An Algorithm Transform DNA Sequences to Improve Accuracy in Similarity Search

Hoang Do Thanh Tung, Phuong Vuong Quang
Institute of Information Technology, Vietnam Academy of Science and Technology, HaNoi, Viet Nam

***Abstract*—Similarity search of DNA sequences is a fundamental problem in the bioinformatics, serving as the basis for many other problems. In this, the calculation of the similarity value between sequences is the most important, with the Edit distance (ED) commonly used due to its high accuracy, but slow speed. With the advantage of transforming the original DNA sequences into numerical vector form that retaining unique features based on properties. The calculation processing on these transformed data will be much faster, many times faster than a direct comparison on the original sequence. Additionally, from a long DNA sequence, after transformation, it typically has a lower storage capacity, making it have good data compression. The challenge of this job is to develop algorithms based on features that maintain biological significance while ensuring search accuracy, which is also the problem to be solved. Previous methods often used pure mathematical statistics such as frequency statistics and matrix transformations to construct features. In this paper, an improved algorithm is proposed based on both biological significances and mathematical statistics to transforming gene data into numerical vectors for ease of storage and to improve accuracy in similarity search between DNA sequences. Based on the experimental results, the new algorithm improves the accuracy of similarity calculations while maintaining good performance.**

***Keywords*—Similarity search; data transformation; DNA sequence; big data**

I. INTRODUCTION

Bioinformatics is an interdisciplinary field that develops software methods and tools for understanding biological data, especially when the data sets are large and complex. It combines technologies of applied mathematics, statistics, computer science, biology, chemistry, physics… and biological mathematics. The term bioinformatics is a part of computational biology, and the combination of these sciences is intertwined and mutual. Thus, the research results in this field not only contribute to biology, but also to other fields. Biological data contains gene sequences (DNA - Deoxyribonucleic Acid) and briefly describes which is made up of four nucleotides: A, C, T and G. DNA is a crucial molecule for all living things, not just humans. The application of DNA in science and daily life is diverse and significant, such as crossbreeding, genetic mutation, comparison of species, prediction of phylogenetics, pedigree, genetic diseases, predict disease risk, etc. The fundamental problem in bioinformatics related to DNA sequence processing is similarity search, which is finding subsequences that are the same or similar to a sequence of interest. This leads to further problems such as calculating the similarity value between sequences to draw a phylogenetic tree and sequence alignment to achieve maximum similarity between sequences. These problems are manageable on small and medium data but become complex with big data.

In similarity search, the commonly used method to calculate similarity value between two sequences is Edit Distance (ED) (also known as Levenshtein). The *ED* similarity value between two sequences is the minimum number of steps required to transform one sequence to other, based on three transformations: adding, editing, and deleting each character in the sequence [1]. For example, similarity value between LOVE and MOVIE is ED(LOVE, MOVIE) = 2 because two steps of LOVE → **M**OVE → MOV**I**E are needed. The advantage of this method is can compare sequences of different lengths, thereby flexibly being applied in many problems. However, its computational complexity is $O(n*m)$ where m, n are the lengths of two sequences being compared. With long data, the results take a long time to obtain, making comparisons in long sequences less efficient. Due to the rapid advancements in technology, biological databases generate vast amounts of information and are continuously growing, resulting in a rapid increase in their size.

The large size of the data leads to a high number of I/O operations, resulting in a high cost of storage space and decreased processing performance. To address this issue, researchers are exploring various methods to index or transform the gene data into numerical form to reduce database access and improve query performance. One of the challenges in this field is to develop algorithms for transforming the sequence in a way that retains the biological significance of the data while reducing its size. Previous methods often utilized simple mathematical statistics, such as frequency statistics or matrix transformations, to build features. In this study, an improved algorithm is proposed that not only takes into account frequency, position appearance, and correlation between position and distance of the characters but also considers the biological significance of the sequence, based on the role and function of amino acids in proteins. This approach helps to retain as many features of the sequence as possible, leading to improved accuracy when searching for similarity between two sequences.

The rest of the paper is organized as follows, Section II will present the related work to the research, Section III presents the proposed improved algorithm, Section IV presents the experimental results of the algorithm proposal, and Section V is the conclusion.

