American Journal of Psychiatric Rehabilitation Expert Opinion Article

Exploring Various Representation of Genetic Sequences Via the Chaotic Approach

A. A. Navish[1*], L. Praveenkumar[2], G. Mahadevan[3], S. Riyasdeen[4], S. Ganesan[5]

[1*,2,3]Department of Mathematics, The Gandhigram Rural Institute (Deemed to be University), Dindigul, Tamil Nadu, India - 624 302.
[4]Department of Physics, Government Polytechnic College, Kottur, Theni, Tamil Nadu, India - 625 534.
[5]Department of Mathematics, Government Polytechnic College, Kaniyalampatti, Karur, Tamil Nadu, India - 621 301.

***Corresponding Author:** A. A. Navish
*Email: aa.navish2@gmail.com

Abstract: This study investigates the genetic codes of the Influenza virus through various visual representations, including matrix representation, chaos game representation ($\mathcal{CGR}$) and DNA walks. By applying chaotic concept to these genetic code visualizations, we can able to capture even the smallest changes quickly and precisely. Unlike previous studies that relied on outdated Java software, our research uses Python and the Biopython library, providing a modern and easily accessible approach. As a result, our work offers new insights into the chaotic nature of genetic sequences and presents valuable tools for mathematicians, biologists and computer scientists to better understand genetic data through fractal geometry.

Key Words: Fractal Analysis, Genetic Sequences, Influenza Virus, Fractal Dimension, Biopython

1 Introduction

Mathematical biology is an exciting and rapidly growing field that applies mathematical models to understand and analyze biological concepts and processes. Since grasping biological concepts can be challenging, mathematical modeling provides a way to overcome this difficulty by offering insights into aspects that are not directly observable. It allows us to describe biological systems using mathematical language, making complex processes more understandable. Over the past few decades, various branches of mathematics, such as calculus, probability theory, statistics, linear algebra, graph theory, algebraic geometry, topology, dynamical systems, differential equations and coding theory have been extensively used to study biological phenomena.

Despite the availability of these mathematical tools, analyzing biological structures remains a difficult task due to the irregularity and complexity of biological systems, which do not follow Euclidean geometry. In such cases, the one of the chaotic approach known as fractal analysis offers a promising solution for handling non-Euclidean objects. Fractals are irregular, self-similar structures that can be used to study complex biological patterns.

Genetic codes are fundamental units that serve as the blueprint for all biological processes, dictating how organisms grow, develop, and function at the molecular level. By studying these basic units, we can gain insights into the broader functioning of the entire organism. Inspired by this idea, our research focuses on understanding the genetic makeup of Influenza, a well-known zoonotic disease. To explore the genetic secrets of the Influenza virus, we convert its genetic codes into different visual representations using mathematical models. Although these visual representations appear similar, we use a parameter from the chaotic world called the fractal dimension to examine the characteristics of each representation.

The fractal dimension measures how the complexity of a pattern changes with scale. It reflects the space-filling potential of a pattern and is highly sensitive, with even small changes being detectable through fractional dimension values. In this study, we analyze different representations of the Influenza virus's genetic code, including matrix representations, indicator matrices, $\mathcal{CGR}$ and DNA walks. While many researchers have explored these genetic representations, only a few have used chaotic approaches like fractal analysis. Moreover, most of these studies relied on Java-based software, which is now outdated and no longer easily accessible.

Furthermore, this analysis is recently being done on the Covid investigations to understand and identify the structural behaviors of the coronavirus. Fernade et al. [3] conducted research on coronavirus based on the chaos theory. Hassan et al. [5] utilized fractal analysis to examine the geographical distribution of purine and pyrimidine. Mandal et al. [2] explored the publicly available SARS-CoV-2 genomes using a multifractal method. Sid Ali [7] used the wavelet transform to assess the signature of the mutated corona virus. In which many of them used java based software.

In contrast, our work is based on Python, a modern and widely used programming language. Specifically, we used Biopython and have provided the codes we

developed for our analysis. We hope that this work will be useful for mathematicians, biologists and computer-based researchers and providing a fresh perspective on analyzing genetic data through the lens of chaos theory.

An Quick Overview of Influenza

Influenza is a contagious RNA virus that mainly affects the respiratory system, causing seasonal flu outbreaks. It is classified into four types: A, B, C, and D, in which Influenza A being the most important for human infections. RNA viruses are unstable because their genetic material is more prone to errors during replication, leading to frequent mutations. This rapid mutation allows the virus to change quickly and evade the immune system.

