Compressed DNA Coding using Minimum Variance Huffman Tree

Article *in* IEEE Communications Letters · April 2020

DOI: 10.1109/LCOMM.2020.2991461

CITATIONS
10

READS
155

4 authors:

Pooja Mishra
Indian Institute of Technology (ISM) Dhanbad
8 PUBLICATIONS **27** CITATIONS
SEE PROFILE

Chiranjeev Bhaya
Indian Institute of Technology (ISM) Dhanbad
6 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Arup Kumar Pal
Indian Institute of Technology (ISM) Dhanbad
103 PUBLICATIONS **1,399** CITATIONS
SEE PROFILE

Abhay Kumar Singh
Indian Institute of Technology (ISM) Dhanbad
64 PUBLICATIONS **369** CITATIONS
SEE PROFILE

Compressed DNA Coding Using Minimum Variance Huffman Tree

Pooja Mishra, Chiranjeev Bhaya, Arup Kumar Pal, and Abhay Kumar Singh

***Abstract*— DNA data storage is a highly emerging technology of storing large amount of data in a small volume for a long period of time. However, synthesis of DNA sequences come with a cost that depends on the number of nucleotides present in it. An efficient algorithm to store large amount of data in small number of nucleotides has been proposed which uses minimum-variance Huffman coding. The DNA sequences generated follow GC-constraint and run-length constraint of at most 1. Texts have been stored in lossless manner. Images have been stored in both lossless and lossy manner. In either of the cases, a high code-rate has been attained, thus implying good compression and reduction in cost of synthesis.**

***Index Terms*— Compression, data storage, DNA, Huffman coding.**

I. Introduction

IN THIS era of high-end technological advancement, the production of digital data has been increasing exponentially day by day. According to reports from International Data Corporation (IDC) [1], digital data will reach a mark of 175 zettabytes by 2025. This huge volume of data is not easy to store, manage and protect for a long time. Conventional techniques of storing data like magnetic tape, optical disks, cloud-based storage and so on are voluminous and volatile to external environment.

One of the forthcoming techniques to store data is in form of DNA molecules. DNA stands for deoxyribonucleic acid, which contains the genetic information of almost all living organisms [2]. Located mostly in the cell nucleus and some in mitochondria, these small molecules can store tremendous amount of data that remains intact for thousands of years [2]. It is estimated that one gram of DNA can store around 700 terabytes of data [3]. Owing to these properties, DNA can be efficiently used as a storage medium for huge volumes of data.

A DNA molecule is made up of four types of nitrogen bases (also called nucleotides), namely, Adenine (A), Guanine (G), Cytosine (C) and Thymine (T) attached with phosphate and pentose-sugar. The sequence of these nitrogen bases hold the information in a DNA molecule. However, two important constraints are followed [2].

- *GC-Constraint* - The number of G and C nucleotides in the DNA sequence must be nearly half of the total length of the entire DNA sequence. This ensures lower sequencing-error rates and are easier to synthesize [4].
- *Run-Length Constraint* - Run-length is the repetition of same nucleotide consecutively. A DNA sequence having long run-lengths cause both sequencing and synthesis problems [5].

The main idea behind DNA data storage is to map a digital binary data into a sequence of DNA nucleotides by following the above mentioned constraints. Once the sequence is obtained, it can be synthesized into DNA dust. The DNA dust can be stored for a long period of time, which can be later read and converted back to digital information vis High Throughput Sequencing (HTS) [6]. Since the cost of synthesis is quite high, an efficient encoding algorithm needs to be provided which can store high amount of data in less number of nucleotides. Data can be compressed and stored in either lossless or lossy format. In lossless storage of data, the entire data is preserved and can be retrieved exactly same as that of the original one, which is a convenient approach for text and high-quality image data. In lossy approach, only the significant components of the data are stored and an approximate data can be retrieved which is not exactly same as that of original data but good enough to preserve the necessary information. Images can be stored in lossy approach.

