.base(i+1)
                runlength ← runlength + 1
                i ←i + 1
        end while
        append (runlength, base(i))
end for
```

Work file – Bit Representation

```
for all pairs P1 do
for g = 0 to | P1 |
        while g+1 < | P1 | && P1.runlength(g) ≥
        6
                R ← runlength, base
                g ←g + 1
        end while
        for j = 0 to | R | - 1 do
                br ← set bit representation
        end for
lastbase ← set unique bit representation
append (br, lastbase)
end for
```

Index file – Symbol representation

```
for all pairs P1 do
for g = 0 to | P1 |
        while g+1 < | P1 | && P1.runlength(g) ≤
        5
                NR ← runlength, base
                g ←g + 1
        end while
        for j = 0 to | NR | - 1 do
                s ← symbol
        end for
slastbase ← unique symbol
end for
for i to s
        s.frequency++
        codewords ← createTree(s.frequency)
        append (codewords)
end for
```

Table 6. Description of standard bench mark datasets

Source of Sequence	Name of the Sequence	Length (Bytes)	Size of File (Kilobytes)
Chloroplast	Chmpxx	121024	118.19
Human	Humdystrop	38770	37.86
	Humhbb	73308	71.59
	Humhprtb	56737	55.40
Mitochondria	Mpomtcg	186609	182.23
Virus	Vaccg	191737	187.24

This relationship is examined with three cases: the best, average and worst. It also depends on the number of repetitive and non-repetitive bases of DNA sequences.

5.2 Best case

The best case is when compression ratio is good. A sequence with several repetitive bases is taken for best case.
A sequence D with size 40 bytes is taken.

D={AAAAAAAAAGGGGGGCCCCCCTTTTTCCCCCCCCTTTTT}, | D | = 40

After applying RLE to D

9A 6G 6C 5T 9C 5T

Work file

9A 9C
0000-00 0000-01

Index file

6G 6C 5T 5T
V N = =

Table 7 illustrates the bits required to represent the bases after applying improved Huffman coding in index file. Compression ratio for best case = 0.45 bpb.

5.3 Average case

A sequence D with merely average repetitive bases of size 40 bytes is taken.

D={AAAAAAATCCCCCGGTTTCCCCCCCCCCCCCCCCAAA GGCC}, | D | = 40

After applying RLE to D

7A 1T 5C 2G 3T 15C 3A 2G 2C

Work file

15C
0110 - 01

Index file

7A 1T 5C 2G 3T 3A 2G 2C
G Y M R ; C R *

Table 7. Improved huffman coding (best case)

Bases with Runlength	Symbol Representation	Symbol's Frequency	Code-Word	Size of Code word	Bits Required
5T	=	2	0	1	2
6G	V	1	10	2	2
6C	N	1	11	2	2
Total bits Required					6

Table 8. Improved huffman coding (average case)

Bases with Runlength	Symbol Representation	Symbol's Frequency	Code-Word	Size of Code word	Bits Required
2G	R	2	00	2	4
7A	G	1	01	2	2
1T	Y	1	100	3	3
5C	M	1	101	3	3
3T	;	1	110	3	3
3A	C	1	111	4	4
2C	*	1	111	4	4
Total bits Required					23

Table 8 illustrates the bits required to represent the bases after applying improved Huffman coding in index file. Compression ratio for average case = 0.72 bpb.

5.4 Worst case

A sequence D with less repetition is taken.

D={TAACGGGTCTCGGGGTTTTTCCCCACGTCCCGGCTAAAGT}, | D | = 40

After applying RLE to D

1 2 1 3 1 1 1 1 4 5 4 1 1 1 1 3 2 1 1 3 1 1
T A C G T C T C G T C A C G T C G C T A G T

Index file

1 2 1 3 1 1 1 1 4 5 4 1 1 1 1 3 2 1 1 3 1 1
T A C G T C T C G T C A C G T C G C T A G T
Y B I S Y I Y I T] L A I Ç Y K R I Y C Ç Y

Table 9 illustrates the bits required to represent the bases after applying improved Huffman coding in index file. Compression ratio for worst case = 1.8 bpb.

