‘DNA as a Storage Device: Binary to Quaternary’

FOR

Dr. Sumit Biswas
(Department of Biological Sciences)

BY

Name of the Student
Neel Bhandari

ID Number
2018B1A70084G

In the partial fulfillment of the requirements of
BIO F266 - Study Oriented Project

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
27 April 2020

Table of Contents

1. Introduction

This report describes a system with possible modifications and trade-offs for an efficient system to store large chunks of data in a base-level hierarchy DNA based storage. Here we will discuss various methods towards achieving this goal. Our goal was to compress data efficiently without compromising on its reliability and then transform the data to nucleotide sequences.

1.1 Text Data

"Here lies the Tengwar inscription in the Black Speech of Mordor Three Rings for the Elvenkings under the sky Seven for the Dwarflords in their halls of stone Nine for Mortal Men doomed to die One for the Dark Lord on his dark throne In the Land of Mordor where the Shadows lie One Ring to rule them all One Ring to find them One Ring to bring them all and in the darkness bind them in the Land of Mordor where the Shadows lie"

1.2 Binary Data - Raw Conversion

```
01001000 01100101 01110010 01100101 00100000 01101100 01101001 01100101 01110011 00100000 01110100 01101000 01100101
00100000 01010100 01100101 01101110 01100111 01110111 01100001 01110010 00100000 01101001 01101110 01110011 01100011
01110010 01101001 01110000 01110100 01101001 01101111 01101110 00100000 01101001 01101110 00100000 01110100 01101000
01100101 00100000 01000010 01101100 01100001 01100011 01101011 00100000 01010011 01110000 01100101 01100101 01100011
01101000 00100000 01101111 01100110 00100000 01001101 01101111 01110010 01100100 01101111 01110010 00100000 01010100
01101000 01110010 01100101 01100101 00100000 01010010 01101001 01101110 01100111 01110011 00100000 01100110 01101111
01110010 00100000 01110100 01101000 01100101 00100000 01000101 01101100 01110110 01100101 01101110 00101101 01101011
01101001 01101110 01100111 01110011 00100000 01110101 01101110 01100100 01100101 01110010 00100000 01110100 01101000
01100101 00100000 01110011 01101011 01111001 00101100 00100000 01010011 01100101 01110110 01100101 01101110 00100000
01100110 01101111 01110010 00100000 01110100 01101000 01100101 00100000 01000100 01110111 01100001 01110010 01100110
00101101 01101100 01101111 01110010 01100100 01110011 00100000 01101001 01101110 00100000 01110100 01101000 01100101
01101001 01110010 00100000 01101000 01100001 01101100 01101100 01110011 00100000 01101111 01100110 00100000 01110011
01110100 01101111 01101110 01100101 00101100 00100000 01001110 01101001 01101110 01100101 00100000 01100110 01101111
01110010 00100000 01001101 01101111 01110010 01110100 01100001 01101100 00100000 01001101 01100101 01101110 00100000
01100100 01101111 01101111 01101101 01100101 01100100 00100000 01110100 01101111 00100000 01100100 01101001 01100101
00101100 00100000 01001111 01101110 01100101 00100000 01100110 01101111 01110010 00100000 01110100 01101000 01100101
00100000 01000100 01100001 01110010 01101011 00100000 01001100 01101111 01110010 01100100 00100000 01101111 01101110
00100000 01101000 01101001 01110011 00100000 01100100 01100001 01110010 01101011 00100000 01110100 01101000 01110010
01101111 01101110 01100101 00100000 01001001 01101110 00100000 01110100 01101000 01100101 00100000 01001100 01100001
01101110 01100100 00100000 01101111 01100110 00100000 01001101 01101111 01110010 01100100 01101111 01110010 00100000
01110111 01101000 01100101 01110010 01100101 00100000 01110100 01101000 01100101 00100000 01010011 01101000 01100001
01100100 01101111 01110111 01110011 00100000 01101100 01101001 01100101 00101110 00100000 01001111 01101110 01100101
00100000 01010010 01101001 01101110 01100111 00100000 01110100 01101111 00100000 01110010 01110101 01101100 01100101
00100000 01110100 01101000 01100101 01101101 00100000 01100001 01101100 01101100 00101100 00100000 01001111 01101110
01100101 00100000 01010010 01101001 01101110 01100111 00100000 01110100 01101111 00100000 01100110 01101001 01101110
01100100 00100000 01110100 01101000 01100101 01101101 00101100 00100000 01001111 01101110 01100101 00100000 01010010
01101001 01101110 01100111 00100000 01110100 01101111 00100000 01100010 01110010 01101001 01101110 01100111 00100000
01110100 01101000 01100101 01101101 00100000 01100001 01101100 01101100 00100000 01100001 01101110 01100100 00100000
01101001 01101110 00100000 01110100 01101000 01100101 00100000 01100100 01100001 01110010 01101011 01101110 01100101
01110011 01110011 00100000 01100010 01101001 01101110 01100100 00100000 01110100 01101000 01100101 01101101 00100000
01101001 01101110 00100000 01110100 01101000 01100101 00100000 01001100 01100001 01101110 01100100 00100000 01101111
01100110 00100000 01001101 01101111 01110010 01100100 01101111 01110010 00100000 01110111 01101000 01100101 01110010
01100101 00100000 01110100 01101000 01100101 00100000 01010011 01101000 01100001 01100100 01101111 01110111 01110011
00100000 01101100 01101001 01100101 00101110
```