II. Related Work

A lot of research has been carried out to calculate similarity between sequences. Through the survey, a number of published methods were found. Some typical algorithms such as: Smith - Waterman [2] and BLAST [3] are well-known algorithms for performing sequence comparisons through representative sequences. This group of algorithms has high accuracy, but limitation is inefficient when size of database is too large or continuously added. Many other algorithms, improved from the famous BLAST algorithm, also allow performing calculations quickly such as Flash algorithm [4], ProperSearch tool [5], CAFE method [6]. BIS (Bitmap indexing structure) [7], IDC (Incrementally Decreasing Cover) [8], FRESCO (Framework for REferential Sequence Compresion) [9], Modified HuffBit Compress Algorithm [10]. These algorithms perform and compression search by interfering with data structures such as matrix analysis, genomic statistics, frequency changes, etc. In addition, Metric space indexing techniques [4], Williams & Zobel [11] and Ozturk and Ferhatosmanoglu [12] are indexing methods that use special transformations that convert the input DNA sequence into numerical vectors based on features. The objective of these methods is to find the most effective features to generate the vectors, so as to preserve the biological information of the original sequence and perform efficient similarity searches. Studies by Kahveci and Singh [13], Yongkun Li et al [14] have found that using frequency and position-distance characteristics resulted in better performance compared to other features.

The *Hfwd*^{2} method [1] is a technique that partitions a long DNA sequence into smaller subsequences using a window length, and then transforms these windows into numerical vectors that represent the frequency of appearance of the characters (A, C, T, G) as well as their combinations (N-grams). To enhance the features stored in the vector, this method also incorporates the position of each character in the sequence. To address the issue of uneven distribution of characters in the DNA sequence, the authors divide the sequence into two equal parts and generate the vectors based on these two parts to ensure that the comparison is more equal. While this method results in a lower accuracy compared to direct comparison due to the noise introduced during the transformation process, it offers a significant improvement in performance and speed during similarity searches. However, *Hfwd*^{2} has a few limitations, such as the use of a simple positional parameter, and the lack of consideration for the biological significance of the characters in the sequence. Below, we propose an improved algorithm from *Hfwd*^{2}, in which numerical vectors transformation is not only built with features based on frequency, position appearance, correlation between position and distance way of characters, but also interested in the biological significance of sequences based on amino acids, applied on wavelet transform to increase features stored after transformation. Theoretically, it is expected to improve accuracy of calculating similarity value between sequences.

Current methods are still continuing to research solutions to improve accuracy of similarity search with transformation vectors. In this paper, we propose an improved algorithm for transforming gene sequences into numerical vectors. This algorithm considers not only mathematical statistics such as frequency and position appearance, but also the biological significance of amino acids in the sequences. By doing so, we aim to maintain as much information as possible from the original sequence while improving the accuracy of similarity searches.

III. Proposed Method

The transformation of original sequences into numerical vectors has been discussed in the above sections. This approach can optimize storage in databases and increase computational performance. The challenge is to preserve the biological meaning of the sequences in the numerical vectors after transformation, in order to maintain search accuracy. To address this, we propose an improved algorithm compared to *Hfwd*^{2}. This new algorithm considers not just the frequency, position appearance, and correlation between position and distance of characters, but also the biological significance of the sequences based on amino acids. Additionally, the use of wavelet transform increases the amount of information stored in the numerical vectors. Theoretically, it is expected to improve accuracy of calculating similarity value between sequences.

In the pre-processing stage, we also partitioned the DNA sequences into equal length windows and transform each partition into numerical vectors using the feature parameters we have proposed. These vectors are then stored in a centralized database for later retrieval. When a new sequence needs to be compared for similarity, it will undergo the same pre-processing step and then be compared with the vectors in the database. The resulting set of similarities will be outputted. The general process of the method is shown in Fig. 1.

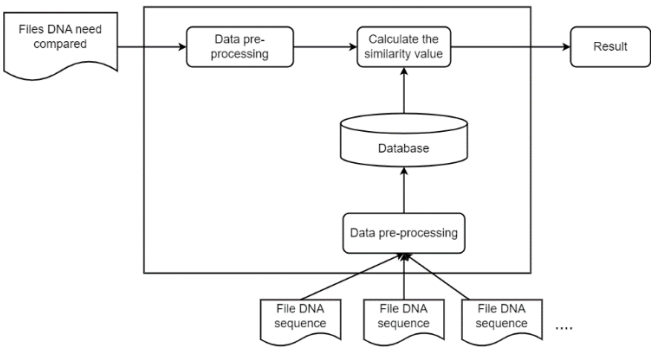

Fig. 1. General process of the method.

