A Study on High Information Density Encoding Scheme in DNA-Based Data Storage.

Abdulwahab Alobaid, Noura Aljeri

***Abstract*—This work performs a study on a new DNA encoding scheme that aims for high information density in the context of DNA-Based data storage. With the increasing data generation and the need for efficient storage solutions to cope with this demand, DNA storage showed promising capabilities due to its high-density storage capacity. This work explores and applies a new DNA encoding scheme, and determines its efficiency in reducing the length of the encoded DNA possibly holding data and optimizing its storage in DNA molecules. This work also studies the implications of high information density on biological errors, and storage. Preliminary results show that such high information density is still far from real world applications, as the codebook resulting from the encoding process is significantly large, additionally, the error penalty grows with the information density, which makes adopting error correction methods to the encoding scheme a necessary step.**

Keywords: DNA, Information Density, Encoding, Data, Storage, Error Correction, Bioinformatics, Algorithms.

I. Introduction & Motivation

According to a study done by [1], the data generated worldwide in 2025 will reach 175 zettabytes, meaning standard storage mediums will have problems with this increase in data consumption and usage, meaning more efficient data storage solutions have to be explored and developed to cope with this everincreasing demand. One of the promising alternatives being explored for handling massive amounts of data is DNA storage. DNA, the molecule that encodes genetic information in all living things, has an incredibly dense storage capacity. In fact, one gram of DNA is capable of storing 215 petabytes, i.e., 215 million gigabytes [2] Making it an attractive candidate for long-term data storage. However, the process of synthesizing and sequencing DNA is currently quite costly and complex, where it costs $7000 to synthesize 2 megabytes of data and another $2000 to read it [3]. So to reduce this overhead, many competitive encoding and compression techniques have been developed, aiming to maximize the efficiency of data storage in DNA molecules. This work, explores an encoding scheme that aims to maximize information density in the context of DNA-based data storage. While the implementation is still in progress and certain components are under development, this preliminary study investigates the potential of the proposed method by evaluating key metrics such as information density and codebook overhead. The work also identifies critical challenges and outlines directions for future enhancement, including error correction and practical feasibility assessments.

II. Related Work

Although this field is considered relatively new, the DNA storage system is very promising. However DNA sequences come very long in general, so searching through the whole DNA is considered tedious especially for large DNA sequences, the work introduced by [4] designs a compression algorithm that can recover each file individually and with no errors, using a random access approach, meaning direct access to specific files within the DNA sequence. This is very effective because this way; operating systems, server files, large archives, etc.. can be stored efficiently. Another work also utilises Huffman algorithm to build DNA encryption algorithm [5]. This work basically takes a DNA sequence, encodes it using Huffman algorithm, and then applies multiple encryption by using XOR operations followed by a diffusion with permutation boxes. For works related to performance, a new lossless DNA compression algorithm based on A single-block encoding scheme was introduced by [6]. This work is divided into phases, the most notable thing is that the algorithm notes the nucleotide distribution i.e. the frequency and position of nucleotides, then it generates a reference file that is later used to encode the DNA sequence, the method smartly uses the position of neighbouring bits to generate the next bits.

III. Methodology

The encoding process generates a DNA sequence that represents an arbitrary file, along with a codebook used by the decoding process to accurately reconstruct the original file from the sequence, making this method a lossless encoding method.

A. Encoding

The algorithm begins by converting the input file into binary format. The binary stream is then divided into segments of three bytes each, which are grouped into 3-tuples. For each tuple, a simple maximum function is applied: if the first value is the largest, the nucleotide ‘A’ is written to the output; if the second is the largest, ‘T’ is written; if the third is the largest, ‘G’ is written. If the first and second values are both greater than the third, ‘C’ is written. This process continues until the entire binary file is encoded into a DNA sequence.

Each time a nucleotide is written to the output sequence, a corresponding entry is added to the codebook. For example, the entry A(99, 03, 55) represents the original 3-byte binary segment, with each byte converted to its decimal equivalent. This mapping allows the decoding process to reverse the transformation and recover the original binary data.

In many cases, the binary representation of the file is not evenly divisible by three, resulting in an incomplete final tuple. To address this, padding is added by filling the remaining

positions in the tuple with predefined dummy values (by adding 1s in binary). These padding values are also recorded in the codebook to ensure that the decoder can accurately reconstruct the original binary data.