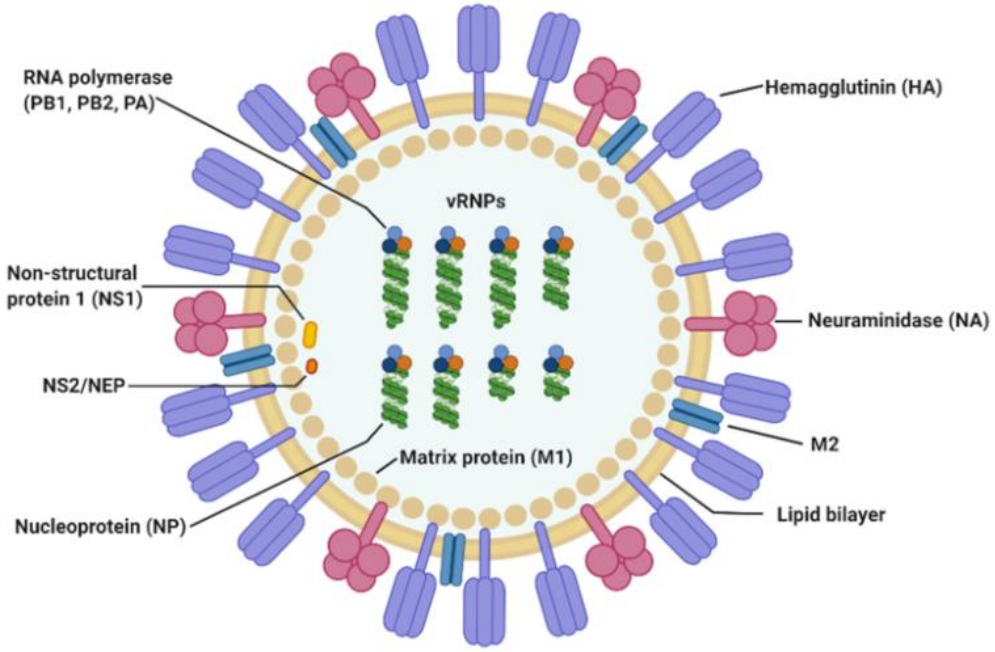


Figure 1: Segments of Influenza A

The Influenza A virus has 8 RNA segments (refer Figure 1). Each one has a specific job in helping the virus infect and spread. Segment 1 makes PB2, a protein that helps the virus copy its RNA inside cells. Segment 2 makes PB1, which works with PB2 to make more viral RNA. Segment 3 makes PA, another protein that helps in making RNA. Segment 4 makes Hemagglutinin (HA), a surface protein that helps the virus attach to and enter host cells. Segment 5 makes Nucleoprotein (NP), which binds to the virus's RNA to help it replicate. Segment 6 makes Neuraminidase (NA), a protein that helps the virus leave infected cells and spread. Segment 7 makes Matrix proteins (M1 and M2). M1 helps the virus assemble and leave the host cell, while M2 helps the virus uncoat inside the cell. Finally, Segment 8 makes NS1 and NS2. These proteins help the virus avoid the immune system and export viral RNA from the host cell. The most remarkable segments are HA and NA because they change to avoid immune detection. Also M2 and NS1 are important for the virus to escape the immune system and continue spreading.

2 Data Acquisition

The primary objective of this study is to explore the various representations of codon sequences using a chaotic approach. To ensure a comprehensive analysis, we do not limit our focus to a single sample sequence. Instead, we incorporate codon sequences from distinct Influenza A, allowing for a broader exploration. For our study, we have taken three datasets of H1N1 Influenza A. The first dataset corresponds to the initial strain discovered in 1918, famously known as the Spanish flu. The remaining two datasets are from the 2009 pandemic, collected from different regions and are commonly referred to as the swine flu. The codon sequences used in this study are sourced from the National Center for Biotechnology Information (NCBI) database, a reputable and extensive repository for biological data. Detailed information about the data is presented in the following Table 1. Throughout this study, A_1 represents the 1918 H1N1 virus (https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=88776), while A_2 and A_3 represent the 2009 H1N1 virus strains (https://www.ncbi.nlm.nih.gov/genomes/FLU/SwineFlu.html) obtained from two different regions.

Segment	Base Pairing (bp) Count		
	A_1	A_2	A_3
1	2280	2280	2280
2	2274	2274	2274
3	2151	2151	2151
4	1220*	1701	1701
5	1497	1497	1497
6	1410	1410	1410
7	982	982	982
8	838	863	863

Table 1: The base pair count of different segments of Influenza A, obtained from cDNA sequences on NCBI, is provided. The link to access these sequences is given in the Appendix (refer Table 5). The cDNA used in the experiment is complete, except for segment 4 of A_1, which is partial.

From the table, we see that most of the base pair counts are the same, but there are differences in segment 4 and segment 8. This suggests that, while the virus's overall structure stays the same, these two segments might have gone through changes. Sequence variation means the virus's genetic code has changed without affecting the segment's length. This usually happens when the virus mutates over time, which is called antigenic drift. RNA viruses like the flu often mutate to help them escape the immune system. On the other hand, changes in base pair count mean the virus has gone through insertions, deletions or reassortment, which can affect the virus's behavior (such as how easily it spreads or how it avoids the immune system). Both sequence changes and base pair count changes are important for understanding how the virus evolves and adapts and can help us track new outbreaks. These variations are key for scientists to develop better vaccines and treatments.

The Influenza virus is an RNA virus, but the genetic codes obtained from NCBI are typically in the form of cDNA, which is derived from the RNA of the virus. This is common practice in genomics because DNA is more stable and easier to manipulate than RNA. Moreover, the genetic data is often provided in .fasta format, which can be read into Python using the Biopython library. Specifically, the SeqIO.parse() function from Biopython allows you to read FASTA files and access the sequence data, including the sequence ID, description and nucleotide sequences for further analysis.

3 Distinct Representation of Codon Sequences

This section presents various representations of the considered genetic codes. While direct sequence comparison helps identify specific genetic differences, focusing on and developing different representation methods like matrix representation, $\mathcal{CGR}$ and DNA walks helps to reveal larger patterns, structural changes and evolutionary trends. These techniques simplify complex data and make it easier to spot mutations, hidden features and changes in the virus. They provide a broader view of the genome, speeding up analysis and helping researchers better understand viral behavior and evolution. As a result, not only biologists but also people with basic computer skills can analyze and explore the genetic sequences more easily.

3.1 Matrix Representation

Matrix representations of DNA sequences provide an effective way to visualize and analyze genetic data in a structured format. In this study, we utilize two types of matrices: the Difference Matrix and the Indicator Matrix, which enable efficient sequence comparison and the identification of key patterns.

3.1.1 Difference Matrix

To analyze DNA sequences effectively, they are often represented as numerical values or simplified mappings depending on the context [7]. These representations help to translate complex genetic data into forms that are easier to visualize and analyze computationally.

We can the DNA sequence as a function $f(x, y)$, where four colors are used to visually represent the nucleotides: Red for Adenine (A), Blue for Thymine (T), Green for Guanine (G) and Yellow for Cytosine (C). This color-coding method is particularly useful for visualizing genetic sequences, allowing for a clear distinction between the four bases. By assigning a distinct color to each nucleotide, the sequence is transformed into a visually comprehensible format. This color-coded representation enhances our ability to quickly detect patterns, such as repetitive sequences or GC content, which may be of biological significance. It also aids in the identification of structural features, such as palindromes, which are crucial in understanding processes like DNA replication and gene regulation.

However, we can identify the variation of sequences using color-coded images, the difference matrix quick the process. When comparing two color-coded DNA sequences, the difference matrix visually represents where the sequences match or differ

based on color. Identical colors (value 0) are represented by a black square, indicating that the colors at corresponding positions in both sequences are the same. Different colors (value 1) are represented by a white square, highlighting positions where the colors differ. Hence, the difference matrix is represented by the binary color. This visual representation is particularly useful for quickly identifying positions where the two sequences are identical or different, based on the colors in the image. The presence of white squares indicates the differences between the sequences, while black squares show where they match.

Here, we use the python function plot_dna_sequence_in_grid(dna_sequence) to create the color-coded image of a codon sequence and generate_difference_matrix (seq1, seq2) compares two codon sequences, which generate a difference matrix in binary color.