Files containing DNA sequences after being collected will be sent to the data pre-processing module. The system will remove any characters that are not A, C, T, G from the sequence, and partitioned the original sequence into equal length windows. The length of these windows can be adjusted to meet the user's needs, and we will explore the optimal range for best results in the experimental section. These operations can be easily performed using support functions from the C# language's String library. Finally, the partitions are subjected to transformation algorithms to generate multidimensional numerical vectors.

These vectors will be indexed so that when making comparisons based on similarity, they will be much faster than

the original sequence. The pre-processing model is depicted in Fig. 2.

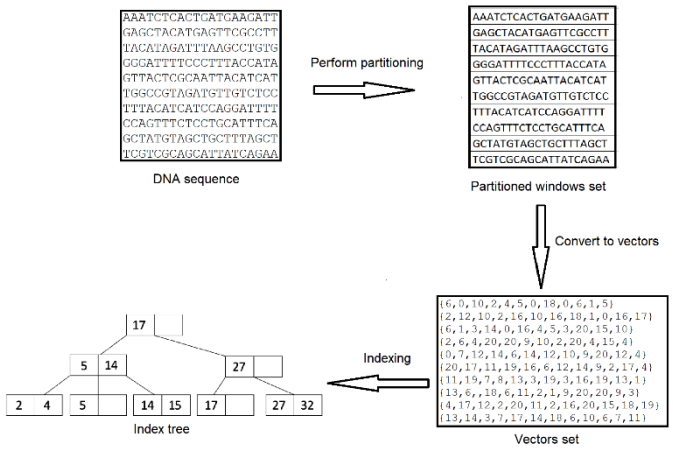

Fig. 2. Data pre-processing and indexing model.

A. Algorithms Feature Extraction for DNA Sequence Transformation

Just like feature extraction in natural language processing, the goal of this step is to show how to transform text data into vectors with numerical values. This is often the most important step that determines whether the final approach is successful for a real problem [15]. Here, we will present a proposal of four features that can be used to generate a numerical vector from the input DNA sequence. These features help to capture important information about the DNA sequence for perform similarity calculation.

Given the sequence S = {s_0, s_1, s_2, ..., s_{l-1}} where l is the length of the S, the algorithms to build feature of the sequence S are as follows:

1) Algorithm transform based on combinatoric: Let ω = {a_1, a_2, ..., a_α} where N-grams is the combination of characters built in the sequence S, α is length of the combination of characters. The DNA sequence S will only includes four nucleotides A, C, G, T, so when choosing N=1 we have the combination ω = {A, C, G, T}, when N=2 the combination ω = { AA, AC, AG, AT, TA, TC, TG, TT, GA, GC, GG, GT, TA, TC, TG, TT}, and so on. The larger N gets, the combination grows exponentially. Each element in the ω combination may or may not appear in the original sequence S, and may appear in different positions, which creates unique characteristics for each DNA sequence. This algorithm adds a parameter to calculate the correlation between position and distance compared to Hfwd[2]. The purpose is to store the parameters of frequency, position, and correlation between position and distance of characters that appear in the sequence, and use them as unique features of each sequence to calculate the similarity value later.

a) Frequency appearance

$$W_{x\alpha} = \begin{cases} 1, x_\alpha = s_i \\ 0, x_\alpha <> s_i \end{cases} \quad (1)$$

Where i = 1,2…, l; $x_\alpha \in \omega$; if character x_α appear in position s_i under consideration, it has 1, otherwise it has 0.

$$a_{x\alpha} = \sum_{i=1}^{l} w_{x\alpha}(s_i) \quad (2)$$

$a_{x\alpha} \geq 0$ is total appearance of the x_α in the sequence S.

b) Position appearance

$$b_{x\alpha} = \sum_{i=1}^{l} i * w_{x\alpha}(s_i) \quad (3)$$

$b_{x\alpha} \geq 0$ is sum of position values of x_α in the sequence S.

c) Correlation between position and distance

$$c_{x\alpha} = \frac{\sum_{i=1}^{l} (i * w_{x\alpha}(s_i)) * (i * w_{x\alpha}(s_i) - i_{pre} * w_{x\alpha}(s_{i\_pre}))}{a_{x\alpha}} \quad (4)$$

$c_{a\alpha} \geq 0$ is average of (position * distance), distance calculated as the difference between the current position and the previous position of x_α in the sequence S.

