CHAPTER 4
COMPRESSION ALGORITHMS FOR REPETITIVE AND NON-REPETITIVE DNA SEQUENCES.

4.1: INTRODUCTION
4.2: EXISTING COMPRESSION ALGORITHMS
4.3: GENBIT COMPRESS – ALGORITHM
4.3.1: PROPOSED ALGORITHM – GENBIT COMPRESS
4.3.1.1: ANALYSIS FOR VARIOUS CASES.
4.3.2: ENCODING ALGORITHM
4.3.3: DECODING ALGORITHM
4.3.4: EXAMPLE AND COMPARISON
4.3.4.1: WORST CASE
4.3.4.2: BEST CASE
4.3.4.3: AVERAGE CASE
4.3.5: METHODOLOGY OF GENBIT COMPRESS
4.4: HUFFBIT COMPRESS – ALGORITHM TO COMPRESS DNA SEQUENCES USING EXTENDED BINARY TREES.
4.4.1. PROPOSED ALGORITHM – HUFFBIT COMPRESS
4.4.2: EXTENDED BINARY TREE
4.4.2.1: EXTENDED BINARY TREE FOR DNA SEQUENCE {A,C,G,T}
4.4.3: HUFFMAN CODING
4.4.4: ENCODE ALGORITHM
4.4.5: CONSTRUCTION OF HUFFMAN TREE TO MINIMIZE THE LENGTH OF CODED SEQUENCE:
4.4.6: DECODE ALGORITHM.
4.4.7: EXAMPLES AND CALCULATIONS
4.4.7.1: BEST CASE:
4.4.7.2: AVERAGE CASE:
4.4.7.3: WORST CASE:
4.5: CONCLUSION
4.1: Introduction

Life is strongly associated with organization and structure (Schrodinger E, 1944) with the completion of 1000 genomes project, the project is estimated to generate about 8.2 billion bases per day, with the total sequence to exceed 6 trillion nucleotide bases.

The DNA molecule is made up of a concatenation of four different kinds of nucleotides namely: Adenine, Thymine, Cytosine and Guanine (A, T, C, G). General purpose compression algorithms do not perform well with biological sequences. (Giancarlo et al, 2009) have provided a review of compression algorithms designed for biological sequences.

Finding the characteristics and comparing Genomes is a major task (Koonin EV, 1999; Wooley, 1999). In mathematical point of view, compression implies understanding and comprehension (Li et al, 1998). Compression is a great tool for Genome comparison and for studying various properties of Genomes. DNA sequences, which encode life should be compressible. It is well known that DNA sequences in higher eukaryotes contain many tandem repeats, and essentials genes (like rRNAs) have many copies. It is also proved that genes duplicate themselves sometimes for evolutionary purposes. All these facts conclude that DNA sequences should be compressible. The compression of DNA sequences is not an easy task. (Grumbach S et al, 1994, Rivals et al, 1995; Chen et al, 2000). DNA sequences consists of only four nucleotides bases {a, c, g, t}. Two bits are enough to store each base. The standard compression softwares such as “compress”, “gzip”, “bzip2”, “winzip” expanded the DNA genome file more than compressing it. Most of the existing software tools worked well for English text compression (Bell et al, 1990) but not for DNA Genomes.

Increasing genome sequence data of organisms lead DNA database size two or three times bigger annually. Thus it becomes very hard to download and maintain data in a local system. Other algorithms specifically designed for DNA sequences compression did not manage to achieve average compression rate below 1.7 bits/base. Algorithms for Compressing DNA sequences, such as Ziv-Lempel compression...
algorithms (Ziv et al, 1977, Lempeletal, 1978), Biocompress (Grumbach et al, 1993), Gencompress (Chen X et al, 2000) and DNAcompress (Chen X et al, 2002) compress DNA sequences. Their average compression rate is about 1.74 bits per base. Hence we present a new compression algorithm named “DNABIT Compress” whose compression rate is below 1.56 bits per base (for Best case) even for larger genome (nearly 2,00,000 characters).

