PolyHalt DNA Encoding

Group Number 3
Dhirubhai Ambani Institute of Information and Communication Technology
Guide : Prof. Manish K Gupta

***Abstract*—DNA data storage has gained significant attention as a potential solution for information storage at unprecedented densities. However, challenges such as homopolymers and secondary structures in DNA sequences pose obstacles to efficient and reliable encoding. In this paper, we introduce a novel DNA encoding scheme that effectively prevents the formation of homopolymers, minimizes the occurrence of secondary structures and takes care about GC content as well as the edit distance among codewords.**

Through a comprehensive analysis, we demonstrate the enhanced reliability and efficiency of our encoding scheme. The prevention of homopolymers ensures better error correction capabilities, while the mitigation of secondary structures contributes to improved sequencing accuracy.

***Keywords*—DNA data storage, Encoding scheme, Homopolymer prevention, secondary structure minimization, error correction, GC content, Stochastic Search,Information Density.**

I. Introduction

In the fast-changing world of science, blending computer know-how with life sciences has opened up a whole new way to store digital info – in DNA. This conference paper digs into the latest ways we're encoding data in DNA, giving a big picture of where we stand and what exciting things might come next. Our digital data is growing crazy fast, and the usual ways we store it have limits. That's where DNA steps in, offering a way to store lots of info in a tiny, long-lasting package. This conference paper dives into the methods we're using to turn digital data into a DNA-friendly form. It's not just about storing data; it's about secure communication, molecular computing, and even creating custom biological stuff. Here, we'll explore the many ways scientists are encoding info in DNA, from using the four letters of the genetic alphabet to smart techniques that fix errors and add extra info for safety. And we won't just talk theory – we'll also look at how these methods are being used in the real world. Think data backup, secure communication, and keeping info safe in the digital world. This conference paper isn't just for the experts. We'll keep it simple, exploring how DNA encoding works, why it matters, and the cool things it can do. From packing in lots of info in a tiny space to making sure the info is read correctly, we'll break down the challenges and the practical uses of DNA encoding. So, whether you're a science whiz or just curious, join us as we uncover the secrets of encoding information in the language of life. Get ready for a journey into the exciting world where bits and bytes meet DNA code-words, unlocking new possibilities for how we store and use information.

II. Data Transformation

A. What is DNA encoding?

DNA encoding is the process of translating digital information into the language of DNA, utilizing the four-letter genetic alphabet (Adenine, Thymine, Cytosine and Guanine) to represent binary data. This process holds promise for compact and long-term data storage.

To illustrate, consider a simple encoding scheme where we represent the binary sequence 0010 using the DNA alphabet:

Binary	DNA Representation
00	A (Adenine)
01	C (Cytosine)
10	G (Guanine)
11	T (Thymine)

In this example, each pair of binary digits is mapped to a corresponding DNA base. The sequence "0010" is encoded as "AG" in DNA. This simplistic representation illustrates the fundamental concept of DNA encoding, where digital information is stored in the molecular structure of DNA.

The advantages of DNA encoding include high information density and long-term stability, making it a promising solution for archival storage of data.

B. What is DNA decoding?

DNA decoding is the process of interpreting the information stored in DNA back into its original digital form. This reverse process involves translating the sequence of DNA bases into binary code or another digital representation.

Consider the following DNA sequence:

```
ATCG
```

To decode this DNA sequence, we establish a correspondence between each DNA base and its digital counterpart:

DNA Base	Binary Representation
A (adenine)	00
C (cytosine)	01
G (guanine)	10
T (Thymine)	11

Applying this mapping to our DNA sequence, we decode it as:

```
00 11 01 10
```

In this example, decoding involves translating each DNA base back into its binary representation. This process allows us to retrieve the original digital information that was encoded in the DNA sequence.

The significance of DNA decoding lies in its ability to retrieve digital data stored in DNA, making it a crucial step in utilizing DNA as a storage medium.

III. Understanding Homopolymers

A. What are Homopolymers?

Homopolymers in DNA refer to consecutive repetitions of the same nucleotide, such as A, T, C, or G, within a DNA sequence. For instance, a sequence like `AAGAAGAAG` contains a homopolymer of adenine (A) due to its consecutive repetition of the same nucleotide.