- N-grams selection

With DNA sequences, when N=1 the combination of characters has the value $4^1 = 4$, when N=2 the combination has the value $4^2=16$, when N=3 the combination has the value $4^3=64$, etc. Increasing N can increase the amount of information stored in the vectors, but it also increases the computational cost of generating the vectors and calculating the similarity. According to a suggestion in [1], and based on our own experiments selecting N=1,2,3,4, we has been found that N=2 gives good results for comparisons and is also computationally efficient.

Algorithm 1. CombinatoricTranform(subsequence S)

INPUT: subsequence S
OUTPUT: a vector combinatoric 48 dimensions
BEGIN
Initialize ω = {AA, AC, AG, AT, TA, TC, TG, TT, GA, GC, GG, GT, TA, TC, TG, TT}.
Initialize vector vCombinatoric 48 dimensions
FOR (i=0 to 15) **{**
vCombinatoric[i] = Count frequency of element in ω (by fomula(2)) **}**
FOR (i=16 to 31) **{**
vCombinatoric[i] = Calculate sum of localtion of element in ω (by fomula(3)) **}**
FOR (i=32 to 47) **{**
vCombinatoric[i] = Calculate value of correlation between position and distance of element in ω (by fomula(4)) **}**
RETURN vCombinatoric
END

The pseudocode for Algorithm 1 outlines the steps to transform an input DNA subsequence into a 48-dimensional numerical vector based on the formulas (1), (2), (3), and (4).

For example, with the sequence S = {AGTAGTGCTA}. We can calculate CombinatoricTranform(S) = {0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 4, 0, 6, 0, 0, 0, 0}.

2) Algorithm transform based on covariance: Covariance is a measure of how much two random variables vary together. It's similar to variance, but where variance tells you how a single variable varies, covariance tells you how two variables vary together. Theoretically, covariance is used by analysts as a way to look at the overall for one or more related variables. In the context of biological and genetic data, variables are specific positions of nucleotides in the gene. Genetic covariance studies whether two positions of nucleotides on the gene evolve independently or they evolve together [16]. The authors in [17] also presented the idea of using covariance as a feature represented when transform the DNA sequence. This feature could provide additional information about the correlation between nucleotides in the gene, which could be useful in further analysis and comparison with other DNA sequences.

$$cov_{x,y} = \frac{\sum\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)}{N-1} \quad (5)$$

Where $(x, y) \in$ {(A,T); (A;G); (A,C); (T,G); (T,C); (G,C)}, x_i, y_i is position of x, y on the sequence, $\bar{x}, \bar{y}$ is average position value of x, y in the sequence.

We use the covariance formula to build features through the vector $\{ cov_{A,C}, cov_{A,G}, cov_{A,T}, cov_{T,C}, cov_{T,G}, cov_{G,C} \}$ showing correlation between the positions of the nucleotides together.

The pseudocode for Algorithm 2 explains how to transform an input subsequence into a 6-dimensional vector based on formula (5).

Algorithm 2. CovarianceTranform(subsequence S)

```
INPUT: subsequence S
OUTPUT: a vector covariance 6 dimensions
BEGIN
    Initialize ω = {(A,T); (A;G); (A,C); (T,G); (T,C); (G,C)}.
    Initialize vector vCovariance 6 dimensions
    FOR (i=0 to 15) {
        vCovariance[i] = Calculate value of correlation
        between position and distance of element in ω (by
        fomula(5))
    }
    RETURN vCovariance
END
```

For example, with the given sequence S, we can calculate CovarianceTranform(S) = { -1.33, -2.3, -3, -1.6, -2.8, -1.24}.

3) Algorithm transform based on Haar wavelet: The Discrete Wavelet Transform (DWT) is a widely used method in digital signal processing, data mining, information retrieval, text clustering and classification, digital image processing, etc. due to its simplicity and efficiency. DWT involves dividing a signal into two parts: high frequency and low frequency. The low frequency part is further divided into high and low frequency parts through a process called downsampling. The encoding complexity is linear and allows for multiple resolution levels. The Haar filter, being the simplest possible wavelet, is often used in the analysis of signals with abrupt transitions and is commonly used in analyzing time series or ordinal data. We use Haar filter to divide the original sequence into two parts: approximation (by summing) and detail (by subtracting) in pairs of values in in two vectors.