4.2: EXISTING COMPRESSION ALGORITHMS

Most of the compression methods (Navarro G et al, 2007) used today including DNA compression fall into two categories.

- First is statistical method, (Durbin R et al, 1999) which compresses data by replacing a more popular symbol to a shorter code.
- Second is dictionary-based scheme (Cherniavsky N et al, 2004, Nevill-Manning C G et al, 1997), which compresses data by replacing long sequences by short pointer information to the same sequences in a dictionary.

In statistical methods (Ferragina P et al, 2008), arithmetic coding and CTW are known to compress the DNA data well (Matsumoto et al, 2000) and Huffman coding is known to compress not very efficiently. The Burrows-Wheeler transform (Lippert, R A, 2005) (BWT, also called block-sorting compression), (Michael Burrows and David Wheeler.) is an algorithm used in data compression techniques such as bzip2. Both the LZ77 and LZ78 algorithms work on this principle. In GS Compress, LZ77 scheme with reverse complement is introduced as a dictionary based scheme. E. Rivals (Rivals et al, 1995) another compression algorithm Cfact, which searches the longest exact matching repeat using sux tree data structure in an entire sequence. Sadeh has proposed lossy data compression schemes based on approximate string matching and proved some asymptotic properties with respect to stationary sources. In spite of the good compression ratio, arithmetic coding and CTW have disadvantages such as low decompression speed.
4.3: PROPOSED ALGORITHM – GENBIT COMPRESS

In this paper we consider the problem of DNA compression both for repetitive and non-repetitive DNA sequences. To improve the compression rate, a new technique named GenBit Compress has been devised, which is much effective with respect to compression rate. Here an encoding scheme containing 8 possible bits has been introduced. Since the DNA sequence contains only A,C,G,T letters, we named each Character as a “Base”.

The input sequence is divided into fragments, where each fragment = 4 characters. Thus in this coding scheme, 256(2 power 8 = 256) combinations can be represented. Hence every DNA segment containing four bases is replaced by a 8 bit binary number “00000000”. If the consecutive fragments are same, then a specific bit “1” is introduced as a 9th bit. If the consecutive fragments are different, then a specific bit “0” is introduced as a 9th bit to the 8 bit unique number.

GenBit Compress (RajaRajeswari et al., 2010) is a simple algorithm without Dynamic programming approach. It takes an input of a DNA sequence of length n, and divides into n/4 number of fragments. The left out individual bases (fragment length<4) is assigned 4 unique “2” bits. (A=”00”, g=”01”, c=”10”, t=”11”)

The Total number of bits per byte(ℜ) is calculated as:

\[
\mathcal{R} = \frac{9}{4} (n - \tau) + 2(\tau) - 9 (\Upsilon)
\]

Where \( n \) = length of the given sequence.
\( \tau = (n \mod 4) \), number of bases excluded from \((n \mod 4)\).
\( \Upsilon \) = Number of repetitive fragments (fragments = 4 bases {ACGT}) present in the given sequence

Compression Rate = Number of Bits/Total number of Bytes.
Separate analysis for the proposed algorithm is given for all the three cases (worst case, best case and average case).

**Worst case:** In the worst case there are no repetitive fragments and individual bases are maximum and \( \tau < 4 \). In this algorithm, the worst-case compression rate is the highest. (Compression Rate: 2.238)

**Best Case:** In this algorithm, maximum repetitive fragments are \( n/8 \) and \( \tau = 0 \). The Best-case efficiency is proved in this GenBit algorithm since its compression Rate = 1.125, which is the best among other cases.

**Average Case:** The Average case efficiency of this algorithm defines the compression rate of a typical input or a random input which is not given by neither the worst case nor the best-case efficiency. Number of fragments = \( n/16 \) and \( \tau = 2 \), Compression Rate = 1.727.

**4.3.1.1: ANALYSIS FOR VARIOUS CASES.**

**CASE 1: DNA sequence with same fragments.**

**Example 1:**
Given DNA input sequence = `aaaa aaaa`
Length of \( n = 8 \).
Assigned Unique bit number = “00000001”
The nineth bit “1” is the specific bit since the Consecutive fragments “aaaa aaaa” match with each other.