5.5 Results of EIBDNASCA for standard datasets

To statistically compare the performance of the

Table 9. Improved huffman coding (worst case)

Bases with Runlength	Symbol Representation	Symbol's Frequency	Code Word	Size of Code Word	Bits Required
1T	Y	6	00	2	12
1C	I	5	01	2	10
1G	Q	2	1000	4	8
2A	B	1	1001	4	4
3G	S	1	1010	4	4
4G	T	1	1011	4	4
5T	]	1	1100	4	4
4C	L	1	1101	4	4
1A	A	1	1110	4	4
3C	K	1	111100	6	6
2G	R	1	111101	6	6
3A	C	1	111110	6	6
Total bits Required					72

Table 10. Standard datasets results of EIBDNASCA

DNA Sequence	Actual Size (Bytes)	Reduced Size (Bytes)	Compression Ratio (bps)	Compression Gain %	Time Taken (Seconds)	
					Compression	Decompression
Chmpxx	121024	17345	1.14	85.66	0.544	0.578
Humdystrop	38770	6186	1.27	84.04	0.513	0.562
Humhbb	73308	11132	1.21	84.81	0.519	0.546
Humhprtb	56737	8936	1.25	84.25	0.509	0.591
Mpomtcg	186609	30086	1.28	83.87	0.699	0.676
Vaccg	191737	29716	1.23	84.50	0.756	0.794
Average			1.23	84.52	0.590	0.625

algorithm to that of the standard algorithms and existing compression algorithms the EIBDNASCA is experimented with basic RLE method along with improved Huffman coding method. Table 10 summarizes the results of EIBDNASCA. Experimental results demonstrate that an average compression ratio of 1.23 bpb and average compression gain of 84.52% is achieved. The average time taken for compression is 0.590 seconds and that of decompression is 0.625 seconds.

Table 11. Comparison analysis of EIBDNASCA over existing algorithms

DNA Sequence	DNAC-SBE [12]	CBCA [13]	ECT [14]	HOARDNA Comp [10]	Bit Reduction [11]	IBDNASCA	EIBDNASCA
Chmpxx	1.60	-	1.58	1.33	-	1.40	1.14
Humdystrop	1.72	1.55	-	1.39	1.64	1.53	1.27
Humhbb	1.71	1.55	1.83	1.44	1.65	1.50	1.21
Humhprtb	1.72	1.54	1.85	1.45	-	1.51	1.25
Mpomtcg	1.72	1.55	-	1.40	1.62	1.57	1.28
Vaccg	1.67	1.57	1.78	1.32	1.66	1.52	1.23
Average Ratio	1.69	1.55	1.76	1.38	1.64	1.51	1.23

Table 11 presents a comparison analysis of several DNA sequence compression algorithms, including ECT, DNAC-SBE, Bit Reduction, CBCA, HOARDNA Comp, IBDNASCA, and the Enhanced Index Based DNA Sequence Compression Algorithm (EIBDNASCA). Among these algorithms, EIBDNASCA stands out with an impressive compression ratio of 1.23. This indicates that EIBDNASCA can compress DNA sequences to approximately 81.30% of their original size, making it the most efficient algorithm in the study. In contrast, other algorithms such as ECT, DNAC-SBE, Bit Reduction, CBCA, HOARDNA Comp, and IBDNASCA achieved compression ratios of 1.76, 1.69, 1.64, 1.55, 1.38, and 1.51, respectively, resulting in percentage size reductions ranging from 56.81% to 66.22%. EIBDNASCA consistently outperforms the other algorithms in terms of percentage of size reduction. While HOARDNAComp achieved the better compression efficiency with a percentage size reduction of 72.46%, EIBDNASCA managed to surpass it by a significant margin, achieving an 81.30% reduction. On average, EIBDNASCA achieved a reduction of 81.30%, which is substantially higher than the average reductions of the other algorithms, ranging from 56.81% to 66.22%. This data highlights EIBDNASCA's exceptional ability to compress DNA sequences more effectively than existing methods, potentially leading to more efficient storage and transmission of genetic data.

Table 12 presents a comprehensive comparison of the EIBDNASCA with various standard algorithms used for DNA sequence compression. The data reveals that EIBDNASCA consistently outperforms all other algorithms in terms of compression efficiency. For instance, when considering the average compression ratio, EIBDNASCA achieves an impressive value of 1.23, which is significantly better than the average ratios obtained by the standard algorithms, ranging from 1.74 to 2.27. This indicates that EIBDNASCA can compress DNA sequences to a much smaller size compared to its counterparts. The average percentage of size reduction further underscores the exceptional performance of EIBDNASCA, as it achieves an average reduction of 81.33%, while the standard algorithms achieve reductions ranging from 44.05% to 57.47%. This substantial difference in compression capabilities positions EIBDNASCA as a cutting-edge solution for DNA sequence compression, offering the potential for more efficient storage and analysis of genetic data.