3.1.2 Indicator Matrix

The Indicator Matrix serves as an important analytical tool, making complex DNA sequence data more understandable. To explore the mathematical representation of a DNA sequence, consider the set $\mathcal{X}$ representing the four nucleotides of DNA, i.e.,

$$\mathcal{X} = \{A, T, C, G\}$$

The DNA sequence can be represented as a finite symbolic sequence, denoted by $\mathcal{S}$ and defined as:

$$\mathcal{S} = \{x_k\}_{k=1,2,3,\ldots,N}$$

where x_k represents the nucleotide at position k and $x_k \in \mathcal{X}$. Thus, the sequence $\mathcal{S}$ is a set of N values, each corresponding to a specific nucleotide in the sequence:

$$x_k = (k, x), \quad x \in \mathcal{X}$$

The DNA sequence can be represented using an indicator matrix, which is a sparse symmetric matrix with binary values $\{0,1\}$. This matrix, $M_{k,l}$, is defined as a function: $M: \mathcal{S} \times \mathcal{S} \to \{0,1\}$ such that

$$M_{k,l} = \begin{cases} 1 & \text{if} x_k = x_l \\ 0 & \text{if} x_k \neq x_l \end{cases}$$

This indicator matrix $M_{k,l}$ is a square matrix with dimensions $N \times N$. An example of this matrix is as follows:

.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
A	1	0	0	0	1	.	.	.
T	0	0	0	1	0	.	.	.
C	0	0	1	0	0	.	.	.
G	0	1	0	0	0	.	.	.
A	1	0	0	0	1	.	.	.
$M_{k,l}$	A	G	C	T	A	.	.	.

This traditional method enables the visual representation of the sequence in a 2D space, where a black dot is placed when $M_{k,l} = 1$ and a white spot when $M_{k,l} = 0$.

DNA sequences exhibit two primary types of base pairings: complementary base pairs (A-T, G-C) and non-complementary pairings (A-C, G-T, etc.). To capture this distinction, we modify the traditional $M_{k,l}$ function to better reflect these different pairing types. The modified function, denoted as $M^m_{k,l}$, is defined as: $M^m: \mathcal{S} \times \mathcal{S} \to \{0,1,2,3\}$ such that

$$M^m_{k,l} = \begin{cases} 1 & \text{if} x_k = x_l \\ 0 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{G, T\} \text{or} \{A, C\} \\ 2 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{T, C\} \text{or} \{A, G\} \\ 3 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{C, G\} \text{or} \{A, T\} \end{cases}$$

Thus, $M^m_{k,l}$ is a matrix that can take the values 0, 1, 2, or 3. An example of this matrix is:

.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
A	1	2	0	3	1	.	.	.
T	3	0	2	1	3	.	.	.
C	0	3	1	2	0	.	.	.
G	2	1	3	0	2	.	.	.
A	1	2	0	3	1	.	.	.
$M_{k,l}$	A	G	C	T	A	.	.	.

To further analyze the nucleotide pairings, this matrix can be decomposed into four binary matrices, $M^{m_0}_{k,l}, M^{m_1}_{k,l}, M^{m_2}_{k,l}, M^{m_3}_{k,l}$, as follows:

$$M^{m_0}_{k,l} = \begin{cases} 1 & \text{if} x_k = x_l \\ 0 & \text{otherwise} \end{cases}$$

$$M^{m_1}_{k,l} = \begin{cases} 1 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{G, T\} \text{or} \{A, C\} \\ 0 & \text{otherwise} \end{cases}$$

$$M^{m_2}_{k,l} = \begin{cases} 1 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{T, C\} \text{or} \{A, G\} \\ 0 & \text{otherwise} \end{cases}$$

$$M^{m_3}_{k,l} = \begin{cases} 1 & \text{if} x_k \neq x_l \text{and} \{x_k, x_l\} \in \{C, G\} \text{or} \{A, T\} \\ 0 & \text{otherwise} \end{cases}$$

By decomposing the indicator matrix into four distinct binary matrices, each reveals a different aspect of nucleotide pairings, providing a more detailed representation of the DNA sequence. This decomposition aids in advanced analyses, such as distinguishing complementary base pairs (A-T, G-C) from mismatches, highlighting conserved regions and areas of divergence for more efficient sequence alignment and pinpointing recurring patterns for motif detection. Overall, this approach enhances the depth and precision of DNA sequence analysis, making it a valuable tool for mutation detection, sequence alignment and genome-wide studies.

The indicator matrix allows for a more precise nucleotide-level comparisons, which the color-coded image cannot provide. While the color-coded image offers a quick visual representation, making it easy to spot patterns, similarities and differences at a glance, it lacks the detailed information necessary for tasks like sequence alignment and mutation detection. Thus, the indicator matrix provides the precision required for these in-depth analyses.

The function create_indicator_matrix(sequence) is used to generate the four binary indicator matrices

based on the given sequence. These matrices are created by comparing nucleotide pairs in the sequence and applying specific rules: $M_{k,l}^{m_0}$ is filled with 1's where the nucleotides are identical, $M_{k,l}^{m_1}$ is for pairs in the sets {G, T} or {A, C}, $M_{k,l}^{m_2}$ is for pairs in {T, C} or {A, G} and $M_{k,l}^{m_3}$ is for pairs in {C, G} or {A, T}.

3.2 Chaos Game Representation ($\mathcal{CGR}$)

The DNA sequence can be transformed into a two-dimensional real-valued representation using $\mathcal{CGR}$ [6], which helps preserve the statistical properties of the sequence while providing insight into both local and global patterns. In $\mathcal{CGR}$, each nucleotide in the sequence is mapped to a specific point in a 2D space, leading to a unique representation for every DNA sequence. Let's consider a DNA sequence represented as:

$$Seq = \{c_1, c_2, \dots, c_n, \dots, c_N\}$$

where c_n is the n^{th} nucleotide in the sequence, which is mapped to the coordinates $(c_x(n), c_y(n))$. The values of $c_x(n)$ and $c_y(n)$ are defined as:

$$c_x(n) = \begin{cases} 1 & \text{if} c_n = A \\ -1 & \text{if} c_n = T \\ -1 & \text{if} c_n = C \\ 1 & \text{if} c_n = G \end{cases}$$

and

$$c_y(n) = \begin{cases} 1 & \text{if} c_n = A \\ 1 & \text{if} c_n = T \\ -1 & \text{if} c_n = C \\ -1 & \text{if} c_n = G \end{cases}$$

By using these mappings, the sequence is transformed into $\mathcal{CGR}$ coordinates $(C_x(n), C_y(n))$, which can be calculated recursively starting from the initial point $(C_x(0), C_y(0)) = (0,0)$. The recursion equations are as follows:

$$C_x(n) = \frac{1}{2}[c_x(n) + C_x(n-1)] \quad n = 1,2,3, \dots, N$$

$$C_y(n) = \frac{1}{2}[c_y(n) + C_y(n-1)] \quad n = 1,2,3, \dots, N$$

The frequencies of genetic motifs, or n-mers, can be determined by partitioning the $\mathcal{CGR}$ space into grids and counting the occurrences of points in each sub-region. By splitting the $\mathcal{CGR}$ space into four quadrants, the top-right quadrant will represent sequences terminating in G. This method allows for the counting of bases, such as G and extends to counting other n-mers by recursively dividing the quadrants into smaller sectors that correspond to subsequences, such as GG, TG, AG and CG. This process continues iteratively, yielding counts for n-mers of increasing length. Figure 2 illustrates the relationship between n-mers and the sub-squares of the Chaos Game Representation.