Pioneered by Church *et al.* [3], they presented a seminal paper on DNA based data storage in which the data was converted into 8 bit ASCII and every 0 was replaced with A or C and 1 with G or T, disallowing homopolymer runs of length greater than 3 and preserving the GC-constraint. Goldman *et al.* [7] compressed the raw data using ternary Huffman Coding and then applied differential coding to the obtained sequence. It maintained the run-length constraint with run-length of at most 1, giving a better code-rate. However, GC-constraint was not preserved. Grass *et al.* [8] proposed a method that combined both the data encoding process and error correcting mechanism using Reed-Solomon codes. Blawat *et al.* [9] proposed a forward error-correction scheme so that correct codewords could be retrieved even after substitution, insertion or deletion errors. Yaniv Erlich and Dina Zielinski [10] used Luby transform to design DNA fountain that would produce DNA strings following run-length constraint, GC-constraint and a minimum Hamming distance. Song *et al.* [11], Wang *et al.* [12], and Kees A Schouhamer Immink and Kui Cai [13] proposed mathematical methods for generating codes preserving run-length and GC-constraints. Limbachiya *et al.* [14] proposed an altruistic algorithm that generates DNA codewords of a given length, each having a minimum specified Hamming distance with each other so that all the codewords followed run-length of at most one and the GC-constraint.

Manuscript received February 18, 2020; revised March 26, 2020; accepted April 23, 2020. Date of publication April 30, 2020; date of current version August 12, 2020. The associate editor coordinating the review of this letter and approving it for publication was M. Egan. *(Corresponding author: Arup Kumar Pal.)*

Pooja Mishra, Chiranjeev Bhaya, and Arup Kumar Pal are with the Department of Computer Science and Engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad 826004, India (e-mail: arupkrpal@gmail.com).

Abhay Kumar Singh is with the Department of Mathematics and Computing, Indian Institute of Technology (Indian School of Mines), Dhanbad 826004, India.

Digital Object Identifier 10.1109/LCOMM.2020.2991461

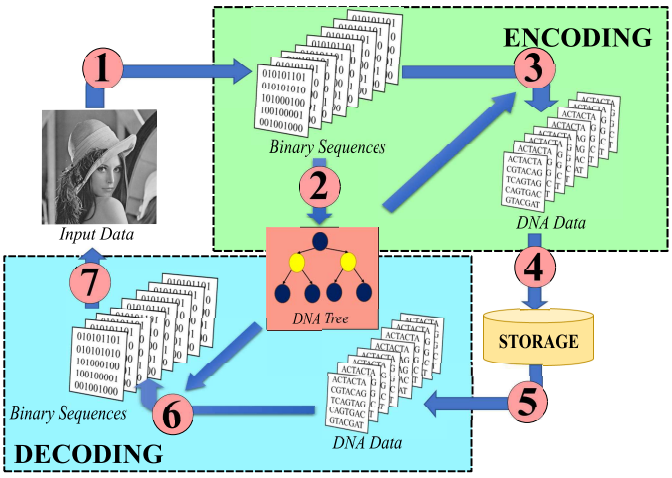


Fig. 1. Proposed Method : 1. The input data is first converted to binary sequence, 2. DNA Tree is created using Minimum Variance Huffman coding, 3. Using DNA Tree and binary sequence, DNA data is obtained by the encoding algorithm, 4. DNA sequences are stored, 5. Stored DNA data is recovered, 6. Using the recovered DNA data and the same DNA Tree, binary sequences are recovered using the decoding algorithm, 7. The original input data is recovered from the recovered binary sequences.

In this letter, an efficient method of compression of digital data into DNA sequences by using minimum-variance Huffman tree has been proposed. The proposed algorithm interleaves compression and constraint coding, where the generated DNA sequences preserve the GC-constraint and run-length constraint of at most one and also gives a high code-rate, thus allowing large amount of information to be stored using small number of nucleotides.

II. Prerequisites

A. Huffman Codes

Huffman codes are a class of prefix codes that use Huffman algorithm for its design. Huffman codes are compact codes, i.e., for a given source input, it produces the code having minimum average length. The algorithm for construction of Huffman codes has been described in [15]. All source symbols are sorted according to their probabilities of occurrence. Symbols having lower probabilities are placed farther from the root. Encoded symbols are determined by the path of the node from the root.