2. Encoding and Compression

We will discuss various compression softwares and encodings that have been experimented upon to give the best possible compression ratios and trade-offs with redundancy and security.

2.1 Compression algorithms

1. **Gz Compression**: Gzip is a data compression algorithm capable of compression and decompression of data on the fly. It is a lossless compression.

   Ratio for sample text: ***1.79***

2. **Bzip2 Compression:** Bzip2 is an open source compression software using the Burrows-Wheeler Transform. This compression is only a data compressor and not a file archiver. It is a lossless compression.

   Ratio for sample text: ***1.60***

3. **DEFLATE Compression:** DEFLATE is a lossless compression algorithm that is a combination of LZSS and Huffman coding. This algorithm has various implementations for many different file types including 7-zip and is hence extensively used.

   Ratio for sample text: ***1.88***

4. **LZW compression:** Lempel-Ziv-Welch compression is a lossless compression mostly used in graphic representation in GIF and optionally in PDF format.

   Ratio for sample text: ***1.68***

5. **Huffman Encoding:** Huffman coding gives a key-value table that can be used for an optimal and lossless compression of data. It bases it's key-value table on the entropy of the data and hence knowing the data beforehand is important. It is a very common coding and is widely used.

   Ratio for sample text: ***2.0015***

2.2 Encoding Types

1. **2 window repeats :** The most comfortable and convenient way of encoding binary data to ATGC repeats. We take two binary values at a time and map them to A, T, G, C, respectively. We could take larger window sizes, but it wouldn't fit in the ATGC format as we have only 4 bits to work with 00,01,10,11.

   Effectively this provides a compression ratio of **2**.

2. **Goldman encoding** : This method provides redundancy to the data, making it more robust for storage, thus increasing its viability. In this method, we use overlapping strands so that if one strand is faulty, it can be corrected by multiple other strands. The compression ratio for this is <1 as we increase the data to increase robustness. We need to evaluate the marginal return on increasing strand length to understand the ideal point of overlap length vs robustness.

   The compression(rather extension) in this case is tunable by adjusting the strand overlap.

3. **XOR encoding** : This method is also an alternative to increasing data redundancy. In it we take 2 independent strands and perform an XOR operation on them, this creates a third strand. The unique property of XOR is that if we now XOR any of the 2 strands, it generates the third strand. 2 or more strands having faults is unlikely, thus this method provides us with an method to cross check the data while it's being used.

   The compression ratio in this case will be **0.67** if we take all distinct pairs or $\frac{n}{(nC2)+n}$ if we use all pairs from all strands.