Analyzing the individual DNA sequences, EIBDNASCA consistently exhibits superior performance across all cases. For instance, for the sequence "Chmpxx" EIBDNASCA achieves a compression ratio of 1.14, whereas the standard algorithms range from 1.50 to 2.25, resulting in percentage size reductions of 44.05% to 54.94%. Similarly, for the "Humdystrop" sequence, EIBDNASCA attains a compression ratio of 1.27, while the standard algorithms range from 1.89 to 2.37, leading to percentage size reductions of 54.05% to 57.47%. These results demonstrate the consistent superiority of EIBDNASCA in compressing DNA sequences across diverse cases.

Fig. 1 depicts that the algorithm achieves good compression ratio for the standard benchmark datasets (Chmpxx, Humdystrop, Humhbb, Humhprtb, Mpomtcg and Vaccg). Among these Chmpxx and Humhbb produces higher compression ratio of 1.14 bpb and 1.21 bpb respectively.

Fig. 2 gives the comparison analysis over existing algorithms. X-axis denotes the algorithms and Y-axis denotes the average compression ratio obtained. In almost all datasets the proposed work resulted with notable performance when compared with the existing works.

Fig. 3 describes the relation between compression ratio achieved by the algorithm and standard compression algorithms for the six standard datasets. The various average compression ratios 2.27, 1.85, 1.82, 1.80, 1.79, 1.74 and 1.23 in terms of bpb were acquired. Thus a major observation about

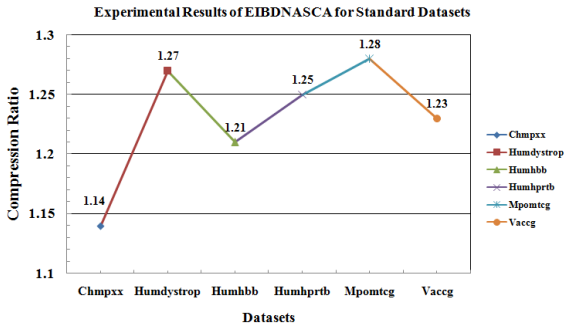

Figure. 1 Experimental results of EIBDNASCA for standard datasets

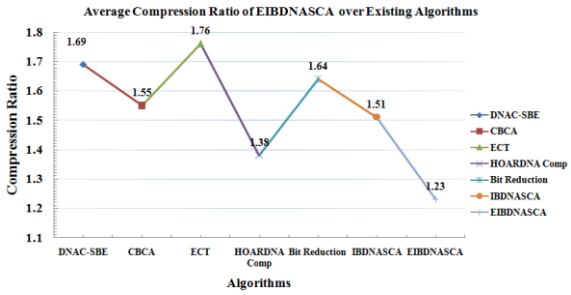

Figure. 2 Average compression ratio of EIBDNASCA over existing algorithms

Table 12. Comparison over standard algorithms

DNA Sequence	WinRAR	Bio-Compres2 [3]	Gen Compress [4]	DNA Compress [5]	Ge-NML[7]	DNASC [8]	EIBDNASCA
Chmpxx	2.25	1.68	1.67	1.67	1.66	1.50	1.14
Humdystrop	2.37	1.93	1.92	1.91	1.91	1.89	1.27
Humhbb	2.22	1.88	1.82	1.79	-	-	1.21
Humhprtb	2.23	1.91	1.85	1.82	1.76	1.71	1.25
Mpomtcg	2.30	1.94	1.91	1.89	1.88	1.88	1.28
Vaccg	2.23	1.76	1.76	1.76	1.76	1.70	1.23
Average Ratio	2.27	1.85	1.82	1.80	1.79	1.74	1.23

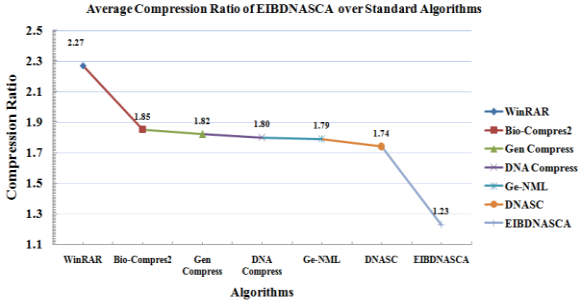

Figure. 3 Average compression ratio of EIBDNASCA over standard algorithms

the proposed work is that when comparing standard algorithms and the EIBDNASCA results, one can see that EIBDNASCA improved the compression ratio of the previous works; however there is notable difference among the saving percentage that is achieved by the EIBDNASCA.