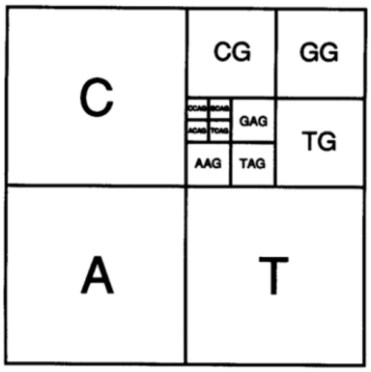


Figure 2: Relevance between n-mer and the sub-squares of CGR

An intriguing aspect of $\mathcal{CGR}$ is that visual representations of different sections of a genome exhibit a similar structure to that of the entire genome. Consequently, analyzing parts of a genome can still yield a reliable genomic signature. This property is especially useful when working with incomplete genome data, allowing for comparisons between non-homologous sequences.

The use of $\mathcal{CGR}$ is particularly beneficial in various bioinformatics applications, such as motif discovery, sequence comparison and the study of genomic structures. It enables researchers to identify recurring patterns in DNA sequences and compare different genomes efficiently. The $\mathcal{CGR}$'s ability to handle both complete and partial genomic data makes it an invaluable tool for analyzing large-scale genomic datasets, detecting mutations and identifying functional elements across different organisms.

Here, the function generate_cgr(sequence) generates the $\mathcal{CGR}$ of a given sequence. The sequence is mapped to a series of points in a 2D space based on the coordinates assigned to each nucleotide: A maps to (1, 1), T maps to (-1, 1), G maps to (1, -1) and C maps to (-1, -1). Starting at the center of the plot, each subsequent nucleotide generates a point that is the midpoint between the current point and the corner corresponding to the nucleotide's coordinates. The function count_n_mers(sequence, n) is used to count the occurrences of n-mers (subsequences of length n) in the given sequence.

3.3 DNA Walk

The development of chaos in genetic sequences can be explored using DNA walks, which visually represent genomic sequences where the steps correspond to the nucleotides in the DNA. This technique is used to assess the complexity of genes and the nucleotide variation within them. DNA walks serve as a symbolic method for expressing the contextual information of a DNA sequence as a graph, originating from the digital representation of DNA sequences [8].

A DNA walk is essentially the cumulative progression given by the sum $\sum X_n, n = 1,2,\dots,N-1$, where $X_n \in \{0,1,2,3\}$ represents the direction taken by each nucleotide. The DNA walk is a cumulative sum $\{X_0, X_0 + X_1, \dots, \sum_{k=0}^{N-1} X_k\}$ on the DNA string [1].

From the digital interpretation of a DNA sequence, we define the following sums:

$$a_n = \sum_{k=1}^{n-1} u(A, x_k), \quad t_n = \sum_{k=1}^{n-1} u(T, x_k), \quad c_n = \sum_{k=1}^{n-1} u(C, x_k), \quad g_n = \sum_{k=1}^{n-1} u(G, x_k)$$

These sums allow us to plot the DNA walk using the following coordinates:

$$W_n = \sin(a_n^2) - \sin(g_n^2), \quad V_n = \sin(t_n^2) - \sin(c_n^2)$$

Thus, a DNA walk forms a planar trajectory derived from the directional representation of the DNA sequence. The nucleotides A, T, C and G are encoded with the directions: west, east, south and north, respectively, creating a visual path that reflects the structure and variation within the sequence.

DNA walks are unique in that they provide a geometric representation of nucleotide sequences, where each step's direction is determined by the corresponding nucleotide. This visualization method is powerful in revealing patterns of sequence complexity and in detecting structural variations, such as mutations, repeats, or local variations in the genome. Additionally, DNA walks are particularly useful in understanding sequence periodicity and can be employed to study genetic sequences from various organisms, offering a visual comparison of sequence structures and their evolutionary dynamics.

In conclusion, DNA walks provide a distinct and informative method for exploring genomic sequences. Their ability to capture both the local and global characteristics of DNA makes them a valuable tool for genetic analysis, motif detection and sequence comparison. The use of directional encoding and cumulative summing allows for efficient visualization and analysis of nucleotide variations, which is especially useful in genomic studies where understanding sequence complexity and structure is critical.

The function compute_dna_walk(dna_sequence, step_size) generates a 2D DNA walk by mapping each nucleotide to a directional movement in the plane. The nucleotides A, C, G and T are mapped to North, South, East and West, respectively, based on the function encode_nucleotide(nucleotide, step_size). Starting at the origin (0, 0), the walk is constructed by iteratively applying the corresponding directional movement for each nucleotide in the sequence, resulting in two lists of coordinates (walk_x and walk_y). The walk is then visualized using the plot_dna_walk(walk_x, walk_y) function, which plots the walk on a 2D grid.

4 Exploring the Distinct Representations using Chaotic Approach

Most genetic code representations appear visually similar due to the same base pair count and very minute changes in their sequence. However, these subtle differences can reflect significant biological variations. To address the confusion caused by these similarities, we use a valuable parameter known as the fractal dimension.

Fractal dimension is a mathematical concept used to quantify the complexity of a structure or pattern. In the context of genetic sequences, fractal dimension helps capture the self-similar and non-linear patterns in the sequence, even when the changes are too small to be easily detected through direct comparison. In this case, fractal dimension allows us to distinguish between different representations of sequences that might appear similar at first glance but have underlying biological differences. There are various methods to calculate fractal dimension, but we selected the most suitable procedure based on the nature of the representation.