In the case that a character is unevenly distributed in different regions of the DNA sequence, for example, if character A is abundant in the first half but scarce in the second half, the algorithm for transforming based on these features can have lower efficiency. The goal of applying Haar wavelet is to overcome this limitation, improve the accuracy when calculating and comparing the similarity values of the transformed sequences. The pseudocode for Algorithm 3 presents the process of using Haar wavelet to transform vectors.

Algorithm 3. WaveletHaarTranform(vector u, vector v, integer *l*)

```
INPUT: vector u,v have l dimensions
OUTPUT: a vector l*2 dimensions
BEGIN
    Initialize vector vHigh, vLow l dimensions
    FOR (i=0 to l-1) {
        vHigh[i] = u[i] + v[i]
        vLow[i] = u[i] - v[i]
    }
    Initialize vector vWavelet l*2 dimensions
    FOR (i=0 to l-1) {
        vWavelet[i] = vHigh[i]
    }
    FOR (i=l to l*2 - 1) {
        vWavelet[i] = vLow[i]
    }
    RETURN vWavelet
END
```

4) Algorithm transform based on codon: The human body, along with other organisms, produces proteins through a process called transcription. Genes in biological cells store the information needed to construct proteins in the form of DNA sequences. During transcription, the DNA information is transcribed into messenger RNA (mRNA), which carries the genetic information. This mRNA sequence is then translated into a sequence of amino acids, which make up the proteins. Researchers have discovered that a group of three nucleotides encodes a single amino acid [18]. With four nucleotides, A, C, T, and G, there are 64 possible codons and 20 different amino acids.

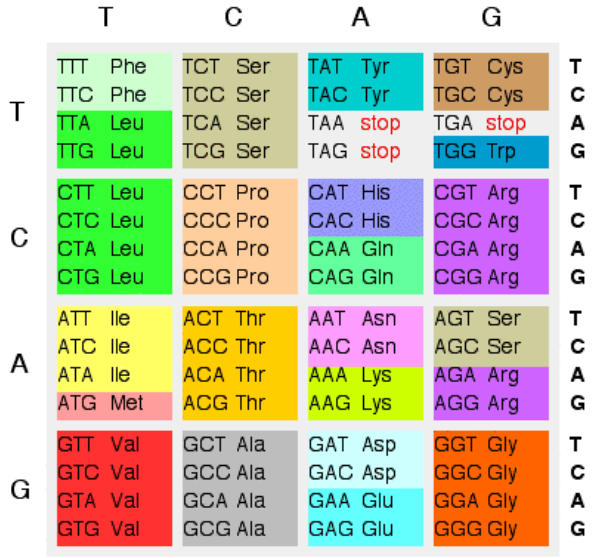

Fig. 3. Table mapping three nucleotides to codon.

Fig. 3 shows how the various combinations of three bases in the coding strand of DNA are used to code for individual amino acids - shown by their three letter abbreviation. The combination of codons determines the sequence of amino acids in the resulting protein, which ultimately determines its structure and function. In this way, codons are the key to translating genetic information stored in DNA into the functional proteins that carry out vital cellular processes in all living organisms. Almost the same as above, we will use frequency appearance and correlation of position and distance to store the feature. The goal is to capture features related to biological significance and store them in vectors after transforming the input sequences. Thereby improving the efficiency when calculating and comparing the similarity value.

Let β a collection of amino acids, β = {Phe, Leu, Ile, Met, Val, Ser, Pro, Thr, Ala, Tyr, His, Gln, Asn, Lys, Asp, Glu, Cys, Trp, Arg, Gly}.

a) Frequency appearance

$$d_{y\alpha} = \sum_{i=1}^{l} w_{y\alpha}(s_i) \quad (6)$$

$d_{y\alpha} \geq 0$ is total appearance of the y_α in the sequence S.

b) Correlation between position and distance

$$e_{y\alpha} = \frac{\sum_{i=1}^{l} (i * w_{y\alpha}(s_i)) * (i * w_{y\alpha}(s_i) - i_{pre} * w_{y\alpha}(s_{i\_pre}))}{d_{y\alpha}} \quad (7)$$

$e_{y\alpha} \geq 0$ is average of (position * distance), distance calculated as the difference between the current position and the previous position of y_α in the sequence S.

The pseudocode of Algorithm 4 explains how to transform an input subsequence into a 40-dimensional vector based on formulas (6) and (7).