6. Conclusion

The work focused on lossless algorithm that reduces the size of DNA sequence. The novel algorithm exploits the features of basic RLE as well as improved Huffman coding for the various datasets to find the repetitive and non-repetitive bases. The repetitive bases are represented in binary form. The last base is uniquely represented in work file. The non-repetitive bases are transferred to index file and are represented by symbols. Here, the last base is denoted by unique symbol (not flag bit). As a result, the proposed algorithm achieves good compression ratio of 1.23 bpb and also improves the capacity of storage medium (84.52%). The time taken for compression is 0.590 seconds and that of decompression is 0.625 seconds. The illustration of the work in the previous section evidently shows that the EIBDNASCA can be useful in the reduction of the size of DNA sequences. The results give substantial development over the existing and state of the art compression algorithms.

Conflicts of interest

The authors declare no conflicts of interest.

Author contribution

Arunachalaprabu G assumed the responsibility of overseeing the conceptualization, development of the methodology, software implementation, meticulous analysis, efficient resource management, data collection, preparation of the original draft, thorough review and editing of the manuscript, as well as the thoughtful visualization of the paper. On the other hand, Fathima Bibi K performed the critical tasks of supervision and software administration.

References

[1] www.ncbi.nim.nih.gov/genbank/statistics

[2] S. Grumbach and F. Tahi, "Compression of DNA Sequences", In: *Proc. of Conf. on Data Compression*, pp. 340-350, 1993.

[3] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: Genetic sequences", *Information Processing and Management*, Vol. 30, No. 6, pp. 875-886, 1994.

[4] X. Chen, Kwong and M. Li, "A Compression algorithm for DNA sequences and its application in genome comparison", In: *Proc. of the Tenth Workshop on Genome and Informatics*, Vol. 10, pp. 51-61, 1999.

[5] X. Chen, M. Li, B. Ma, and J. Tromp, "DNACompress: fast and effective DNA sequence compression", *Bioinformatics*, Vol. 18, No. 12, pp. 1696-1698, 2002.

[6] I. Tabus, G. Korodi, and J. Rissanen, "DNA sequence compression using the normalized maximum likelihood model for discrete regression", In: *Proc. of the Data Compression Conf.*, pp. 253-262, 2003.

[7] G. Korodi and I. Tabus, "An efficient Normalized Maximum Likelihood for DNA Sequence Compression", *ACM Trans., Information Systems*, Vol. 23 No. 1, pp. 3-34, 2005.

[8] K. N. Mishra, A. Agarwal, and E. Abdelhadi, "An efficient Horizontal and Vertical method for online DNA sequence compression", *International Journal of Computer Applications*, Vol. 3, No. 1, pp. 39-46, 2010.

[9] M. Krishnamoorthy, and R. Karthikeyan, "Classification Techniques for Medicinal Databases using Auto-Regression and Firefly Algorithm", *Journal of Algebraic Statistics*, Vol. 13, No. 3, pp. 1130-1136, 2022.

[10] G. P. Arya and R. Bharti, "DNA Compression using Particle Swarm Optimization", *Journal of Advanced Research in Dynamical and Control Systems*, Vol. 05 (Special Issue), pp. 295-302, 2017.

[11] R. Gilmary and G. Murugesan, "Bit Reduction based Compression Algorithm for DNA Sequences", *International Journal of Scientific Research in Science, Engineering and Technology*, Vol. 8, No. 5, pp. 270-277, 2021.

[12] D. Mansouri and X. Yuan, "One-Bit DNA Compression Algorithm", In: *Proc. of International Conf. on Neural Information Processing*, Cambodia, pp. 376-386, 2018.

[13] G. Murugesan, "Codon Based Compression Algorithm for DNA Sequences with Two Bit Encoding", *European Journal of Molecular and Clinical Medicine*, Vol. 07, No. 10, pp. 33-41, 2020.

[14] H. Chen, "Application of Genome Sequence Based on Entropy Coding", In: *Proc. of International Conf. on Intelligent Computing*, Automation and Systems, pp. 156-159, 2020.

[15] K. Sayood, "Introduction to Data Compression", *Morgan Kaufmann Series*, Fourth Edition, Elsevier, 2007.