B. Minimum Variance Huffman Tree

Definition 1 (Variance of Huffman Tree): [16] The variance σ^2 of a Huffman tree containing n leaf nodes corresponding to each symbol of the source, the probability of their occurrence being p_i and their distance from the root being l_i is given as $\sigma^2 = \sum_{i=1}^{n} p_i(l_i - \bar{R})^2$ where $\bar{R}$ is the average length of each codeword and is given by $\bar{R} = \sum_{i=1}^{n} p_i l_i$

Definition 2 (Minimum Variance Huffman Tree): [16] Out of all possible Huffman trees that can be drawn from a source, the tree having minimum variance is called the minimum variance Huffman Tree.

C. Code Rate

Definition 3: Let b be the number of bits used to represent the original data and n be the total number of DNA nucleotides used to store the same data in the form of DNA without implementation of any additional redundant information for error-correction, indexing, addressing, etc. Then code rate $\mathcal{R}$ is defined as $\mathcal{R} = \frac{b}{n}$ bits/nt (bits per nucleotide).

Code rate is a measurement of how many bits of the original data can be represented by a single DNA nucleotide. Higher code-rate implies more compression of data and reduction in cost of synthesis and sequencing.

III. Proposed Method

In this method, the concept of minimum variance binary Huffman tree has been used to create a DNA sequences of nucleotides which preserve the constraints.

A. Algorithm

The first step for compressing data into DNA sequence is obtaining the binary equivalent of the input data. For text data, each character is converted to its corresponding 8-bit ASCII value as described in Algorithm 1. Images are decomposed into their bit-planes. Since every pixel is composed of 8-bits, eight bit-planes will be obtained corresponding to every bit of every pixel, as described in Algorithm 2. Images can be stored either in lossless or lossy manner. For lossless storage, all the eight bit planes are stored, whereas for lossy storage, the last two bit-planes are dropped out since they contain very less visual information and the image can be reconstructed back without much perceptual loss.

Algorithm 1 Text to Binary

Input : Text file T containing ASCII characters
Output: Binary file B

for *every character* c *in* T **do**
  $v = \texttt{ASCII}\ (c)$
  $b = \texttt{binary}\ (v)$
  Append b to B
return B

Algorithm 2 Image to Binary

Input : Grayscale Image I, each pixel having depth 8-bits
Output: 8 Bit-planes $P_1, P_2, \ldots, P_8$

for *every pixel* p *in* I **do**
  $b[1 \ldots 8] = \texttt{binary}\ (p)$
  for $i = 1$ **to** 8 **do**
    Append b_i to P_i
return $P_1, P_2, \ldots, P_8$

After obtaining the binary data, the corresponding DNA tree needs to be constructed which is required for both encoding and decoding process. The binary data is divided into non-overlapping blocks of length $blen$. Taking each block as a source symbol, the corresponding minimum-variance Huffman tree $\mathcal{T}$ is constructed. Starting from the root, the edges to

the left child nodes are alternately labeled with A and G and the right ones with T and C. A complementary tree $\mathcal{T}'$ is constructed by replacing A with G, T with C, G with A and C with T in $\mathcal{T}'$. Codebooks $\mathcal{C}$ and $\mathcal{C}'$ are created from trees $\mathcal{T}$ and $\mathcal{T}'$ respectively as described in Algorithm 3.

Algorithm 3 Binary to DNA tree

Input : Binary data B
Output: Labeled Binary Trees $\mathcal{T}$ and $\mathcal{T}'$, and codebooks $\mathcal{C}$ and $\mathcal{C}'$

Divide the binary data into non-overlapping blocks of length $blen$
Create a minimum variance Huffman tree $\mathcal{T}$, taking each block as source symbol
for *every edge* $(u, v) \in \mathcal{T}$ **do**
  if $v == u$.left **then**
    if u *is at even distance from* $\mathcal{T}$.root **then**
      $(u, v) = A$
    else
      $(u, v) = G$
  else
    if u *is at even distance from* $\mathcal{T}$.root **then**
      $(u, v) = T$
    else
      $(u, v) = C$
Create $\mathcal{T}' = \mathcal{T}$
for *every edge* $(u, v) \in \mathcal{T}'$ **do**
  if $(u, v) == A$ **then**
    $(u, v) = G$
  else if $(u, v) == G$ **then**
    $(u, v) = A$
  else if $(u, v) == C$ **then**
    $(u, v) = T$
  else if $(u, v) == T$ **then**
    $(u, v) = C$
Create codebooks $\mathcal{C}$ and $\mathcal{C}'$ from trees $\mathcal{T}$ and $\mathcal{T}'$ respectively.
return $\mathcal{T}, \mathcal{T}', \mathcal{C}, \mathcal{C}'$