Algorithm 4. CodonTranform(subsequence S)

INPUT: subsequence S
OUTPUT: a vector codon 6 dimensions
BEGIN
Initialize β = {Trp, Phe, Tyr, His, Asn, Asp, Cys, Gln, Lys, Glu, Ile, Val, Pro, Thr, Ala, Gly, Ser, Leu, Arg }.
Initialize vector vCodon 40 dimensions
FOR (i=0 to 19) **{**
vCodon [i] = Count frequency of element in β (by fomula(6))
}
FOR (i=20 to 39) **{**
vCodon [i] = Count frequency of element in β (by fomula(6))
}
RETURN vCodon
END

For example, with the given sequence S, will be calculate CodonTranform(S) = {0, 1, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 7, 0, 0, 6.5, 4.5, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0}.

B. The Combine Algorithm Transforms DNA Sequences into Vectors

In Part A, we have presented four algorithms that will be used in vector transformation, this section will present an algorithm that combines the above algorithms to generate the final feature vector. Algorithm to calculate similarity value will be performed on these feature vectors.

Definition 1. Given the sequence S = {s0, s1, s2, ..., sl-1} where l is length of S, we define the vector FCC(S) as follows:

$$FCC(S) = [\text{CombinatoricTranform(S)}, \text{CovarianceTranform(S)}, \text{CodonTranform(S)}] \quad (8)$$

The final feature vector *FCC(S)* is generated by merging the vectors produced by Algorithms 1, 2, and 4. Example, for sequence S = {AGTAGTGCTA}, we have *FCC(S) = { 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 1, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 7, 0, 0, 6.5, 4.5, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, -1.33, -2.33, -3, -1.6, -2.8, -1.24}*

Definition 2. Given the sequence S = {s0, s1, s2, ..., sl-1} where l is length of S, split S into Sa, Sb have equal length, we define the vector FCCW(S) as follows:

$$Va = [FCC(S_a)]$$

$$Vb = [FCC(S_b)]$$

$$FCCW(S) = [V_a + V_b, V_a - V_b] \quad (9)$$

In case where S has an odd number of characters then length $S_b = S_a+1$, that is the last odd character will be in the subsequence S_b. The vectors V_a, V_b can be calculated from the FCC formula in Definition 1. Applying the Haar wavelet filter according to Algorithm 3, concatenate 2 vectors V_a, V_b to generate an *FCCW* vector containing the features of the original sequence. In this vector, it will contain information about the frequency, the position and the correlation between position and distance of the features mentioned above, through

which the vector will be able to store the unique feature of the original sequence. In this way, an original subsequence of any length will be stored under a fixed length vector have 188 dimensions. Pseudocode for Algorithm 5 describes how to transform an input DNA sequence into a list of output final vectors.

Algorithm 5. TranformDNA(sequence S)

```
INPUT: DNA sequence S need tranform, window size W
OUTPUT: feature vectors
BEGIN
    //Calculate the number of windows will be cut from the Input string
    IF (Length(S) % W > 0){wCount = Length(S)/W +1}
    ELSE { wCount = Length(S)/W }
    //Partition sequence S into subsequences with window size
    FOR (i=0 to (wCount – 1)) {
        position = i*W
        listSub += Substring(S, position, W)
    }
    //Transform subsequences to vectors
    FOR (each subsequence in listSub) {
        Devide a subsequence into two areas Sa, Sb
        Calculation Va = [FCC(Sa)], Vb = [FCC(Sb)] (by fomula(8))
        vectors += WaveletHaarTranform(Va,Vb) (by fomula(9)) }
    RETURN listVectors
END
```

Algorithm 6. *FCCWD*(vector u, vector v)

```
INPUT: vector u, v
OUTPUT: similarity FCCWD between u and v
BEGIN
    posDis = 0; negDis = 0;
    FOR EACH (each vector u_i, v_i in u, v){
    IF(u_i > v_i) {
        posDis += u_i - v_i }
    ELSE {
        negDis += abs(u_i - v_i) } }
    m = min (posDis, negDis)
    μ = | posDis - negDis | / 2
    IF(m < μ) {
        RETURN μ/N }
    ELSE {
        RETURN (μ + (m - μ))/N }
END
```

The similarity value of two vectors is determined by the distance between the two vectors. To calculate this distance, the algorithm presented in [1] is employed. This algorithm calculates the distance between two vectors by finding the maximum number of operations required to transform from vector *u* to vector *v*. Algorithm 6 will calculate these values for each pair of *(u,v)* and storing the result in the variables posDis and negDis.

- Algorithmic complexity

The generated vectors will only contain numbers, which results in a computational complexity of O(*l*), much improved compared to O(l^2) of the *ED* algorithm.