B. What is the degree of Homopolymers?

The degree of homopolymers is the number of nucleotide that repeats in the DNA sequence.

1) **1 degree homopolymers:** If the number of repetitive nucleotide is 1 then it is called one degree homopolymers. Ex. AA
2) **2 degree homopolymers:** If the number of repetitive nucleotide is 2 then it is called two degree homopolymers. Ex. AGAG
   The examples of 3 and 4 degree homopolymers are AGCAGC and AGCTAGCT respectively.

C. Why should we Prevent Homopolymers?

Preventing homopolymers is crucial for several reasons:

1) **Sequencing Accuracy:** Homopolymers can complicate DNA sequencing by making it challenging to accurately determine the length of the repeated region. This may lead to sequencing errors.
2) **Synthesis Errors:** During DNA synthesis processes, such as PCR(Polymerase Chain Reaction) or DNA synthesis in the laboratory, homopolymers may result in errors like nucleotide insertions or deletions.
3) **Data Analysis Challenges:** Analyzing data with homopolymers can be computationally demanding. Accurately interpreting sequencing data with homopolymeric regions often requires sophisticated algorithms and error correction techniques.
4) **Biological Implications:** In certain biological contexts, the presence of homopolymers may have functional implications, such as affecting the secondary structure of DNA or influencing interactions with proteins.

IV. Conflict Free DNA Codes

Imagine you have a DNA string, which is essentially a sequence of letters representing the genetic code (A for adenine, T for thymine, G for guanine, and C for cytosine).

Now, this DNA string is $\mathcal{L}$ conflict-free if it doesn't have repeated consecutive sequences of letters for any sub-string length from 1 to $\mathcal{L}$.

For Example, Let's say you have a DNA string : $ATCGATCGA$.

- For $t = 1$ (sub-strings of length 1): A, T, C, G are all different, so it's conflict-free for $t = 1$.
- For $t = 2$ (sub-strings of length 2): AT, CG, AT, CG are not consequtive sub-sequences, so it's conflict-free for $t = 2$.
- For $t = 3$ (sub-strings of length 3): ATC, GAT, CGA are all different, so it's conflict-free for $t = 3$.
- For $t = 4$ (sub-strings of length 4): $ATCG, ATCG$ have repeated consecutive sub-sequences, so it's not conflict-free for $t = 4$.

The DNA code is called complete conflict free if it is conflict free for $t = n/2$.

V. Understanding GC Content

GC content refers to the percentage of nucleotides in a DNA or RNA sequence that are either guanine (G) or cytosine (C). It is a crucial aspect of DNA analysis with implications for various biological processes. Understanding GC content provides insights into the stability, structure, and function of DNA.

A. Calculation of GC Content

GC content is calculated using the formula:

$$\text{GC content } (\%) = \left(\frac{\text{Number of G and C nucleotide}}{\text{Total number of nucleotide}} \right) \times 100$$

For example, in the sequence `ATCGATCG`, there are four G and C nucleotides out of a total of eight nucleotides. Therefore, the GC content is:

$$\text{GC content } (\%) = \left(\frac{4}{8} \right) \times 100 = 50\%$$

B. Biological Significance

The GC content of a DNA sequence is biologically significant for several reasons:

1) **Thermal Stability:** DNA sequences with higher GC content generally have higher thermal stability due to the stronger bonding between G and C nucleotides (three hydrogen bonds) compared to A and T nucleotides (two hydrogen bonds).
2) **Structural Implications:** GC-rich regions are associated with specific DNA structures, such as hairpin loops and stem-loop structures, influencing the overall structure and function of the DNA.
3) **Primer Design:** In molecular biology applications, knowledge of GC content is crucial for designing primers with balanced GC content, optimizing PCR reactions, and ensuring successful DNA amplification.

C. GC Content Analysis

GC content analysis is commonly performed using bioinformatics tools. These tools help researchers examine the GC content of entire genomes, specific genes, or individual DNA sequences. Understanding GC content variations can provide valuable information for functional genomics and molecular biology research.