For encoding, input data is read block-by-block. Starting with codebook $\mathcal{C}$, corresponding DNA sequence is obtained. If the sequence is of odd length, a switch is made to the complement codebook to ensure the preservation of run-length and GC-constraint. The decoding algorithm uses the same DNA tree and traverses from the root according to the sequence read character-by-character from the DNA string. On encountering a leaf node, the corresponding value gives the decoded block information. Similar to encoding algorithm, a switch is made to the complementary tree if the path from the root to the leaf node occurs to be of odd length. The encoding and decoding algorithms are described in Algorithm 4 and 5 respectively.

B. Deductions

Theorem 1: Let $\mathcal{T}$ be the minimum variance Huffman tree for a given source and $\mathcal{T}'$ be its complement tree. The generated DNA sequence produced always has a run-length of at most 1.

Algorithm 4 Encoding

Input : Binary data B and codebooks $\mathcal{C}$ and $\mathcal{C}'$ generated by Algorithm 3 for B
Output: DNA sequence D

current_codebook $= \mathcal{C}$
for *every block* b *of length* $blen$ *in* B **do**
  $c =$ codeword from current_codebook for b
  if length $(c) ==$ *odd* **then**
    if current_codebook $== \mathcal{C}$ **then**
      current_codebook $= \mathcal{C}'$
    else
      current_codebook $= \mathcal{C}$
  Append c to D
return D

Algorithm 5 Decoding

Input : DNA Sequence D and corresponding DNA tree $\mathcal{T}$ and $\mathcal{T}'$
Output: Binary sequence B

current_tree $= \mathcal{T}$
node $=$ current_tree.root
for *every nucleotide* n *in* D **do**
  node $=$ Traverse (node $, n$)
  if node *is a leaf node* **then**
    Append the symbol associated with node to B
    if *distance from* current_tree.root *to* node *is odd* **then**
      if current_tree $== \mathcal{T}$ **then**
        current_tree $= \mathcal{T}'$
      else
        current_tree $= \mathcal{T}$
return B

Proof: In the Huffman tree $\mathcal{T}$ created using the proposed algorithm, the paths from the root to the leaf nodes are labeled alternatively with (A, T) and (G, C) sequences. So, it is obvious that the codewords derived by traversing from root to the leaf do not have any run-length.

The tree $\mathcal{T}$ starts with A or T bases. If a leaf node is present at an even distance from the root, the corresponding codeword will end with G or C. The next symbol will be decoded from root of the same tree $\mathcal{T}$, hence, will begin with A or T, thus guaranteeing no run-length among adjacent codewords. Similar analogy holds for the complement tree $\mathcal{T}'$

If a leaf node is present at an odd distance from the root, the corresponding codeword will end in the same base-pair as that of the leaves of the root - A or T if the $current\_tree = \mathcal{T}$, or G or C if the $current\_tree = \mathcal{T}'$. The tree and its complement start with different base-pairs. To decode the next symbol, a switch is done from the $current\_tree$ to its complement tree, thus ensuring that the next occurring base in the DNA sequence is always different from the previous one. □

Theorem 2: Let the number of nucleotides generated in the DNA sequence with G and C bases be n_{GC} and those with A and T bases be n_{AT}. They are related as

$$n_{AT} - 1 \leq n_{GC} \leq n_{AT}$$

thus holding the property of GC-constraint.

Proof: A leaf of the tree may be at either an even distance or an odd distance from the root.