IV. Experimental Results

In test model, we will take a text input DNA sequence and partition it into smaller subsequences using a selected window length. Next, we will use the *FCCWD* and *Hfwd*^{2} algorithms to transform these subsequences into a set of vectors, which will be stored in a general database. With these vectors, it will be possible to calculate the similarity values between them, and obtain the results. Lastly, we will compare the average results of the improved algorithm with the reference *ED* algorithm to evaluate the efficiency improvement achieved, as illustrated in Fig. 4.

We employed the use of Visual Studio 2019 and the C# programming language, as well as SQL Server 2019 for database management. The tests were conducted on a computer with the following configuration: CPU: Intel Xeon E2224G at 3.5 GHz, RAM: 16 GB, Hard Disk: Intel SSD 250 GB, Operating System: Windows Server 2019.

The test dataset is 171 genes of the family Poxviridae virus (32.9 MB), 172 genes of the family Asfarviridae virus (31.4 MB), 203 genes of the family Herpesviridae virus (32.6 MB), 23 genes of the family Corona virus (0.9 MB) available for download at the website https://www.ncbi.nlm.nih.gov/.

As the scenario has been built, we will compare the results of the similarity value calculation using three methods, with the standard *ED* method being the target to aim for. We will partition the data file into subsequences of different window lengths W = (1000, 2500, 5000, 7500, 10000, 15000, 20000) and evaluate the efficiency based on the average of the calculations. Since the parameters differ, the similarity values may not be on the same frame of reference. To compare the results, we will use the *k* nearest neighbor method on the resulting data. This means, given an input DNA sequence, we will find the cluster of *k* sequences with the closest similarity value for each method. The intersection of the resulting *FCCWD* clusters with *ED* and *Hfwd*^{2} with *ED* will represent the accuracy of the algorithms. As an example, if we search for similarities using the standard *ED* method, the result will be a set A with *k* elements having minimum values. The *FCCWD* method will find a set B also with *k* elements. The intersection of A and B (A ∩ B) will be the elements that *FCCWD* correctly found compared to *ED* as illustrated in Fig. 5.

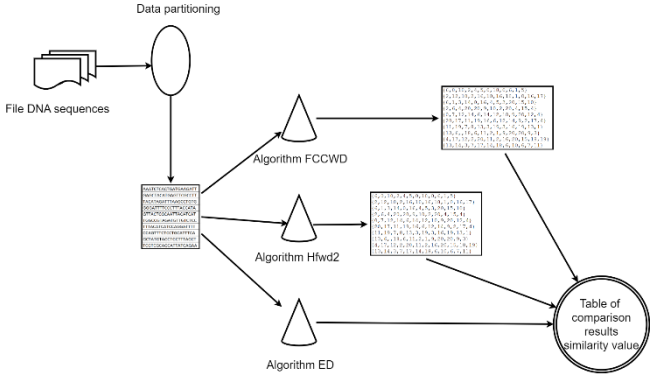

Fig. 4. Model of experiment.

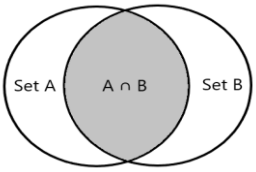

Fig. 5. Example of intersection of 2 result sets.

Fig. 6 and Fig. 7 illustrate the results for scenarios where k equals 5 and 10. The average true positive rate of *FCCWD* is higher than *Hfwd*^{2} under these scenarios. The efficiency of method is highest when the window length is between 5000 and 10000, but noticeably decreases when the window length is smaller than 5000 or larger than 10000.

Fig. 8 and Fig. 9 present the results for scenarios where k is equal to 15 and 20. Although performing worse than *Hfwd*^{2} at W = 1000 with k=20, but the average true positive rate of *FCCWD* still better than *Hfwd*^{2} in the remaining cases. The efficiency of method remains low for window lengths smaller than 5000, but proves to be stable when window lengths are larger than 5000.

The results indicate that when the window length is small, accuracy is lower and gradually improves as the window length increases. With small window lengths, the small number of characters creates high noise levels, leading to larger errors. When the sequence is longer, the accuracy of *FCCWD* is higher because we have added more parameters to store the eigenstate better than the *Hfwd*^{2} method. Therefore, this method is also more efficient when comparing large sequences than *Hfwd*^{2} tested on small windows from 100 to 1000. The best accuracy of the proposed method is the windows length from 5000 to 10000.