VI. Understanding Secondary Structure Formation with Reverse Complement

The formation of secondary structures in nucleic acids plays a crucial role in various biological processes. One concept integral to understanding these structures is the idea of the "reverse complement."

A. Reverse Complement

The reverse complement of a DNA or RNA sequence is a fundamental concept in molecular biology. It involves two operations:

1) **Reverse:** Reversing the order of nucleotides in the sequence.
2) **Complement:** Replacing each nucleotide with its complementary base: adenine (A) with thymine (T) in DNA or uracil (U) in RNA, and vice versa; guanine (G) with cytosine (C) and vice versa.

For example, consider the DNA sequence `ACCTGAG`. The reverse complement is `ACCTGAG` → `GAGTCCA` → `CTCAGGT`.

B. Secondary Structure Formation

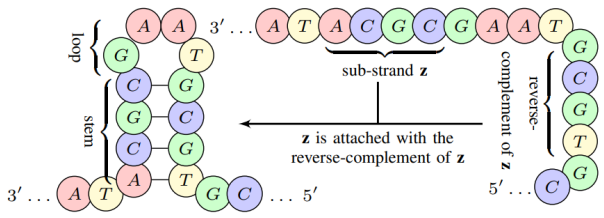

Fig. 1: Schematic representation of a hairpin-like secondary structure in DNA.

Secondary structures in nucleic acids often involve the formation of base pairs through hydrogen bonding between complementary bases. The concept of reverse complement is particularly relevant in this context.

1) Hairpin Loops: Hairpin loops are common secondary structures formed by a single-stranded nucleic acid folding back on itself. The formation of a hairpin loop often involves the pairing of complementary bases, facilitated by the reverse complementarity of the sequence.

2) Three way junctions: Three-way junctions, also known as Y-junctions, occur when three strands of nucleic acid come together. This structure is common in processes like DNA replication and recombination.

VII. Understanding Hamming Distance

Hamming distance is a metric used to measure the difference between two strings of equal length. It quantifies the minimum number of substitutions required to change one string into the other. Hamming distance is particularly useful in error detection and correction, cryptography, and DNA sequence analysis.

A. Calculation of Hamming Distance

The Hamming distance between two strings of equal length n is calculated by comparing corresponding symbols and counting the positions where the symbols differ. Mathematically, the Hamming distance d_H is given by:

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

where $\delta(x_i, y_i)$ is the Kronecker delta function, equal to 0 if $x_i = y_i$ and 1 otherwise.

For example, consider the strings $x =$ "101010" and $y =$ "111000". The Hamming distance between x and y is 3, as there are three positions where the symbols differ.

B. Applications of Hamming Distance

Hamming distance finds applications in various fields:

1) **Error Detection and Correction:** In coding theory, Hamming distance is used to detect and correct errors in transmitted data. Codes with higher minimum Hamming distance can correct more errors.
2) **Cryptography:** Hamming distance is employed in cryptographic algorithms for measuring the similarity between encrypted and decrypted messages, helping to assess the security of cryptographic systems.
3) **DNA Sequence Analysis:** In bioinformatics, Hamming distance is used to compare DNA sequences, identify mutations, and assess genetic diversity.
4) **Data Clustering:** Hamming distance is utilized in clustering algorithms to group similar data points based on their binary representations.

C. Limitations and Considerations

While Hamming distance is a valuable metric, it has limitations. It assumes equal importance for all positions in the string and is sensitive to changes in every position. For certain applications, considering different weights for different positions may be more appropriate.

VIII. Understanding Edit Distance

Edit distance, also known as Levenshtein distance, is a metric used to quantify the similarity between two strings by measuring the minimum number of single-character edits (insertions, deletions, or substitutions) needed to transform one string into the other. Edit distance is widely applied in spell checking, DNA sequence alignment, and natural language processing.

A. Calculation of Edit Distance

The edit distance between two strings x and y, each of length m and n respectively, is calculated using dynamic programming. The edit distance matrix D is populated iteratively, and the final edit distance is obtained from the bottom-right corner of the matrix.