If the leaf is at an even distance from the root, the length of the codeword can be written as $2k_e$ for some integer k_e. Out of these $2k_e$ nucleotides, k_e nucleotides will contain A or T and remaining k_e will contain G or C as the edges in the tree are alternatively labeled. So all the codewords having even length have

$$n_{AT} = n_{GC} \tag{1}$$

If the leaf is at an odd distance from the root, the length of the codeword can be written as $2k_o + 1$ for some integer k_o. If the root of the tree is $\mathcal{T}$, there will be $k_o + 1$ number of nucleotides containing A ot T and k number of nucleotides containing C or G base-pairs, since $\mathcal{T}$ starts with A and T base-pairs as edges from the root and ends in the same. Hence, $n_{AT} = n_{GC} + 1$. If the root of the tree is $\mathcal{T}'$, there will be k_o number of nucleotides containing A ot T and $k_o + 1$ number of nucleotides containing C or G base-pairs, since $\mathcal{T}'$ starts with G and C base-pairs as edges from the root and ends in the same. Hence, $n_{GC} = n_{AT} + 1$.

On encountering an odd distance from the root to the leaf, a switch from the tree to its complement tree is done, so there will be either equal number of switches from $\mathcal{T}$ to $\mathcal{T}'$ and vice-versa, or an extra switch from $\mathcal{T}$ to $\mathcal{T}'$.
If there are equal number of switches, say k,
$n_{AT} = k(n_{GC} + 1)$ for switching from $\mathcal{T}$ to $\mathcal{T}'$, and
$n_{GC} = k(n_{AT} + 1)$ for switching from $\mathcal{T}'$ to $\mathcal{T}$, and from these two equations, it can be inferred that

$$n_{AT} = n_{GC} \tag{2}$$

If there is one extra switch from $\mathcal{T}$ to $\mathcal{T}'$,
$n_{AT} = k(n_{GC} + 1) + 1$ for switching from $\mathcal{T}$ to $\mathcal{T}'$, and
$n_{GC} = k(n_{AT} + 1)$ for switching from $\mathcal{T}'$ to $\mathcal{T}$, and from these two equations, it can be inferred that

$$n_{AT} = n_{GC} + 1 \tag{3}$$

Therefore, from the equations 1, 2 and 3, we have

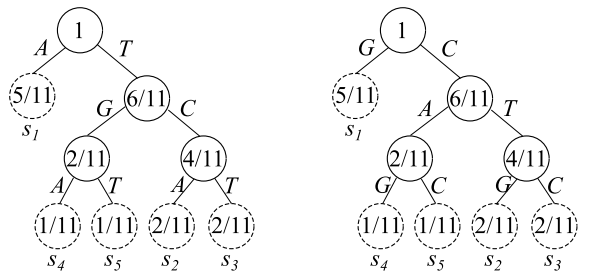
$$n_{AT} - 1 \leq n_{GC} \leq n_{AT}$$

□

C. Illustration

Consider the text **abracadabra** which has to be encoded into DNA sequence. The proposed algorithm in Section III works as follows

1) Converting the raw data into binary format.
   - Using 8-bit ASCII representation, the text can be represented into binary as
     01100001011000100111001001100001011000 11
     01100001011001000110000101100001001110 0
     1001100001



(a) Minimum Variance Huffman Tree $\mathcal{T}$ (b) Complement Huffman Tree $\mathcal{T}'$

Fig. 2. Minimum Variance Huffman Tree and its Complement.

2) Creating the DNA tree
   - Using block length $blen = 8$, the binary data is divided into non-overlapping blocks as
     01100001 01100010 01110010 01100001 01100011 01100001 01100100 01100001 01100010 01110010 01100001
   - Calculating the source probability

Symbol	s_1 = 01100001	s_2 = 01100010	s_3 = 01110010	s_4 = 01100011	s_5 = 01100100
Probability	5/11	2/11	2/11	1/11	1/11

   - Constructing the minimum variance Huffman tree and its complement tree as shown in Fig. 2.
   - Create the codebook $\mathcal{C}$ and its complementary codebook $\mathcal{C}'$.