**CASE 2: DNA SEQUENCE WITH DIFFERENT FRAGMENTS.**

**Example 2:**
Given DNA input sequence = `acgt atgc`
Length of \( n = 8 \)
Assigned unique bit number = “00000000 0”
The nineth bit “0” is the specific bit since the consecutive fragments “acgt atgc” does not match with each other.

**CASE 3: DNA SEQUENCE WITH \( \tau = 0 \)**

**Example 3:**
Given DNA input sequence = agct aaaa
Length of given sequence = 8.
Number of individual bases(τ) = n mod 4 = 8 mod 4 = 0.
Assigned unique bit number = “0010100000000000”.

**CASE 4: DNA SEQUENCE WITH** \( n \mod 4 \neq 0 \) \( \tau \neq 0 \).

**Example 4:**
Given DNA input sequence = agct aaaa tt
Length of input sequence = 10.
Assigned unique bit number = “001010000000000000 1111”
The individual bases which are excluded after fragmentation, allocates “2” bits for “tt”
i.e., t=”11”.

**4.3.2: ENCODING ALGORITHM**

**Input:** Input String(INSTRING) Containing A, T, G and C

**Output:** Encoded String (OUTSTRING)

**PROCEDURE ENCODE**

**Begin**
1: Divide the given DNA sequence into fragments, where each fragment consists of 4 characters.
2: Generate all possible combinations of DNA sequence (A, C, G, T). (Since the sequence contains
4 different bases, there will be \( 4^4 = 256 \) combinations).
3: Assign unique 8 bit number ("0" & "1") to each fragment.
4: If the consecutive fragments are same, assign a specific bit “1” to the 8 bit unique number as a 9th bit.
5: If the consecutive fragments are different, assign a specific bit “0” to the 8 bit unique number as a 9th bit.
6: Repeat the steps 4 and 5 until the length of sequence is “n- τ”. (where n = length of the given sequence and τ = n mod 4)
7: Allocate unique “2” bit number to individual bases if \( \tau > 0 \).
8: Transfer the 9 bit binary number to the output String (OUTSTRING).

End

The Decryption algorithm involves the same procedure as Encryption in the reverse form.

4.3.3: DECODING ALGORITHM

**Input:** Input String

**Output:** Decoded String (DECSTRING)

**PROCEDURE DECODE**

**Begin**

1: Generate all possible combinations for \{A,C,G,T\}.
2: Allocate unique 8 bit number to each combination.
3: Divide given binary code in to 9 bit segments.
4: If 9th bit is equal to “1” , the corresponding combination is taken two times , otherwise normal.
5: Repeat step 4 , until the end of the input sequence is reached.
6: If there are any individual bases(\( \tau > 0 \)), the corresponding binary code gets transformed.
   (Assigned values for bases are : a=”00”, G=”01”, c=”10”, t=”11”).

**End**

4.3.4: EXAMPLE AND COMPARISON

The Total number of bits per byte (\( \mathfrak{R} \)) is calculated as :

\[
\mathfrak{R} = \frac{9}{4} (n - \tau) + 2(\tau) - 9(\Upsilon).
\]

where \( n = \) length of the given sequence.
\( \tau = (n \mod 4) \) , number of bases excluded from \((n \mod 4)\).
\[ \Upsilon = \text{Number of repetitive fragments (fragments = 4 bases \{ACGT\}) present in the given sequence.} \]

\textbf{4.3.4.1: Worst case:}

Consider there is no repetitive fragments (4 bases \{-A,C,G,T\}) and individual bases are maximum. i.e., \((\tau < 4)\).

\[ \mathfrak{R} = \frac{9}{4} (n - \tau) + 2(\tau) - 9(\Upsilon). \]

\textbf{Example:}

let \(n=67\)

\[ \tau = n \mod 4 \]

\[ = 3 \]

\[ = 9/4(64)+2(3)-0. \]

Total bits(\(\mathfrak{R}\)) = 150.