The recurrence relation for calculating $D[i, j]$, the edit distance between the first i characters of x and the first j characters of y, is given by:

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 & \text{(deletion)} \\ D[i,j-1]+1 & \text{(insertion)} \\ D[i-1,j-1]+\delta(x_i,y_j) & \text{(substitution)} \end{cases}$$

where $\delta(x_i, y_j)$ is 0 if $x_i = y_j$ and 1 otherwise.

For example, consider the strings $x =$ "kitten" and $y =$ "sitting". The edit distance between x and y is 3, as three operations ("s"-substitution in place of "k", "i"-substitution in place of "e", "g" - insertion at the last place.) are required to transform x into y.

B. Applications of Edit Distance

Edit distance has various applications in different domains:

1) **Spell Checking:** Edit distance is used in spell checking algorithms to suggest corrections for misspelled words by identifying words with minimal edit distances.
2) **Genetic Sequence Alignment:** In Bioinformatics, edit distance is employed for aligning DNA or protein sequences to identify evolutionary relationships and mutations.
3) **Natural Language Processing:** Edit distance is utilized in text processing tasks such as auto-correction, machine translation, and plagiarism detection.
4) **OCR Correction:** Optical character recognition (OCR) systems use edit distance to correct errors in recognized text by identifying the most likely corrections based on minimal edits.

C. Considerations and Variations

While edit distance is a powerful metric, it may not always capture the semantic similarity between strings. Additionally, variations of the edit distance metric exist, such as normalized edit distance, which considers the length of the strings, providing a normalized measure of similarity.

IX. Why Edit Distance is Better in Our Case

In certain scenarios, edit distance proves to be a superior metric for measuring similarity between strings compared to other distance metrics. Let's explore why edit distance is particularly well-suited for our case through examples and considerations.

A. Comparision

Consider a case where we are comparing two text strings: $x =$ "intention" and $y =$ "execution". Our goal is to quantify the similarity between these strings.

1) Scenario 1: Hamming Distance: If we were to use Hamming distance, which only counts the number of differing positions between equal-length strings, it would not be effective in capturing the similarity in this scenario. The strings have different lengths, and Hamming distance is not designed to handle such cases.

2) Scenario 2: Edit Distance: Edit distance, on the other hand, excels in scenarios where the strings have varying lengths and require different operations (insertions, deletions, substitutions) for alignment. Edit distance can be counted by using dynammic programming.

B. Example: 'ACTGACTG' and 'ATGACTGA'

Consider the strings:

$$x = \text{"ACTGACTG"}$$

$$y = \text{"ATGACTGA"}$$

1) Hamming Distance Calculation:: The Hamming distance is applicable when the strings are of equal length. In this case:

"ACTGACTG"

"ATGACTGA"

The Hamming distance is calculated by counting differing positions:

$$\text{Hamming distance} = 7$$

2) Edit Distance Calculation:: Now, let's calculate the edit distance between x and y using dynamic programming:

$$D = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 1 & 0 & 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 2 & 1 & 0 & 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 & 1 & 1 & 1 & 2 & 3 \\ 6 & 5 & 4 & 3 & 2 & 1 & 2 & 2 & 3 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 2 & 2 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 2 & 2 \end{bmatrix}$$

The edit distance between "ACTGACTG" and "ATGACTGA" is 2.

3) Comparison and Conclusion:: The Hamming distance is 7, indicating that the strings differ at 7 positions. However, the edit distance is 2, suggesting a closer similarity, as only 2 operations are needed to transform one string into the other.

In this example, the edit distance better demonstrates the similarity between the strings 'ACTGACTG' and 'ATGACTGA' compared to the Hamming distance.

$$\text{Hamming distance} = 7$$

$$\text{Edit distance} = 2$$

The lower edit distance reflects a higher similarity between the strings.