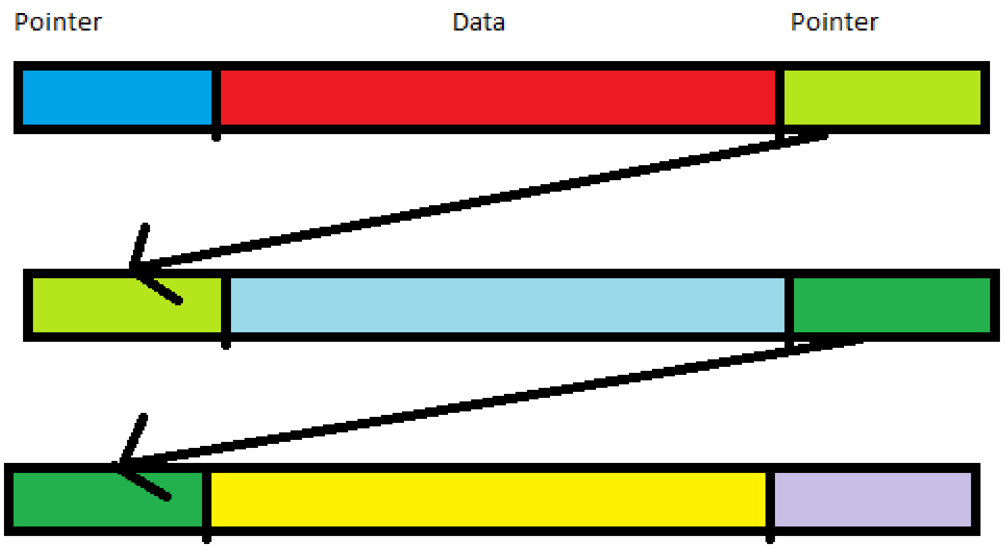
4. **Ternary encoding** : This is an encoding method which is used to prevent "AAGGTTT" repeats in the sequence so we get something like "ATGCGCT". This increases the Data Size but prevents errors during synthesis. As discussed before, we most likely won't be using this.

   The compression ratio in this case is **0.75**.

3. Synthesis

After encoding our data into DNA nucleotide sequence we move onto synthesis.

While synthesizing our DNA strands of the converted data there are a few limitations that arise. Namely, limiting strand length with accuracy (limited to a maximum of 200 nucleotides) and addition of pointers at the start of sequence.To overcome this problem we break our data into smaller fragments and attach a unique primer and pointer combination(this will help during Sequencing PCR Run).



This method also allows for continuity recognition within a file or a larger package and allows for unique file distinguishability.

Synthesis of DNA can be done via column-based oligo synthesis or array-based oligo synthesis.

Array-based oligo synthesis is a cost effective method with those being produced by microarrays being significantly cheaper than column-based synthesis. However array-based synthesis is currently plagued with a higher number of error rates than column-based synthesis which primarily suffers from single-base deletions.

4. Discussion

In this section we go over the techniques of encryption as well as storage which will improve the storage method's security and robustness/viability respectively.

4.1 Encryption of data

We also explore the possibility of encrypting the data stored in DNA by the methods of Laplace transforms as described by the "**DSWLT" method** in a paper published in the International Journal for Computer Applications (Som and Som, 2012) and another related paper in the same journal describing the method in detail (Dhingra et al., 2016).

4.1.1 Method of Encryption

Step 1: Select the message, M, to be sent, and convert into suitable ATGC format via binary format, M_{bin}.

Step 2: DNA coded text is converted into Integer coded text, M_{Int} using the Lookup table mapping numeric value to base nucleotide as given in table 4.

Step 3: Each integer in M_{Int} is used as the coefficients of the Laplace transform of the function

f(t) = $\boldsymbol{Gte^t}$ i.e. $\mathbf{L\{f(t)\}= L\{} \sum_{n=0}^{\infty} G_n \frac{t^{n+1}}{n!}\}$, where $G_n \geq 0 \ \forall \ n \geq 8$.