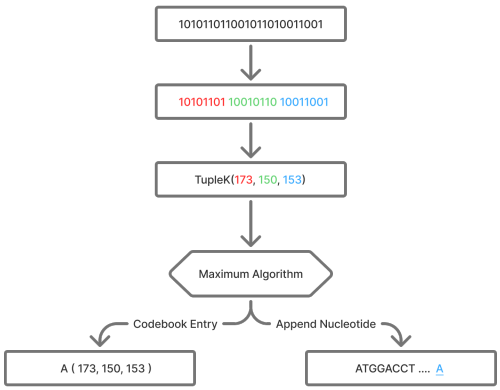

Figure 1. Encoding Algorithm Flowchart

B. Decoding

In addition to the DNA sequence, the decoding process only requires the codebook produced from the encoding scheme to be able to generate the original file. Which is done by going through base by base in both the code book and the DNA sequence. The decoder converts the tuple into binary segments, and removes any extra padding. After all the file is regenerated to binary, the final step is to convert the file back into it's original format.

C. Example: Encoding a Text File

The following example demonstrates the encoding process using a text file that contains this string:

```
hey, look above!
```

The binary representation of this string is:

```
01101000 01100101 01111001 00101100
00100000 01101100 01101111 01101111
01101011 00100000 01100001 01100010
01101111 01110110 01100101 00100001
```

This binary stream is divided into 3-byte segments, forming the following 3-tuples with the last tuple containing 2-byte padding:

```
tuple1: 01101000 01100101 01111001
tuple2: 00101100 00100000 01101100
tuple3: 01101111 01101111 01101011
tuple4: 00100000 01100001 01100010
tuple5: 01101111 01110110 01100101
tuple6: 00100001 11111111 11111111
```

Each tuple is then converted to its decimal representation:

```
tuple1: (104, 101, 121)
tuple2: (44, 32, 108)
tuple3: (111, 111, 107)
tuple4: (32, 97, 98)
tuple5: (111, 118, 101)
tuple6: (33, 255, 255)          (padding)
```

Applying the max function to each tuple, the resulting DNA sequence is:

```
CCATTT
```

And the codebook is constructed as follows:

```
C (104, 101, 121)
C (44, 32, 108)
A (111, 111, 107)
T (32, 97, 98)
T (111, 118, 101)
T (33, 255, 255)
```

IV. Results and Discussion

In terms of information density, this method surpasses all existing coding schemes known to date, to the best of our knowledge. For reference, Table I presents a representative subset of prior works compiled from Organick et al. [4] for comparison.

Work	Information Density (bits/base)
This Work	**24.00**
Organick et al.	1.10
Bornholt et al.	0.85
Grass et al.	1.26

Table I
Comparison of information density achieved by various DNA storage methods

Despite achieving high information density, real-world implementation presents significant practical limitations. These constraints are discussed in the following section.

A. Lab Constraints

This work focuses primarily on increasing information density, without addressing certain biological considerations, including:

- GC content balance.
- Homopolymer avoidance, since long runs of identical nucleotides negatively impact sequencing accuracy.
- Avoidance of secondary structures and primer-dimer formations, which can interfere with amplification and sequencing.
- Some implementations, such as those by [4], support random access by assigning a unique primer to each DNA strand (with each strand representing a file), thus incorporating primer design into their methodology.

Although the current implementation does not adhere to these constraints, the max algorithm serves as a placeholder that can be replaced by any other effecient encoding method if it produces a similar codebook.

B. Storage

The problem in the storage lies in the codebook size which is considered very large. The proposed way of storing the each codebook is by using the following class:

```
class CodeBookEntry {
    char base;  // 1-byte

    mi_int1 r;  // 1-byte integer
    mi_int1 g;
    mi_int1 b;
};
```

Given this class, it means that each base will use one CodeBookEntry object that requires 4-bytes. And to calculate the codebook size, the length of the resulting DNA sequence is multiplied by 4 (since each nucleotide has a corresponding codebook entry). Given the encoding scheme, the sequence length is ciel(bytes/3) where 3 is the tuple size. So for a Terabyte, the codebook size will be ciel(bytes/3)*4 which is 1.3333 Terabytes, and that's larger than the original data.

A possible approach to reduce the size of the codebook is to explore interpolation methods over multiple variables. Consider the encoding function as $F(x, y, z, w)$, where x, y, and z represent the three bytes, respectively, and w denotes the position of the nucleotide in the sequence (e.g. 0, 1, 2, ...). If such interpolation techniques can be applied, it may be sufficient to store only a subset of points in the codebook, using interpolation to estimate the remaining entries and reconstruct the original codebook.

However, further investigation is needed to address the following questions:

- Is this approach feasible in practice?
- To what extent will it reduce the size of the codebook?
- What is the accuracy of the interpolation, and what is the minimum number of points required to achieve reliable reconstruction?
- What are the computational requirements? Can the interpolation be efficiently performed on standard hardware, or is parallel computation (e.g., GPU or distributed systems) necessary?