X. Cyclic Vector Index Encoding

Let's consider a cyclic vector of code-words: A, B, C, D, E, F, and a vector of size $n+1$ to represent the indices in a cyclic manner.

$$\begin{aligned} \text{Code-word Vector:} \quad & [A, B, C, D, E, F] \\ \text{Symbol Vector:} \quad & [0, 1, 2, 3, 4] \end{aligned}$$

Now, if the previous element is B, and you want it to correspond to 0, you can use the following mapping:

$$\begin{aligned} \text{Mapping:} \quad & 0 \to C, \\ & 1 \to D, \\ & 2 \to E, \\ & 3 \to F, \\ & 4 \to A \end{aligned}$$

So, for example, if you have the previous element B. If you have the symbol 2, its corresponding bit-value is D.

A. Table construction:

Let's construct a table with previous elements as row names, indexes as column names, and fill it with code-words.

Previous Code-word	Symbols 0	1	2	3	4
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

This table represents the cyclic mapping of indexes based on the given vector of symbols.

This encoding method ensures a cyclic mapping of indices based on the given vector of symbols.

Conclusion

In this encoding scheme, we have utilized a cyclic vector of size $n+1$ to represent n symbols in a data stream. The table demonstrates the cyclic mapping of indexes based on the chosen sequence. By employing $n+1$ code-words, we ensure a comprehensive encoding scheme that accommodates the cyclic nature of the data stream.

XI. Understanding Information Density

Information density refers to the amount of information that can be conveyed or represented in a given set of data. In the context of DNA encoding methods, information density is a crucial metric that reflects how efficiently genetic information is packed into a sequence of codewords.

A. Calculation of Information Density

The information density (ID) can be calculated using the formula:

$$ID = \frac{\log(k)}{N \log(2)}$$

where k represents the number of unique symbols or codewords used in the encoding process. The logarithmic base 2 is employed to measure information density in bits per symbol.Here N represents the length of the codeword.

B. Interpretation

A higher information density indicates more efficient encoding scheme, as more information is being represented using a relatively smaller set of symbols. This is desirable for various applications, as it allows for more compact storage and transmission of genetic data.

Understanding and optimizing information density are critical in the design of DNA encoding methods, where the goal is often to represent genetic information accurately while minimizing the length of the encoded sequence.

In the subsequent sections, we will explore encoding techniques and strategies that contribute to achieving optimal information density in DNA data representation.

XII. Stochastic Search Algorithm

Stochastic search algorithms are a class of optimization methods that involve randomness in the search process. They are particularly useful for exploring solution spaces where the landscape is complex and there might be multiple local optima.Stochastic search algorithms introduce randomness in the search process. This randomness helps the algorithm to escape local optima and explore a broader region of the solution space.We are using this algorithm for choosing the codewords when number of codewords are out of bound.

XIII. Main Algorithm

Generate all 4^N codewords, filter by removing codewords forming secondary structures and violating homopolymer requirement. In the last step, satisfy the edit distance requirement. Run a binary search over all possible lengths of the set of codewords. Check the possibility of codeword formation. Eventually, move to the final result of the binary search. List out all the codewords of the obtained set. Assuming we use these k codewords to represent k symbols, then the information density is calculated as follows: $\frac{\log(k)}{N \log(2)}$.

Helper Functions:

1) *generateCombination:* It returns all possible combinations of codewords.
2) *getReverseComplement:* It returns the reverse complement of DNA sequence s.
3) *checkForSingleDegree($degree$, s):* Check for repeating patterns of length $degree$ in DNA sequence s.
4) *isHomopolymerFree(s, $limit$):* Check if DNA sequence s is homopolymer-free up to $limit$ degrees.

5) *isSecondaryStructureFree(s, $threshold$):* Check if DNA sequence s is free of secondary structures up to $threshold$.
6) *isBalancedGC(s):* Check if DNA sequence s has balanced GC content.
7) *findEditDistance(a, b):* Calculate the edit distance between DNA sequences a and b.
8) *generateKCodewords(k):* Generate a set of k codewords.