Symbol	Probability	Codeword $\mathcal{C}$	Codeword $\mathcal{C}'$
s_1 = 01100001	5/11	A	G
s_2 = 01100010	2/11	TCA	CTG
s_3 = 01110010	2/11	TCT	CTC
s_4 = 01100011	1/11	TGA	CAG
s_5 = 01100100	1/11	TGT	CAC

3) Encoding
   - On encoding the binary data using the algorithm as described in Section III-A, the DNA sequence obtained is

$$ACTGTCTGTGAGTGTGTCACTCA$$

IV. Experimental Results and Analysis

The poem *Where the Mind is Without Fear* by *Rabindranath Tagore* [19] containing 490 ASCII characters and two images of size 256×256 and 512×512, each of 8-bit depth have been encoded to DNA sequence by the proposed algorithm. On binary encoding, the text comprises of 3920 bits, and the images of 524288 bits and 2097152 bits respectively. The resultant code-rates are shown in TABLE II. For lossy storage of images, two least-significant bit-planes have been dropped out.

Although, the proposed algorithm will preserve the constraints and provide same code-rate for any Huffman tree, the use of minimum-variance tree is more effective. Minimum variance Huffman tree is the least skewed of all the possible Huffman tree that can be constructed from a source. This ensures construction of a balanced tree with a low depth, therefore will be more effective for construction, especially when two trees are required in the proposed algorithm. Moreover during synthesis of DNA, long sequences are broken into chunks of oligonucleotides [2]. Codewords obtained from

TABLE I
COMPARISON OF CODE-RATES (IN BITS/NT)

Method	Ref. [3]	Ref. [7]	Ref. [8]	Ref. [17]	Ref. [10]	Ref. [9]	Ref. [11]	Ref. [12]	Ref. [18]	**Proposed algorithm with block-length = 16** Text	Image 256×256 Lossless	Image 256×256 Lossy	Image 512×512 Lossless	Image 512×512 Lossy
Code-rate	1.00	1.58	1.78	1.58	1.98	1.60	1.90	1.92	2.14	2.41	2.09	2.34	2.31	2.69

TABLE II
CODE-RATE (IN BITS/NT)


Where the mind is without fear and the head is held high
Where knowledge is free
Where the world has not been broken up into fragments
By narrow domestic walls
Where words arose out from the depth of truth
Where tireless striving stretches its arms towards perfection
Where the clear stream of reason has not lost its way
Into the dreary desert sand of dead habit
Where the mind is led forward by thee
Into ever-widening thought and action
Into that heaven of freedom, my Father, let my country awake.

Block Length	Text (3920 bits)	Jetplane 256×256 (524288 bits) Lossless	Jetplane 256×256 (524288 bits) Lossy	Lena 512×512 (2097152 bits) Lossless	Lena 512×512 (2097152 bits) Lossy
2	1	1.197639424	1.263519233	1.171221531	1.228295374
3	1	1.319701222	1.42626321	1.341941649	1.455920928
4	1.200353045	1.412354964	1.549806619	1.478360275	1.6378137
5	1.03500761	1.469573272	1.62604682	1.575209711	1.766947553
6	1.127694859	1.516436595	1.688307048	1.663466591	1.884617035
7	1.095008052	1.552518609	1.736227368	1.73751573	1.983353034
8	1.863013699	1.594781342	1.792014274	1.811045915	2.081394553
9	1.256157635	1.618081775	1.822485003	1.853951634	2.138576109
10	1.430073607	1.65386996	1.868023332	1.905226361	2.20672607
11	1.428571429	1.692591432	1.915314108	1.956351118	2.274125968
12	1.81011535	1.742845455	1.972894182	2.00969063	2.343259028
13	1.659211061	1.808863847	2.044277623	2.060840252	2.406915701
14	1.857923497	1.887135225	2.125664766	2.126902269	2.485151213
15	1.928166352	1.972202029	2.211884597	2.195659882	2.560755545
16	2.412773507	2.088921951	2.337733991	2.306773416	2.687611454

Minimum-variance Huffman tree will ensure near-equal storage of information in every chunk, and will be more effective than any other Huffman tree.

It is observed that the code-rate generally increases with increase in block-length, however it may not always be true. Theoretically, on increasing the block-length to n, the number of symbols increase exponentially in $O(2^n)$ in worst-case, however, practically it is much less. This increases the tree-construction complexity when probability computation and sorting needs to be performed, and produces larger trees and codebooks. The encoding complexity also increases since it depends on search time from the codebook. However, since decoding complexity requires reading of the nucleotides, with increase in block-length if the code-rate is higher, the decoding complexity is lesser.