Step 5: The coefficients of $\mathbf{L\{} \sum_{n=0}^{\infty} G_n \frac{t^{n+1}}{n!}\} = \sum_{n=0}^{\infty} \frac{C_n}{s^{n+2}}$ **i. e. Cn** $\forall$ **n** ≥ 0 are taken M_{Lap} is constructed by storing C_n mod 128.

Step 6: Each integer of M_{Lap} is converted to its corresponding ASCII values (7 bit binary equivalents) and on them cumulative XOR as shown in Fig. 3 is performed. The results are stored in M_{XOR}.

Step 7: The ASCII equivalent of the values in M_{XOR} are stored as the cipher text, C.

4.1.2 Method of Decryption

Step 1: The ciphertext C is converted to its corresponding ASCII values (7 bit binary equivalent), C_{ASC}.

Step 2: Cumulative XOR operation is performed and resultant binary streams are converted back to their equivalent decimal form, C_{Lap} to get the coefficients of the Laplace transform back.

Step 3: Inverse Laplace transform is applied over C_{Lap} producing the coefficients which are considered as the Integer codes corresponding to DNA bases, C_{Int}.

Step 4: The Integer codes in C_{Int} are mapped back with DNA bases using table 4 which produces M_{dna}.

Step 5: The DNA bases in M_{dna} are mapped back to their binary codes using table 3 to for M_{Bin}.

Step 6: The binary streams M_{Bin} in are converted to their corresponding ASCII values and hence producing the original message or the plain text M.

4.2 Storage

4.2.1 Storage of information in-vivo

It is unlikely that in-vivo storage of data could ever replace traditional data storage media (electronic) due to its large size (bacterial cell), low density of data stored and additional cost-hefty steps. However, storage of information in-vivo opens up possibilities in the realm of using these cells as "molecular tapes" storing information about various molecular processes and evolutionary history and saving this information for read processes at a later time.The most effective methods of storing in-vivo are in the genome or in a plasmid (ideal).

4.2.2 Storage of information in-vitro

Storage of information in-vitro provides high density of data, easy sequencing and cost-effective methods of storing data that will last longer than any silicon-based electronic storage device. This makes it an ideal candidate for providing long-term, easily accessible, encrypted and safe methods for storing data. Storage of DNA in-vitro is cheap and easy with various ways depending on importance of information and storage duration:

- -164° C or in dried state (Glassy state) (Long term storage - decades)
- -80° C as precipitate under ethanol (Long term storage - years)
- -80° C in tris-EDTA (Medium term storage - months)
- 4° C in tris-EDTA (Short term storage - weeks)

4.3. Sequencing

Most of the active and commercially available sequencing machines use a method known as "sequencing by synthesis". The strand of interest i.e encoded strand serves as a template for the polymerase. Fluorescent nucleotides are used here and are captured after each round thus giving us a final sequence.

The most popular sequencers available in the market are provided by Illumina and are cost-effective.

5. Experiments and Methods

For our needs and purposes, we found that the optimal solution to the Compression and Encoding problem was to use Huffman coding and 2-window repeats. The steps we followed to attain suitable compression and data format are mentioned below step by step and in-depth.

5.1 Original Text Data

We used an arbitrary text sample, from the popular book, “The Lord of the Rings: The Fellowship of the Ring” by J. R. R. Tolkien.

“Here lies the Tengwar inscription in the Black Speech of Mordor Three Rings for the Elvenkings under the sky Seven for the Dwarflords in their halls of stone Nine for Mortal Men doomed to die One for the Dark Lord on his dark throne In the Land of Mordor where the Shadows lie One Ring to rule them all One Ring to find them One Ring to bring them all and in the darkness bind them in the Land of Mordor where the Shadows lie”

5.2 Huffman Encoding[1]

We use Huffman Coding to attain a compression ratio of ~2 from text to binary. It generates unique key-value pairs based on the frequency of letters in the input data.