XIV. Detailed algorithms of helper Functions:

- Initialize N, X, Y, E;
  - N = Length of Codeword
  - X = Minimum degree of Conflict-Free Code-Word
  - Y = Minimum degree of Reverse Complement Free Codeword
  - E = Minimum Edit Distance
- Initialize $precautionCount = 0$
- Initialize $precautionLimit = 1 \times 10^5$
- Initialize $totalCount = 0$
- Initialize all as an empty vector of strings
- Initialize $filtered$ as an empty vector of strings
- Initialize $curr$ as an empty string
- Initialize nuc as the string "ACGT"
- Initialize reverseComplement as an empty unordered map from characters to characters
- Initialize prohibited as an empty unordered map from strings to integers

1) *generateCombinations:*
   - If curr.size() $= N$, increment $totalCount$, append curr to all, and return.
   - Set st to rand()$\%4$ - select a random index.
   - Loop over i from 0 to 3:
     - Set n to $nuc[(i + st)\%4]$.
     - Append n to curr.
     - Recursively call generateCombinations().
     - Pop back the last added base from curr (backtrack).
2) *getReverseComplement:*
   - Initialize an empty string res
   - For each character a in s:
     - Append reverse complement of a to res
   - Reverse res
   - Return res
3) *isHomopolymerFree:*
   - Loop over degree from 1 to $limit$:
     - If checkForSingleDegree(degree, s) is false, return false.
   - Return true.
4) *checkForSingleDegree:*
   - Set n to s.size().
   - Loop over i from 0 to $n - 1$:
     - Set startOne to i.
     - Set startTwo to degree $+ i$.
     - Set temp1 to s.substr(startOne, degree) (Extract a substring of length 'degree').
     - Set temp2 to an empty string.
     - If startTwo $< n$, set temp2 to s.substr(startTwo, degree) (Extract another substring starting 'degree' bases ahead).
     - If temp1 = temp2, return false (Repeating pattern found).
   - Return true.
5) *findEditDistance:*
   - Initialize variables: n (size of s), m (size of t), and a matrix *dp* of size $(n+1) \times (m+1)$ with zeros.
   - Iterate over rows i from 0 to n:
     a) Iterate over columns j from 0 to m:
        - If $i = 0$ or $j = 0$, set *dp*$[i][j] = i + j$.
        - Else if $s[i-1] = t[j-1]$, set *dp*$[i][j]$ = *dp*$[i-1][j-1]$.
        - Else, set *dp*$[i][j] = \min(\min(1 +$ *dp*$[i-1][j], 1 +$ *dp*$[i][j-1]), 1 +$ *dp*$[i-1][j-1])$.
6) *isSecondaryStructureFree:*
   - Set rc to s.
   - Reverse rc.
   - Loop over c in rc:
     - Set c to reverseComplement[c].
   - Set $store$ to N.
   - Set N to s.size().
   - Set $maxi$ to -1.
   - Set dp to initialize 2D array for dynamic programming.
   - Loop over i from N to 0:
     - Loop over j from N to 0:
       - If $i = N$ or $j = N$, set dp$[i][j] = 0$ and continue.
       - If s$[i] = rc[j]$:
         - Set dp$[i][j] = 1+$dp$[i+1][j+1]$.
         - Set $maxi$ to max$(maxi,$ dp$[i][j])$.
       - Else, set dp$[i][j] = 0$.
   - Set N to $store$.
   - If $maxi >$ threshold, return false.
   - Return true.
7) *isBalancedGC:*
   - Initialize unordered map mp.
   - Loop over a in s:
     - Increment mp$[a]$ (Count the occurrences of each base).
   - If mp$['A'] +$ mp$['T'] =$ mp$['G'] +$ mp$['C']$, return true.
   - Return false.
8) *generateKCodewords:*
   - Set $valid$ to false.
   - Define getXCodewordsPrint(all, i, $curr$):

- If $valid$ = true, return.
- If $i = 0$:
  - Loop over a in curr:
    - Loop over b in curr:
    - If $a \neq b$ and findEditDistance($a, $b) ¡ H, return (Check Edit distance for each pair in the current set).
  - Set $valid$ to true (Valid set with minimum Edit distance found).
  - Loop over a in curr:
    - Print a.
  - Print "".
  - Return.
- If $valid$ = true, return.
- Loop over a in all:
  - Set $flag$ to 0.
  - Loop over b in curr:
    - If findEditDistance($a, $b) ¡ H:
    - Set $flag$ to 1.
    - Break.
  - If $flag$, continue (Skip sequences not meeting Edit distance requirement).
  - Append a to curr (Add a sequence to the current set).
  - Recursively call getXCodewordsPrint(all, i - 1, curr) (Recursively search for more codewords).
  - Pop back the last added sequence from curr (Backtrack by removing the last added sequence).