Typically, researchers use Java-based ImageJ or MATLAB to compute fractal dimensions. While MATLAB can create various representations of codon sequences, Python is often a better choice due to its flexibility, powerful libraries (e.g., NumPy, SciPy, Biopython) and memory efficiency, particularly when handling large genomic datasets. In contrast, ImageJ is designed for calculating fractal dimensions but is limited to working with pre-existing images. Python, however, enables both the creation of sequence representations and the calculation of fractal dimensions in a more integrated and efficient manner.

Definition 4.1 *The fractal dimension of the difference matrix and $\mathcal{CGR}$ is calculated using the traditional box-counting method. In this method, the given object is covered with small boxes of size δ. As the box size changes, the number of boxes $\mathcal{N}(\delta)$ required to cover the object varies. The fractal dimension can be calculated using the following expression:*

$$\mathcal{D}_B = \lim_{\delta \to 0} -\frac{\log \mathcal{N}(\delta)}{\log \delta}$$

This formula calculates the fractal dimension by analyzing how the number of boxes increases as the size of the boxes decreases, providing a measure of the object's space filling property. Hence, this method is particularly suitable for analyzing space-filling images, such as the binary images of the $\mathcal{CGR}$ and the difference matrix. These images are composed of black and white pixels, where the black pixels represent a certain structure or feature and

the white pixels represent the background or absence of that feature. In python, the function box_counting(binary_image, epsilon_values) performs the box-counting process.

Definition 4.2 *In the indicator matrix, the statistical properties of the nucleotide distribution remain consistent over time, suggesting a fractal pattern. The fractal dimension* $\mathcal{D}_I$ *quantifies this self-similar distribution and can be calculated using the following formula:*

$$\mathcal{D}_I = \frac{1}{N}\sum_{n=2}^{N}\frac{\log s(n)}{\log n}$$

where, N is the size of the matrix, n is the dimension of the $n \times n$ sub-matrices (minors) randomly selected from the $N \times N$ matrix and $s(n)$ is the average number of 1's in these $n \times n$ minors.

This formula measures how the occurrence of patterns (represented by 1's) in sub-matrices scales with size, providing a measure of the complexity or self-similarity of the nucleotide distribution. A higher fractal dimension indicates a more complex and irregular pattern, while a lower dimension suggests a simpler structure.
The function calculate_pn(matrix, N, step_size) computes the average number of 1's in $n \times n$ minors of the given indicator matrix. It iterates over different sizes n (from 2 to N) with a specified step size, counts the 1's in each minor and calculates the average.

Definition 4.3 *Local connected fractal dimension quantifies the connectivity of DNA walks. This measure is computed for each pixel of the given image and helps to highlight the irregularities of heterogeneous geometrical objects at a local level. It can be calculated using the following equation:*

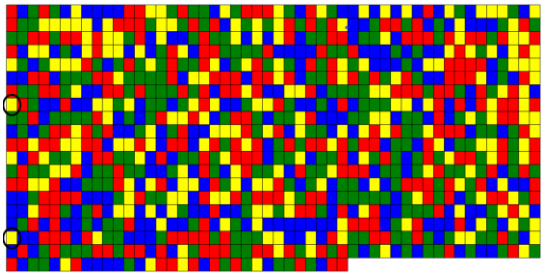
$$\mathcal{D}_L = \lim_{\delta \to 0} -\frac{\log N_\delta}{\log \delta}$$

where, N_δ is the average number of pixels per box for any box size δ.

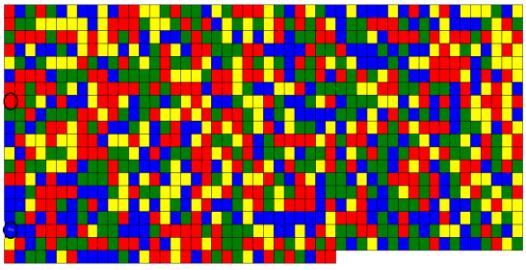
This formula computes the fractal dimension by analyzing the change in the average number of pixels per box as the box size δ decreases, giving a measure of the local geometric complexity. The function local_fractal_dimension is used to estimate the fractal dimension of a DNA walk (generated by compute_dna_walk) using the box-counting method. It divides the 2D space into boxes of varying sizes (ϵ) and counts the number of distinct boxes needed to cover the points of the DNA walk.
The local connected fractal dimension is more suitable for DNA walks because it focuses on how the path scales at smaller levels. It considers how the walk stays connected and evolves over time. This is important for understanding how DNA sequences change in space and time. On the other hand, box-counting works better for static objects and doesn't capture the dynamic and connected nature of DNA walks.

5 Results and Discussion

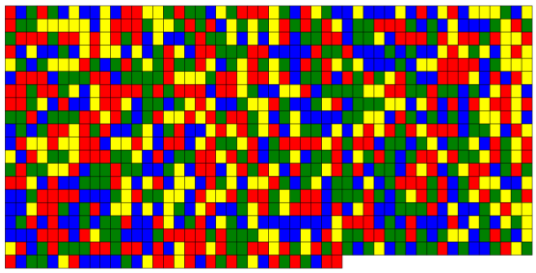
The various representations discussed in this paper already introduced, but they have not been regularized due to the lack of standardized software and further analysis. Java-based software tools such as Graph DNA and C-GREx are available; however, these are currently not functioning properly and cannot be customized according to specific needs. For the indicator matrix, we were unable to find any specialized software. Some attempts have been made using MATLAB, with users posting their work on platforms like MathWorks or the MATLAB GUI page. Unfortunately, we found these implementations difficult to run. Therefore, we have undertaken the task of consolidating these tools and approaches into a Python-based solution using the Biopython environment. The source codes used in this work are provided in the appendix and the various representations generated through these codes are shown in the following figures (which display representations for a sample sequence only).



(a) A_1



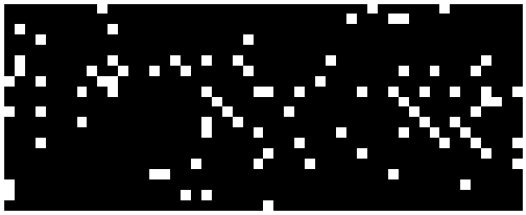
(b) A_2



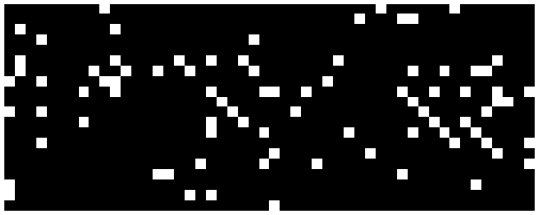
(c) A_3

Figure 3: Color-coded codon sequence of Segment 1

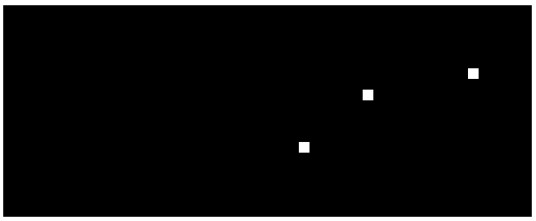
The color-coded image of segment 1 for all three viruses is presented in Figure 3. Through direct visual inspection, the differences between 3a and 3b can be easily identified, with one such difference highlighted in a circled area as an example. However, the differences between 3b and 3c are not immediately apparent at first glance and require a detailed comparison of each grid. In this case, the difference matrix presented in Figure 4 helps to identify these subtle differences more effectively.