Compression Rate = \(\mathfrak{R}/n\)

\[ = 150/67 \]

\[ = 2.238 \text{ bits/bytes}. \]

\textbf{4.3.4.2: Best case:}

Consider there is a probability of occurrence of a maximum of \(n/8\) repetitive fragments in the given sequence. i.e., \((\tau = 0)\)

Then Total number of Bits(\(\mathfrak{R}\)) = \(9/4(n - \tau) + 2(\tau) - 9(\Upsilon)\).

\textbf{Example:}

let \(n=64\)

\[ \tau = 0 \quad \Upsilon = 8. \]

\[ \mathfrak{R} = 9/4(64)+2(0)-9(8) \]

\[ = 144-72 \]
Compression Rate = \( R/n \).

\[ = \frac{72}{64} = 1.125 \text{ bits/bytes}. \]

### 4.3.4.3: Average Case:
Consider \( \tau = 2 \) and number of fragments = \( n/16 \).

\[ R = \frac{9}{4}(64)+2(2)-9(4) \]
\[ = 144+4-36 \]
\[ = 114 \]

Compression Rate = \( R/n \)

\[ = \frac{114}{66} = 1.727 \text{ bits/bytes}. \]

### 4.3.5: METHODOLOGY OF GENBIT COMPRESS

Assume that \( n \) is the length of the sequence and fragments \( \tau = n (\text{mod } 4) \). Each fragment (ACGT) is replaced with 9 bits binary code (0 or 1). For any fragment followed by different fragment the bits assigned are “000000000”. When there is a repetitive segment followed by any segment then the value of the 9th bit (specific bit) raises to “1” otherwise “0”. Then the total number of bits required to encode the sequence of \( n \) byte can be obtained as follows:

\[ R = \frac{9}{4}(n-\tau)+2(\tau)-9(\gamma). \]

Approximately **1.125 bits per byte** is required to encode each base.

Let us consider the sequence:

```
GAAT TTGC AAAA AAAA GCTA ATGC
CTAG GGTT TTTG CCCC CCCC AAAA
TCAG TTGC ATAG GACG
```

Sequence Length = 64.
Bytes to store in a text file = 64 bytes.
Windows XP zip size = 163 bytes.
Biocompress = 14 bytes.
GenBit Compress algorithm = 9 bytes.

Thus our proposed algorithm GenBit compress has the following advantages:
i) Compression ratio of 1.125 bits per base compared to 1.76 bits per base for the other algorithms.
ii) Because the method doesn’t use dynamic programming technique which was used by other methods e.g., BioCompress, GenCompress etc, it is simple and takes less execution time.

4.4: HUFFBIT COMPRESS – ALGORITHM TO COMPRESS DNA SEQUENCES USING EXTENDED BINARY TREES.

In modern molecular biology, the genome is the entirety of an organism's hereditary information. It is encoded either in DNA or, for many types of virus, in RNA. The genome includes both the genes and the non-coding sequences of the DNA.( Ridley M. ,2006 Genome). Increasing genome sequence data of organisms lead DNA database size two or three times bigger annually. Thus it becomes very hard to download and maintain data in a local system. For a four-letter alphabet in DNA (A,C,G,T), an average description length of 2 bits per base can be assigned. For the string A,C,G,T whose sequence length is 1000, assigning A=00, C=01, G=10, T=11, requires 2000 bits of space. The storage of encoded sequence is almost double its original sequence length (Stephane,1994). For efficient storage of DNA sequences, and better compression ratio, a new algorithm(RajaRajeswari et al,2010) is proposed using the concept of Huffman Codes.
4.4.1. PROPOSED ALGORITHM – HUFFBIT COMPRESS

The proposed algorithm is a 2 way process:
(a) First construction of extended binary tree is done
(b) Second, The derived Huffman codes from Extended Binary Tree are used to calculate the number of bits in the encoded sequence length. Compression ratio is compressed size by uncompressed Size.