5.2.1 Huffman Key-Value Table (data specific):

‘ ’=00 E=011 H=1000 R=1011 I=0101 T=1001 N=1100

O=1101 D=11111 M=01000 A=10101 L=11100 S=11101

W=010010 G=101001 K=010011 F=111101 C=1010000 B=1010001

P=11110001 V=11110010 U=11110011 Y=11110000

[1]URL = < https://github.com/DevChuriwala/DNA_Data_Storage>, contains code to convert binary data to non-repeating ATGC format - via conversion to ternary system.

5.2.2 Huffman Binary Data:

```
1000011101101100111000101011111010010011000011001001011110010100101001010101
1011000101110011101101000010110101111100011001010111011100000101110000100110
0001100101000111100101011010000010011001110111110001011011101000010000011011
1110100010001101101111111110110110010011000101101101100101101011100101001111
0100111101110110110010011000011000111110011110010011110001001101011100101001
1110100111100111100111110111011001001100001100111010100111111000000111010111
1110010011110000111101110110110010011000011001111101001010101101111110111100
1101101111111111010001011100001001100001101011011001000101011110011100111010
0110111110100111011001110111000110011000101110001100111101110110110001000110
1101110011010111100000100001111000011111110111010100001111111001001110100111
1101010110011011100011001111011101101100100110000110011111101011011010011001
1100110110111111100110111000010000101111010011111101011011010011001001100010
1111011100011000101110000100110000110011100101011100111110011011111010001000
1101101111111110110110001001010000111011011001001100001100111011000101011111
1110101001011101001110001010110011011100011001011010111001010010010011101001
01111110011111000110010011000011010000001010111100111000011011100011001011010
1110010100100100111010011110101011100111110010011000011010000011011100011001
0110101110010100100100111010010100011011010111001010010010011000011010000010
1011110011100001010111001111100010111000010011000011001111110101101101001111
0001111101111010010100010101110011111001001100001101000000101110000100110000
1100111001010111001111100110111110100010001101101111111110110110001001010000
11101101100100110000110011101100010101111111101010010111010011100010101 1
```

This transforms a 3400 bit data to a 1668 bit data, giving a compression ratio of ***2.038***.

5.3 2-Window Repeats[2]

Using 2-window repeats, we get a sequence of characters 834 nucleotides long from an original sequence which was 1668 bits long. Therefore giving a ratio of **2** and a combined compression ratio of $2.038 \times 2 =$ **4.076**.

GATCTGCACGAGGCCGGTACAACAGTTCGTTAGGTTTTGCATTCACGCTAAGCTTCCATGTTTCTCAATTCA
AGTGATGTTATCGTTTGGAAGTGTCTCCATTGCGGATAATGCCGGAGACTGCCCCGCTGTACATTGCTGTTGG
CGTTACCTACCTCTGCAGTGATGACCGTCGTACCATACTTCAGGTCGGTCGTCGTCCTCTGTACAACACGGGT
CCGAATCTTCCGTACCAACCTCTGCAGTGATGTCCTAGGGCTCCGCCACTGCCCCCTATTCAAGTGATGGCTG
TATTTCGTCACGGTGCCGGTCTGTCTCATGTGAGCGACACCTCTGCATATGCTCACTTCGAAGATCGATCCCT
CTTAACCCGTACGGTCCTTTGTGCGACACCTCTGCAGTGATGTCCGGCTGGTGTCACTGCCCGTGCGATAAGC
CTACCCTTGCTACAGTGAGCCTCATGAGCGATACAACACGTTTCACCGTGCCGGAGACTGCCCCGCTGAGTTA
ACGCTGTACAACACGCATTTCCCGGGTTCTACGAGGCACTCATGTTGGCGTTAGTACGGTTCCGTCCATGTAC
AACTAATTTCGTCAACTCATGTTGGCGTTAGTACGGTCGGGCGTCCAGTGATGGAACTCATGTTGGCGTTAG
TACGGTTATGCTTCAGGTAGTGATGGAAGGCCACGATTTCACCGAGCGATACAACACCCTTGCTACCATCCTC
GGTTATTTCACCGTACAACTAAAGCGATACAACACGTTTCACCGTGCCGGAGACTGCCCCGCTGAGTTAACGC
TGTACAACACGCATTTCCCGGGTTCTACGAGGC

[2] URL = < https://github.com/DevChuriwala/DNA_Data_Storage>, contains code to convert binary data to ATGC format using 2 window repeats.