XV. Implementation

Let $N = 4$, $X = 3$, $Y = 2$, $E = 2$

Codewords that satisfies above requirements are

CTAC
TAGC
TCAC
TCGT
CGTA
CGAT
TGCT
TACG
TCTG
CTGT
CATC
CTCA
CAGA
CACT
TGTC

Number of obtain codewords - 15. Refer to the mapping table mentioned on the next page, for understanding this example.
The given binary string is 001010100101010_2.
For converting it into DNA sequence of above mentioned, N, X, Y E - we need to firstly convert it into binary to 14 base number beacuse we have total 15 codewords that satisfies the requirements.
Binary to 14 base number System :

$$(001010100101010)_2 = (1D90)_{14}$$

Now, using the mapping table mentioned on the next page , we can convert the 14-base string into a DNA sequence:

$$(1D90)_{14} = \text{TCAC TAGC CTCA CAGA}$$

This DNA sequence satisfies the above mentioned requirements of 3 degree Homopolymer free, secondary struture prevention above 2 degree and minimum edit distance of 2. The Information density is 0.976723.

XVI. Analysis

In this section, we conduct a detailed analysis of the DNA encoding method, focusing on the impact of varying codeword length (N) and edit distance (E), while keeping the codeword as Homopolymer free upto X degree(X) and secondary structure free upto Y degree. The objective is to gain insights into how different combinations of N and E influence the number of codewords and the performance of the encoding method.
These fixed parameters provide a consistent baseline for our analysis, allowing us to isolate the effects of codeword length and edit distance.

A. Variable Parameters

- **Codeword Length (N):** Systematically varying the length of the codewords enables us to observe how the encoding method adapts to different information sequence lengths. Various N values represent distinct scenarios of encoding information into DNA.
- **Edit Distance (E):** The edit distance between codewords is a critical factor affecting error correction. By adjusting E, we examine how the encoding method handles variations and errors in the information sequences.

B. Methodology

For each combination of N and E, we analyze the efficiency, error resilience, and information density of the encoding method. The results offer valuable insights into the method's performance under different scenarios.

C. Objective

Our goal is to identify optimal combinations of N and E that strike a balance between maximizing information density and maintaining robust error-correction capabilities.

Fig. 2: Cyclic Mapping Table

Previous	0	1	2	3	4	5	6	7	8	9	A	B	C	D
CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC
TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC
TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC
TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC
CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT
CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA
TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT
TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT
TCTG	CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG
CTGT	CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG
CATC	CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT
CTCA	CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC
CAGA	CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA
CACT	TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA
TGTC	CTAC	TAGC	TCAC	TCGT	CGTA	CGAT	TGCT	TACG	TCTG	CTGT	CATC	CTCA	CAGA	CACT

Fig. 3: No. of codewords for $X = 4$ conflict-free and $Y = 3$ reverse-complement-free DNA codes with varying N and E

	Edit Distance (E)								
Codeword Length (N)	1	2	3	4	5	6	7	8	9
2	3	1							
4	23	15	3	1					
6	75	66	14	6	2	1			
8	80	74	50	20	7	3	1	1	
10	84	77	70	53	24	9	3	2	1

Fig. 4: Information Density for $X = 4$ conflict-free and $Y = 3$ reverse-complement-free DNA codes with varying N and E

	Edit Distance (E)								
Codeword Length (N)	1	2	3	4	5	6	7	8	9
2	0.79								
4	1.13	0.97	0.39						
6	1.04	1.00	0.63	0.43	0.17				
8	0.79	0.77	0.71	0.54	0.35	0.20			
10	0.64	0.62	0.61	0.57	0.46	0.31	0.16	0.10	

D. Table Presentation

The results of our analysis are presented in a comprehensive table ($Fig. 2, 3, 4, 5$) providing an overview of the encoding method's performance across various N and E scenarios.