4.4.2 : EXTENDED BINARY TREE

An extended binary tree is a transformation of any binary tree in to a complete binary tree. This transformation consists of replacing every null subtree of the original tree with “special nodes”. The nodes from the original tree are called internal nodes (Panholzer,2002). The special nodes are called external nodes The following tree is extended binary tree. Empty circles represent internal nodes and boxes represent External nodes. Every internal node consists of exactly 2 children and every external node is a leaf(Jackson et al)

FIGURE 4.1: Extended Binary Tree
4.4.2.1: EXTENDED BINARY TREE FOR DNA SEQUENCE
\{A,C,G,T\}

Consider $\delta(x)$ be the length of a shortest path from node x to an external node in its subtree. Suppose if x is an external node, x is assumed as “0”.

If x is an internal node, $\delta$ value is:

Internal node $\delta = \min\{\delta[\tau], \delta[R]\} + 1$.

Where $\tau =$ left child of x.

and $R =$ right child of x.

Example:

Given Sequence = a, c, g, t.

length = 4.

Number of internal nodes = 3

Number of external nodes = 4.

**FIGURE 4.2:** Extended Binary tree for DNA bases A, C, G, T.
4.4.3: HUFFMAN CODING

The basic idea of Huffman coding is simple (Thomas M, 1991). We take some data we want to compress, say a list of 8 bit characters. We then create a value table where we order the characters by frequency. If we don’t know beforehand how our list of characters will look, we can order the characters by probability of occurring in the string. We then assign code words to the value table, where we assign the short code words to the most probable values. A code word is simply an n-bit integer designed in such a way there are no ambiguities or clashes with shorter code words (Chia, 2008).

Using the figure of a,c,g,t extended binary tree, By convention, bit '0' represents following the left child when tracing out the path and bit '1' represents following the right child when tracing out the path.

**FIGURE 4.3:** Bases with their Huffman Codes.
The huffman codes for A,C,G,T travelling from root to their external node.

A=0
C=10
G=110
T=111

Let “s” be a sequence of bases.

\[ F(x) \text{ is the frequency of bases, then } x \in \{a,c,g,t\} \]

\[ \text{Weighted Extended Path } = \sum_{i=1}^{n} \lambda(I) \times F(I) \]

Where

\[ \lambda(I) = \text{length of path (number of edges on path) from root to external node labeled } i. \]

\[ F(I) = \text{Frequency of occurrences of bases (a,c,g,t).} \]

Calculation of Bits in Encoded sequence =

\[ 1\times f(a) + 2\times f(c) + 3\times f(g) + 3\times f(t). \]

Where \( \lambda(I) \) for base a = 1 (since A=0)
\( \lambda(I) \) for base c = 2 (since C=10)
\( \lambda(I) \) for base g = 3 (since G=110)
\( \lambda(I) \) for base t = 3 (since T=111)

**4.4.4 : ENCODE ALGORITHM**

- Determine the different bases in the given DNA sequence (a,c,g,t) and calculate their Frequencies.
- Construct a Extended Binary Tree with minimum Weighted path (Huffman Tree).
- The External nodes of this tree are labeled by the bases in the sequence.
The weight of each External node is the frequency of the base and that is its label (A, C, G, T).

Traverse the root-to-external node paths and obtain the codes.

Replace the symbols in the sequence by their codes (Huffmans Codes).

4.4.5: CONSTRUCTION OF HUFFMAN TREE TO MINIMIZE THE LENGTH OF CODED SEQUENCE:

- Use codes from a binary tree whose external nodes correspond to the bases in the sequence
- Select the external nodes whose Weighted External Path is minimum
- The binary tree with minimum Weighted External Path for a given set of frequencies is called a Huffman tree.

**FIGURE 4.4:** Construction of Huffman Tree

(a) 

<table>
<thead>
<tr>
<th>Frequency</th>
<th>50</th>
<th>25</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
</table>

Combining the lowest frequency values.

(b) 

Combining the lowest frequency values.
4.4.6: Decode Algorithm.

Decoding is the reverse of coding (Marek, 2009). If we have a bit string, say 00011111010, we read bits until there’s a match in the table. Our example string decodes to AAATGC. Note that the code word table is designed so there are no conflicts. If the table reads A=0, C=10, G=110, T=111 and we encounter the bit 0 in a table, there’s no way we can ever get a C as the A will match all the time.
The standard method of decoding a Huffman coded string is by walking a binary tree, created from the code word table. When we encounter a 0 bit, we move left in the tree, and right when we see a 1. This is the simplest method used in our decoder.