5.4 Discrete DNA Strands

We then create discrete DNA strands based on the explanation in the Synthesis section. For this 834 bit data we require 7 strands which includes 140 bits of data + 10 bits of pointer information (marked in yellow).

For the aforementioned data we generated:

a. GATCTGCACGAGGCCGGTACAACAGTTCGTTAGGTTTTGCATTCACGCTAAGCTTCCATGT TTCTCAATTCAAGTGATGTTATCGTTTGGAAGTGTCTCCATTGCGGATAATGCCGGAGACT GCCCCGCTGTACATTGCTGTTGGCGTTA

b. GTTGGCGTTACCTACCTCTGCAGTGATGACCGTCGTACCATACTTCAGGTCGGTCGTCGTCC TCTGTACAACACGGGTCCGAATCTTCCGTACCAACCTCTGCAGTGATGTCCTAGGGCTCCGC CACTGCCCCCTATTCAAGTGATGGCT

c. AGTGATGGCTGTATTTCGTCACGGTGCCGGTCTGTCTCATGTGAGCGACACCTCTGCATAT GCTCACTTCGAAGATCGATCCCTCTTAACCCGTACGGTCCTTTGTGCGACACCTCTGCAGTG ATGTCCGGCTGGTGTCACTGCCCGTGC

d. CTGCCCGTGCGATAAGCCTACCCTTGCTACAGTGAGCCTCATGAGCGATACAACACGTTTCA CCGTGCCGGAGACTGCCCCGCTGAGTTAACGCTGTACAACACGCATTTCCCGGGTTCTACG AGGCACTCATGTTGGCGTTAGTACGGT

e. TTAGTACGGTTCCGTCCATGTACAACTAATTTCGTCAACTCATGTTGGCGTTAGTACGGTCG GGCGTCCAGTGATGGAACTCATGTTGGCGTTAGTACGGTTATGCTTCAGGTAGTGATGGAA GGCCACGATTTCACCGAGCGATACAAC

f. GCGATACAACACCCTTGCTACCATCCTCGGTTATTTCACCGTACAACTAAAGCGATACAACA CGTTTCACCGTGCCGGAGACTGCCCCGCTGAGTTAACGCTGTACAACACGCATTTCCCGGG TTCTACGAGGC

The number of strands will increase with an increase in the amount of data and the pointers allow the data to be quickly and easily accessible. We have limited our strands to 150 bits, but advancements in technology may allow us to use higher bits/strands in the future.

Work Cited

Bornholt, James, et al. “A DNA-Based Archival Storage System.” *IEEE Micro*, 2017, pp. 1–1., doi:10.1109/mm.2017.264163456.

Ceze, Luis, et al. “Molecular Digital Data Storage Using DNA.” *Nature Reviews Genetics*, vol. 20, no. 8, 2019, pp. 456–466., doi:10.1038/s41576-019-0125-3.

Dhingra, Swati, et al. “Laplace Transformation Based Cryptographic Technique in Network Security.” *International Journal of Computer Applications*, vol. 136, no. 7, 2016, pp. 6–10., doi:10.5120/ijca2016908482.

Goldman, Nick, et al. “Towards Practical, High-Capacity, Low-Maintenance Information Storage in Synthesized DNA.” *Nature*, vol. 494, no. 7435, 2013, pp. 77–80., doi:10.1038/nature11875.

Kosuri, Sriram, and George M Church. “Large-Scale De Novo DNA Synthesis: Technologies and Applications.” *Nature Methods*, vol. 11, no. 5, 2014, pp. 499–507., doi:10.1038/nmeth.2918.

Som, Sukalyan, and Moumita Som. “DNA Secret Writing with Laplace Transform.” *International Journal of Computer Applications*, vol. 50, no. 5, 2012, pp. 44–50., doi:10.5120/7771-0853.