(a) Difference between A_1 and A_2



(b) Difference between A_1 and A_3



(c) Difference between A_2 and A_3

Figure 4: The difference matrix derived from the color-coded representation of Segment 1

However, the difference matrix provides a quick visual way to identify differences, while the fractal dimension of the difference matrix offers a numeric representation. Since the box dimension is based on the space-filling property, it treats the white grid as an object in a black space and measures how much the white grid fills the black space. This means that as the difference between sequences increases, the fractal dimension increases and conversely, as the differences decrease, the fractal dimension also decreases. From the Table 2, we can observe that the differences between A_1A_2 and A_1A_3 are greater than the difference between A_2A_3. The dimension of 0 reflect that, the space is not filling with white grids (ie., there is no difference between the sequences).

Segment	A_1A_2	A_1A_3	A_2A_3
1	1.45801	1.45752	0.03057
2	1.50545	1.50585	0.01287
3	1.47559	1.47609	0.01462
4	-	-	0.00947
5	1.39537	1.39658	0.00960
6	1.47753	1.47753	0.00000
7	1.17443	1.16543	0.15493
8	-	-	0.00000

Table 2: Fractal dimension of difference matrix

Note that the difference matrix is created when the dimensions of the color-coded images are equal. However, for segments 4 and 8, the dimensions of the color-coded images do not match, so it is not possible to compute the difference matrix for these segments. The indicator matrix representation of a sample sequence is shown in Figure 5. The differences between the indicator matrices of A_1 and A_2 are visually identified and circled. However, the difference between the indicator matrices of A_2 and A_3 is not easily traceable due to their high similarity and minute differences.