- Decoding of binary bits leads to the original base sequence.
- The codes (0, 10, 110, 111) are unique and no code is a prefix of another.
- Consequently when the code is examined from left to right, a match with exactly one code is obtained, hence decoding is possible using Huffman’s code.
- No root-to-external node path is a prefix of another such path.
- No path code is a prefix of another path code.

4.4.7: EXAMPLES AND CALCULATIONS

4.4.7.1: BEST CASE:

Given sequence:

```
aaaaaaaaaaaaaaaaaaaaaaaaaacaaaaaaaaaaaaaaaaaaaaagaaaaaaaaaaaaaat......................
```

Sequence length = 1000.

Frequency of base “A” in the given sequence = 996
Frequency of base “C” in the given sequence = 2
Frequency of base “G” in the given sequence = 1
Frequency of base “T” in the given sequence = 1
Huffman codes for bases A=0, C=10, G=110, and T=111.

Number of Bits in Encoded sequence =

\[
1 \times f(a) + 2 \times f(c) + 3 \times f(g) + 3 \times f(t) \\
= 1 \times 996 + 2 \times 2 + 3 \times 1 + 3 \times 1 \\
= 996 + 4 + 3 + 3 \\
= 1006 \text{ bits.}
\]
\textbf{Compression Ratio} = \text{number of bits encoded} / \text{number of bases}.
\[= \frac{1006}{1000} = 1.006 \text{ bits/base}.\]

\textbf{4.4.7.2: AVERAGE CASE:}

Given sequence:
Sequence length = 90.
Frequency of base “A” in the given sequence = 50
Frequency of base “C” in the given sequence = 25
Frequency of base “G” in the given sequence = 10
Frequency of base “T” in the given sequence = 5
Huffman codes for bases A=0, C=10, G=110, and T=111.

Number of Bits in Encoded sequence
\[= 1*f(a)+2*f(c)+3*f(g)+3*f(t).\]
\[= 1*50 + 2*25 + 3*10 + 3*5\]
\[= 50+50+30+15\]
\[= 145 \text{ bits}.\]

\textbf{Compression Ratio} = \text{number of bits encoded} / \text{number of bases}.
\[= \frac{145}{90} = 1.611 \text{ bits/base}.\]

\textbf{4.4.7.3: WORST CASE:}

Let us consider the sequence:
GAAT TTGC AAAA AAAA GCTA ATGC CTAG GGTT TTTG CCCC CCCC AAAA TCAG TTGC ATAG GACG.

Sequence Length = 64.
Frequency of base “A” in the given sequence = 21
Frequency of base “C” in the given sequence = 15
Frequency of base “G” in the given sequence = 13
Frequency of base “T” in the given sequence = 15

Huffman codes for bases A=0, C=10, G=110, and T=111.

Number of Bits in Encoded sequence
\[= 1\times f(a) + 2\times f(c) + 3\times f(g) + 3\times f(t).\]
\[= 1\times 21 + 2\times 15 + 3\times 13 + 3\times 15\]
\[= 21 + 30 + 39 + 45\]
\[= 135 \text{ bits.}\]

Compression Ratio = number of bits encoded / number of bases.
\[= 135 / 64\]
\[= 2.109 \text{ bits/base}\]

4.5: CONCLUSION

A simple DNA compression algorithms which are completely new in its design is proposed to compress DNA sequences which are repetitive as well as non repetitive in nature. DNA sequence analysis i.e. single and multiple alignments are areas of active research in bioinformatics. If the sequence is compressed using proposed Compress algorithms, it will be easier to compress large bytes of DNA sequences with the compression ratio of 1.125 bits per base which will be very useful in sequence comparisons and Multiple sequence Alignment analysis. The simplicity and flexibility of proposed Compress algorithms could make it an invaluable tool for DNA compression in clinical research.