The proposed algorithm is based on variable-length encoding contrary to most of the existing methods which are based on fixed-length encoding. Hence, code-rate depends on the characteristics of the input data and varies from one type of data to another. Moreover, lossy storage of images have a higher code-rates than lossless ones. The comparative results are shown in TABLE I, showing a high code-rate, mostly better than the existing ones.

V. CONCLUSION AND FUTURE WORK

In this proposed algorithm, minimum-variance Huffman tree has been used for generating DNA sequences to store text and image data. The algorithm gives a high code-rate of the DNA sequences, and therefore reduces synthesizing cost. It interleaves compression and variable-length constraint coding. This technique can be further extended to video and other multimedia data. Moreover, error detection and correction can also be implemented in the DNA sequences to make them more robust.

REFERENCES

[1] A. Patrizio. (Dec. 2018). *IDC: Expect 175 Zettabytes of Data Worldwide by 2025*. [Online]. Available: https://www.networkworld.com/article/3325397/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html

[2] S. Mohammadhossein and T. Yazdi, "DNA–based data storage system," Ph.D. dissertation, Univ. Illinois Urbana-Champaign, Champaign, IL, USA, 2017.

[3] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, Sep. 2012.

[4] P. Yakovchuk, "Base-stacking and base-pairing contributions into thermal stability of the DNA double helix," *Nucleic Acids Res.*, vol. 34, no. 2, pp. 564–574, Jan. 2006.

[5] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free DNA-based data storage," *Sci. Rep.*, vol. 7, no. 1, p. 5011, 2017.

[6] L. Organick *et al.*, "Random access in large-scale DNA data storage," *Nature Biotechnol.*, vol. 36, no. 3, p. 242, 2018.

[7] N. Goldman *et al.*, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, Feb. 2013.

[8] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust chemical preservation of digital information on DNA in silica with error-correcting codes," *Angew. Chem. Int. Ed.*, vol. 54, no. 8, pp. 2552–2555, Feb. 2015.

[9] M. Blawat *et al.*, "Forward error correction for DNA data storage," *Procedia Comput. Sci.*, vol. 80, pp. 1011–1022, 2016.

[10] Y. Erlich and D. Zielinski, "Capacity-approaching DNA storage," *bioRxiv*, p. 074237, Sep. 2016.

[11] W. Song, K. Cai, M. Zhang, and C. Yuen, "Codes with run-length and GC-content constraints for DNA-based data storage," *IEEE Commun. Lett.*, vol. 22, no. 10, pp. 2004–2007, Oct. 2018.

[12] Y. Wang, M. Noor-A-Rahim, E. Gunawan, Y. L. Guan, and C. L. Poh, "Construction of bio-constrained code for DNA data storage," *IEEE Commun. Lett.*, vol. 23, no. 6, pp. 963–966, Jun. 2019.

[13] K. A. Schouhamer Immink and K. Cai, "Efficient balanced and maximum homopolymer-run restricted block codes for DNA-based data storage," *IEEE Commun. Lett.*, vol. 23, no. 10, pp. 1676–1679, Oct. 2019.

[14] D. Limbachiya, M. K. Gupta, and V. Aggarwal, "Family of constrained codes for archival DNA data storage," *IEEE Commun. Lett.*, vol. 22, no. 10, pp. 1972–1975, Oct. 2018.

[15] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[16] K. Sayood, *Introduction to Data Compression*. San Mateo, CA, USA: Morgan Kaufmann, 2017.

[17] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and A. Karin, "Strauss DNA-based archival storage system," in *Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2016, pp. 637–649s.

[18] M. Dimopoulou, M. Antonini, P. Barbry, and A. R. Appuswamy, "A biologically constrained encoding solution for long-term storage of images onto synthetic DNA," in *Proc. 27th Eur. Signal Process. Conf. (EUSIPCO)*, 2019, pp. 1–5.

[19] R. T. Gitanjali, *Tredition Classics*. Hamburg, Germany: Tredition Classics, 2012.