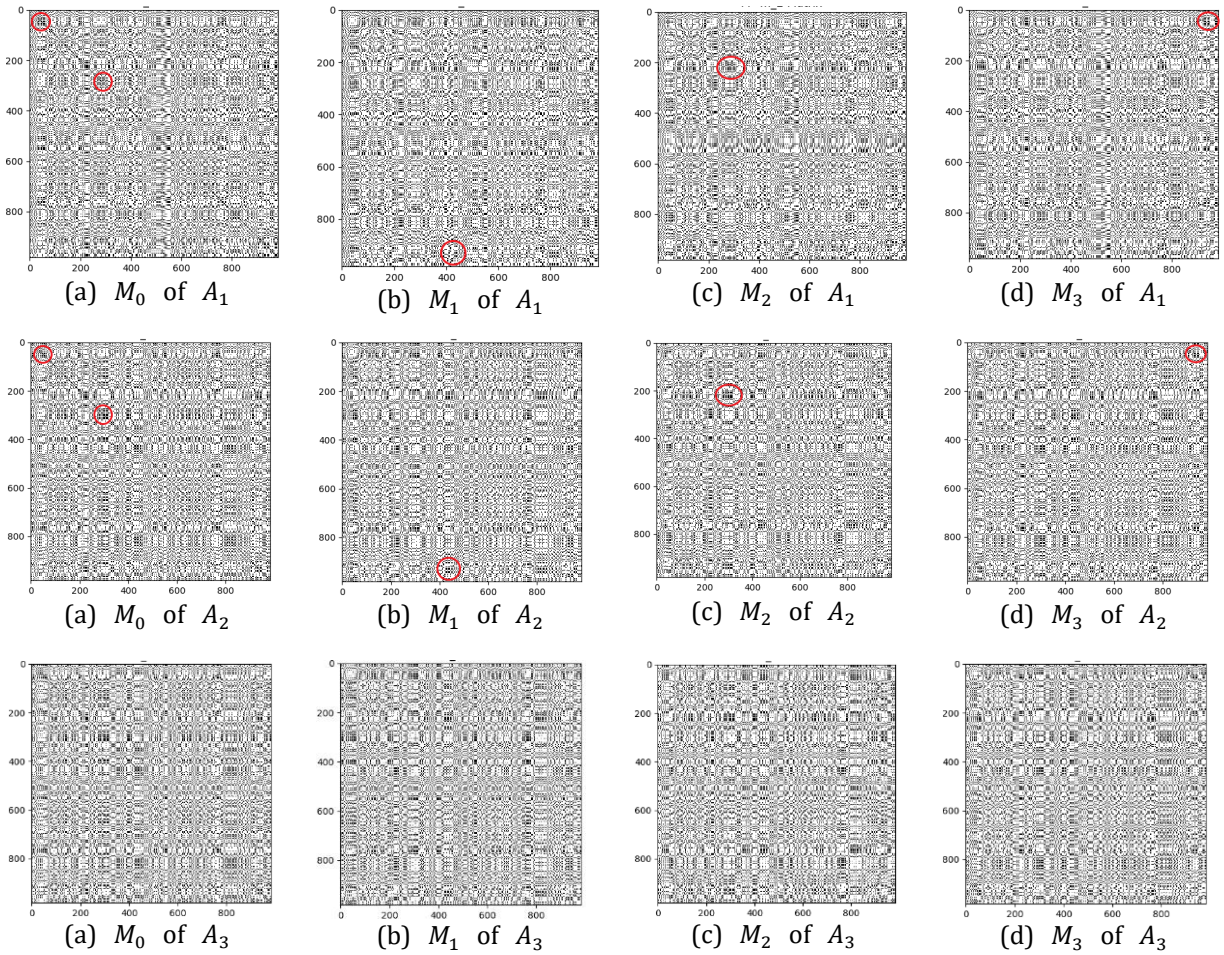


Figure 5: Indicator Matrix representation of Segment 7

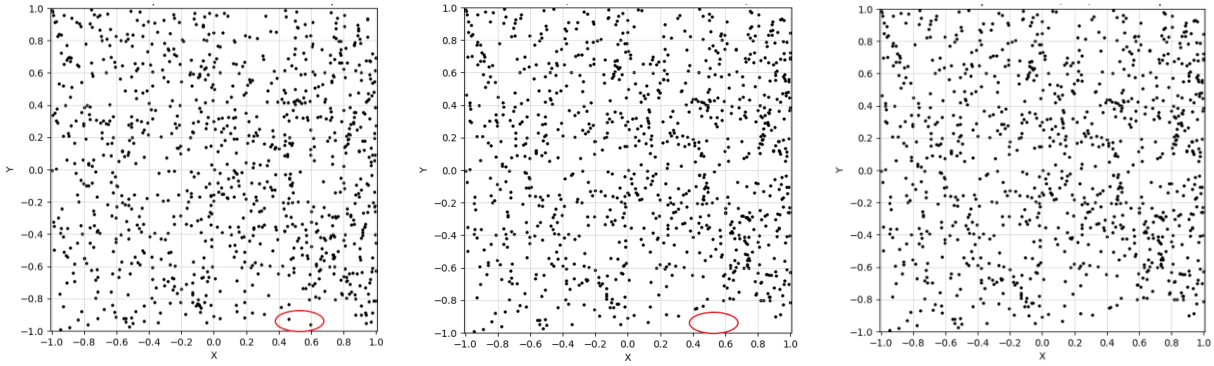
To detect the variation more accurately, the fractal dimension of the indicator matrix is calculated and presented in the Table 3. This dimension is based on the presence of 1's and 0's in the sub-matrix, so even a single change in a 1 or 0 is reflected in the fractal dimension. While the difference matrix fractal dimension provides an overall measure of sequence differences, the fractal dimension of the indicator matrix offers a more detailed analysis, as it evaluates changes across four different segments of the indicator matrix for a single sequence.

Segment	Type	M_0	M_1	M_2	M_3
1	A_1	1.878911	1.846561	1.864819	1.857823
	A_2	1.868991	1.823659	1.840250	1.837919
	A_3	1.869147	1.823815	1.840406	1.838075
2	A_1	1.858182	1.827732	1.838934	1.838192
	A_2	1.866857	1.821525	1.838115	1.835785
	A_3	1.867107	1.821775	1.838365	1.836035
3	A_1	1.857622	1.825272	1.843530	1.836534
	A_2	1.856416	1.827184	1.843674	1.835791

	A_3	1.856568	1.827336	1.843826	1.835943
4	A_1	-	-	-	-
	A_2	1.775760	1.721734	1.739187	1.748605
	A_3	1.777589	1.723563	1.741016	1.750434
5	A_1	1.611679	1.573359	1.606075	1.576470
	A_2	1.617814	1.568994	1.607246	1.573001
	A_3	1.619734	1.570914	1.609166	1.574921
6	A_1	1.572089	1.552232	1.558326	1.565157
	A_2	1.573024	1.550717	1.557899	1.566068
	A_3	1.573024	1.550717	1.557899	1.566068
7	A_1	1.421425	1.412315	1.420521	1.412649
	A_2	1.422638	1.410963	1.419216	1.414084
	A_3	1.422638	1.410963	1.419216	1.414084
8	A_1	1.425017	1.408589	1.418499	1.414721
	A_2	1.428195	1.405775	1.414920	1.417798
	A_3	1.428195	1.405775	1.414920	1.417798

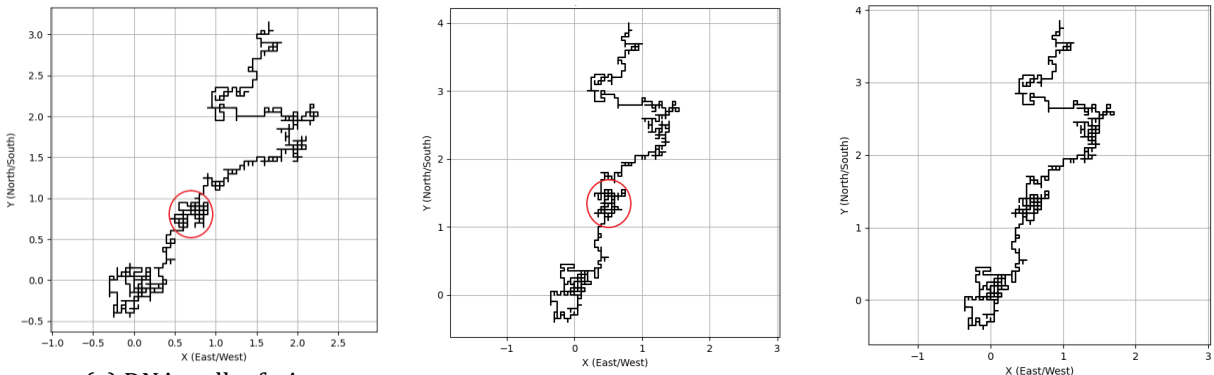
Table 3: Indicator dimension of indicator matrix

The $\mathcal{CGR}$ and DNA walk representations of a sample segment are presented, with visually identifiable differences circled (refer Figures 6 and 7).



(a) $\mathcal{CGR}$ of A_1 (b) $\mathcal{CGR}$ of A_2 (c) $\mathcal{CGR}$ of A_3

Figure 6: $\mathcal{CGR}$ representation of Segment 1



(a) DNA walk of A_1 (b) DNA walk of A_2 (c) DNA walk of A_3

Figure 7: DNA walk of Segment 7

Segment	Type	CGR	DNA Walk
1	A_1	1.3985	0.8783
	A_2	1.3911	0.8972
	A_3	1.3915	0.8949
2	A_1	1.3900	0.8511
	A_2	1.3940	0.8387
	A_3	1.3938	0.8453
3	A_1	1.3800	0.9006
	A_2	1.3820	0.8979
	A_3	1.3821	0.8967
4	A_1	-	-
	A_2	1.3262	0.8437
	A_3	1.3268	0.8375
5	A_1	1.2888	0.8975
	A_2	1.2950	0.8927
	A_3	1.2953	0.8906
6	A_1	1.2846	0.8799
	A_2	1.2845	0.8861
	A_3	1.2845	0.8861
7	A_1	1.1925	0.9186
	A_2	1.1898	0.9371
	A_3	1.1893	0.9245
8	A_1	1.1492	0.9081
	A_2	1.1524	0.8868
	A_3	1.1524	0.8868

Table 4: Fractal dimension of $\mathcal{CGR}$ and DNA walk of the segments

However, to detect the differences more precisely, the fractal dimensions are calculated and listed in Table 4. From the table, it can be seen that although the number of dots in the $\mathcal{CGR}$ is the same for segments with the same base pairs from different viruses, their fractal box dimensions differ. This indicates that the position of the dots also influences the fractal dimension. Similarly, even though the number of moves in the DNA walk is the same, the local connected fractal dimension changes due to variations in the local direction of movement.