C. Error Tolerance

Additionally, one has to identify the amount logical redundancy required by the error correction algorithm, this can also be influenced by different sequencing and synthesis technologies, as they have different indel(insertion, deletion) and subtitution error distributions.

Although achieving high information density is beneficial, it comes at the cost of losing 3 bytes per base, equivalent to 24 bits per base. Compared to other methods, the impact of each error increases significantly, since losing a single base results in losing a substantial amount of information, as illustrated in Figure 2.

Therefore, efficient error correction is essential, particularly as coding rates increase. One alternative is to rely on sequencing coverage; however, this approach generally raises overall costs. It is also important to note that implementing error correction reduces the effective code rate by introducing redundancy into the data. A notable example of an efficient Reed-Solomon error correction implementation for DNA storage is presented in [7].

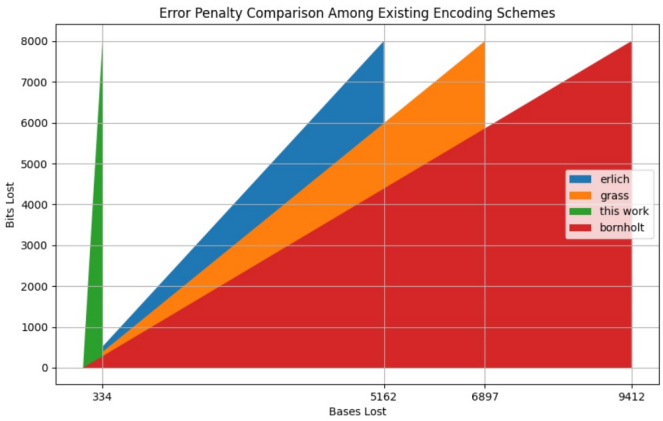

Figure 2. Error Penalty Compared to Other Studies

Additionally, determining the appropriate amount of logical redundancy required by the error correction algorithm is also a considerable factor. This requirement varies depending on the sequencing and synthesis technologies used, as these have different error profiles, including insertion, deletion, and substitutions errors.

V. Future Directions

- Apply error correction using the implementation by [7]: https://github.com/whpress/hedges.
- Identify error distributions.
- Explore possibilities for random access; Organick et al. [4] work is relevant in this context.
- Investigate the implications of increasing information density, e.g., using 4-tuples instead of 3-tuples, or any x-bits per nucleotide. The codebook and error correction parameters are expected to be directly affected.
- Conduct real tests of the encoding scheme in the laboratory once constraints are met (a later step).
- The DNA as an image idea in the conclusion was not a unique idea to this work. This study [8] applies insights from the image-based DNA project: https://github.com/MahdiKarimian/DIF. It might be possible to extend this work for the current implementation.
- Analyze encoding time and memory usage.

References

[1] T. Sullivan, "Ai and the global 'datasphere': How much information will humanity have by 2025?" www.datauniverseevent.com, 06 2023. [Online]. Available: https://www.datauniverseevent.com/en-us/blog/general/AI-and-the-Global-Datasphere-How-Much-Information-Will-Humanity-Have-By-2025.html

[2] M. Tech., "Dna's awesome potential to store the world's data," www.micron.com, 03 2020. [Online]. Available: https://www.micron.com/about/blog/applications/data-center/dnas-awesome-potential-to-store-the-worlds-data

[3] R. F. Service, “Dna could store all of the world’s data in one room,” www.science.org, 03 2017. [Online]. Available: https://www.science.org/content/article/dna-could-store-all-worlds-data-one-room

[4] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. N. Takahashi, S. Newman, H.-Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss, “Random access in large-scale dna data storage,” *Nature Biotechnology*, vol. 36, pp. 242–248, 02 2018. [Online]. Available: https://www.nature.com/articles/nbt.4079

[5] M. Meftah, A. A. Pacha, and N. Hadj-Said, “Dna encryption algorithm based on huffman coding,” *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 25, pp. 1831–1844, 12 2020.

[6] D. Mansouri, X. Yuan, and A. Saidani, “A new lossless dna compression algorithm based on a single-block encoding scheme,” *Algorithms*, vol. 13, p. 99, 04 2020.

[7] W. H. Press, “Hedges: Dna-based error correction code,” https://github.com/whpress/hedges, accessed: 2025-05-29.

[8] M. Karimian, “Dif: Dna image file project,” https://github.com/MahdiKarimian/DIF, accessed: 2025-05-29.