This analysis serves as a guide for selecting appropriate parameters when implementing the DNA encoding method, ensuring its effectiveness in diverse scenarios.

XVII. Conclusion

In this study, we draw several conclusions based on the analysis of DNA coding schemes for conflict-free and reverse-complement-free scenarios.

Key Findings and Statements

1) **Robustness vs. Efficiency Trade-off :**
Keeping the edit distance (E) requirement high enhances the robustness of the coding scheme, as it allows for more efficient error correction. However, this comes at a cost, leading to a decrease in information density (ID) and the number of codewords (C). Mathematically, this trade-off can be expressed as:

$$E \uparrow \Rightarrow ID \downarrow, \quad E \uparrow \Rightarrow \ C \downarrow$$

2) **Impact of Codeword Length :**
Increasing the length of the codeword (N) results in a larger symbol space, allowing for more codewords.

Fig. 5: No. of codewords for $X = 2$ conflict-free and $Y = 5$ reverse-complement-free DNA codes with varying N and E

Codeword Length (N)	Edit Distance (E)								
	1	2	3	4	5	6	7	8	9
2	3	1							
4	23	15	3	1					
6	80	77	17	7	2	1			
8	82	72	56	24	9	3	1	1	
10	85	82	78	57	28	9	4	2	1

Fig. 6: Information Density for $X = 2$ conflict-free and $Y = 5$ reverse-complement-free DNA codes with varying N and E

Codeword Length (N)	Edit Distance (E)								
	1	2	3	4	5	6	7	8	9
2	0.79								
4	1.13	0.97	0.39						
6	1.05	1.04	0.68	0.47	0.17				
8	0.79	0.77	0.72	0.57	0.40	0.20			
10	0.64	0.63	0.62	0.58	0.48	0.32	0.20	0.10	

This is expressed as:

$$N \uparrow \Rightarrow \ C \uparrow$$

3) **Effect of Conflict-Free Requirement :**
Decreasing the conflict-free requirement (X) leads to more flexibility in the encoding scheme, allowing for a greater number of codewords. This relationship is captured by:

$$X \downarrow \Rightarrow \ C \uparrow$$

4) **Impact of Reverse-Complement Constraint :**
Increasing the maximum allowed length for reverse-complement (Y) relaxes constraints and results in more codewords. This is mathematically expressed as:

$$Y \uparrow \Rightarrow \ C \uparrow$$

5) **Relationships Involving Number of Codewords and Parameters :**
The number of codewords (C) is directly proportional to N, inversely proportional to E, directly proportional to Y, and inversely proportional to X. These relationships are expressed as:

$$C \propto N, \quad C \propto \frac{1}{E}, \quad C \propto Y, \quad C \propto \frac{1}{X}$$

6) **Relationships Involving Information Density and Parameters :**
Information density (ID) is inversely proportional to E, directly proportional to the number of codewords (C), inversely proportional to X, and directly proportional to Y. These relationships are summarized as:

$$ID \propto \frac{1}{E}, \quad ID \propto C, \quad ID \propto \frac{1}{X}, \quad ID \propto Y$$

These conclusions provide valuable insights into the design considerations and trade-offs involved in developing efficient and robust DNA coding schemes for various applications.

REFERENCES

[1] "Conflict free DNA codes", Author: Krishna Gopal Benerjee, Sourav Deb, Manish K. Gupta,

[2] "Improved Lower Bounds for Constant GC-Content DNA Codes", Author : Yeow Meng Chee and San Ling

[3] "Bounds on Reversible, Complement, Reversible-Complement, Constant Weight Sum Codes", Author : Krishna Gopal Benerjee, and Adrish Banerjee

[4] "On Secondary Structure Avoiding DNA codes with Reversible and Reversible-Complement Constraints", Author : Krishna Gopal Benerjee, and Adrish Banerjee

[5] "On Homopolymers and Secondary Structures Avoiding, Reversible, Reversible-Complement and GC-balanced DNA Codes", AUthor : Krishna Gopal Benerjee, and Adrish Banerjee