The same dimension for the same sequence in different viruses indicates that the sequences are exactly identical. Specifically, from Tables 2, 3 and 4, we can conclude that segments 6 and 8 of A_2 and A_3 are identical. From Tables 3 and 4, the fractal dimension of different segments of A_2 and A_3 are almost closely near compare with A_1. Thus, the codon sequence of various segments of A_2 and A_3 should be more similar than A_1.

6 Conclusion

Though this study focused on the codon sequences of the Influenza A virus, it integrated three key perspectives: the exploration of various codon sequence representations, the significance of fractal dimension analysis, and the development of Python code to conduct these analyses. The different representations of codon sequences and their corresponding fractal dimensions were unique in their own right, each helping to identify differences between the sequences. Specifically, the difference matrix and DNA walk provided quick visual insights. However, subtle changes that could have significant biological implications were not always easily detectable. In such cases, fractal dimension analysis effectively captured these variations. As a result, fractal dimension analysis proved to be a more precise and reliable method for identifying differences in visually similar representations. The application of such analytical methods in bioinformatics provided a fresh perspective and helped bridge the gap in understanding biological data, especially for non-biologists.

Moreover, these types of studies enhance the automatic detection of exact viral strains and their variants. By employing fractal dimension analysis alongside traditional methods, this study improves the sensitivity and accuracy of identifying minute differences between strains. This is crucial for tracking virus mutations and understanding their evolutionary patterns. Such advancements in bioinformatics enable faster and more reliable viral surveillance, facilitating the detection of emerging variants and potentially allowing for quicker responses to public health threats.

References

1. Cattani, C. (2010). Fractals and hidden symmetries in DNA. Mathematical problems in engineering, 2010.
2. Conway, M. J. (2020). Identification of

coronavirus sequences in carp cDNA from Wuhan, China. Journal of medical virology, 92(9), 1629-1633.
3. e Fernandes, T. D. S., de Oliveira Filho, J. S., & da Silva Lopes, I. M. S. (2020). Fractal signature of coronaviruses related to severe acute respiratory syndrome. Research on Biomedical Engineering, 1-5.
4. Hassan, S. S., Choudhury, P. P.,& Bose, A. (2012). A quantitative model for human olfactory receptors. Nature Precedings, 1-1.
5. Hassan, S.S.; Kumar Rout, R.; Sharma, V. A Quantitative Genomic View of the coronaviruses: SARS-COV2. Preprints 2020, 2020030344.
6. Nair, A. S., Nair, V. V., Arun, K. S., Kant, K., & Dey, A. (2009). Bio-sequence signatures using chaos game representation. In Bioinformatics: Applications in Life and Environmental Sciences (pp. 62-76). Springer, Dordrecht.
7. Ouadfeul, S. A. (2020). Fractal signatures of SARS-CoV2 coronavirus, the indicator matrix, the fractal dimension and the 2D directional wavelet transform: A comparative study with SARS-CoV, MERS-CoV and SARS-like coronavirus. bioRxiv.
8. Navish, A. A., & Uthayakumar, R. (2023). A chaotic approach to recognize the characteristics of genetic codes of covid patients. Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization, 11(3), 399-412.

Appendix

Segment	Type	Link
1	A_1	https://www.ncbi.nlm.nih.gov/nuccore/DQ208309.1https://www.ncbi.nlm.nih.gov/nuccore/DQ208309.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065759https://www.ncbi.nlm.nih.gov/nuccore/CY065759
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569666https://www.ncbi.nlm.nih.gov/nuccore/HM569666
2	A_1	https://www.ncbi.nlm.nih.gov/nuccore/DQ208310.1https://www.ncbi.nlm.nih.gov/nuccore/DQ208310.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065760https://www.ncbi.nlm.nih.gov/nuccore/CY065760
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569665https://www.ncbi.nlm.nih.gov/nuccore/HM569665
3	A_1	https://www.ncbi.nlm.nih.gov/nuccore/DQ208311.2https://www.ncbi.nlm.nih.gov/nuccore/DQ208311.2
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065761https://www.ncbi.nlm.nih.gov/nuccore/CY065761
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569664https://www.ncbi.nlm.nih.gov/nuccore/HM569664
4	A_1	https://www.ncbi.nlm.nih.gov/nuccore/AF116575.1https://www.ncbi.nlm.nih.gov/nuccore/AF116575.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065762https://www.ncbi.nlm.nih.gov/nuccore/CY065762
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569659https://www.ncbi.nlm.nih.gov/nuccore/HM569659
5	A_1	https://www.ncbi.nlm.nih.gov/nuccore/AY744935.1https://www.ncbi.nlm.nih.gov/nuccore/AY744935.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065763https://www.ncbi.nlm.nih.gov/nuccore/CY065763
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569662https://www.ncbi.nlm.nih.gov/nuccore/HM569662
6	A_1	https://www.ncbi.nlm.nih.gov/nuccore/AF250356.2https://www.ncbi.nlm.nih.gov/nuccore/AF250356.2
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065764https://www.ncbi.nlm.nih.gov/nuccore/CY065764
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569661https://www.ncbi.nlm.nih.gov/nuccore/HM569661

7	A_1	https://www.ncbi.nlm.nih.gov/nuccore/AY130766.1https://www.ncbi.nlm.nih.gov/nuccore/AY130766.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065765https://www.ncbi.nlm.nih.gov/nuccore/CY065765
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569660https://www.ncbi.nlm.nih.gov/nuccore/HM569660
8	A_1	https://www.ncbi.nlm.nih.gov/nuccore/AF333238.1https://www.ncbi.nlm.nih.gov/nuccore/AF333238.1
	A_2	https://www.ncbi.nlm.nih.gov/nuccore/CY065766https://www.ncbi.nlm.nih.gov/nuccore/CY065766
	A_3	https://www.ncbi.nlm.nih.gov/nuccore/HM569663https://www.ncbi.nlm.nih.gov/nuccore/HM569663

Table 5: A detailed link of the experimented NCBI nucleotide sequence.

You can get all the Python code used in this work in the RAR file. You can download the RAR file from this https://drive.google.com/file/d/1YuF7rrBDqMfXRwqTVGZD_NkLWSQkDiz7/view?usp=sharing