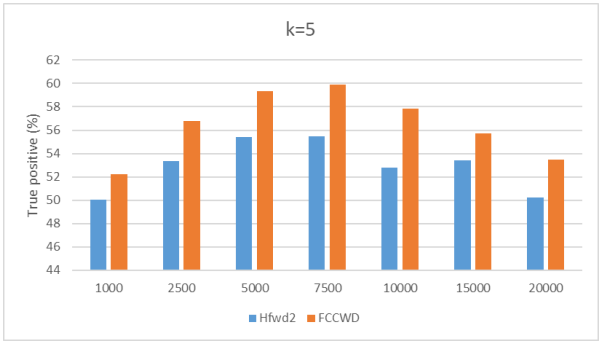

Fig. 6. Compare result with k = 5.

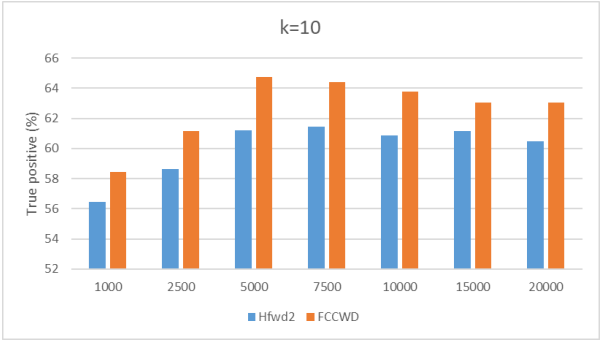

Fig. 7. Compare result with k = 10.

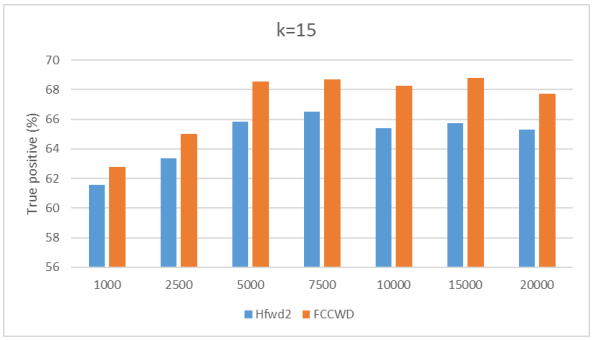

Fig. 8. Compare result with k = 15.

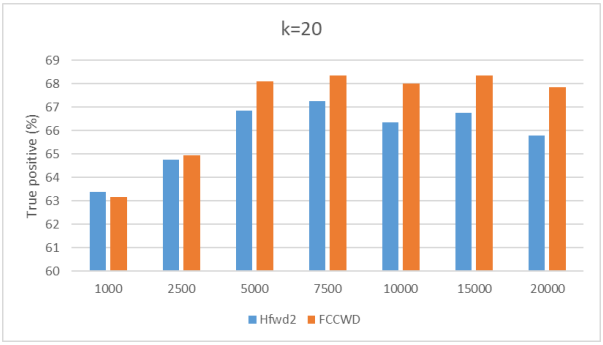

Fig. 9. Compare result with k = 20.

TABLE I. EXECUTION TIME OF METHODS (SECONDS)

W size / Method	1000	2500	5000	7500
ED	25.0029	155.7917	619.9818	1387.816
FCCWD	0.0003	0.0007	0.0014	0.0021
Hfwd^2	0.0003	0.0007	0.0013	0.002
W size / Method	**10000**	**15000**	**20000**	
ED	2457.2346	5496.9212	9730.9857	
FCCWD	0.0028	0.0042	0.0055	
Hfwd^2	0.0027	0.004	0.0053	

Regarding the computational performance shown in Table I and Fig. 10, the computational performance of both *FCCWD* and *Hfwd*^{2} methods is observed to be highly efficient, with computation times only being a few hundredths of a second. On the other hand, *ED* is significantly slower, ranging from a few tens to tens of thousands of seconds. Although *FCCWD* uses more parameters and has a larger vector dimension, the similarity algorithm has linear complexity, making the increase in cost minimal compared to *Hfwd*^{2}. As the window length of the data increases, the computation time of *FCCWD* increases very little, while *ED* increases very quickly.

The experiment also shows that the data compression ratio is quite good, with the larger window length the more compression, as shown in Fig. 11. Because how large a window is, it will also be transformed into a vector with 188-dimensions. This can help the data after transformation to be easily stored and reused in the next times.

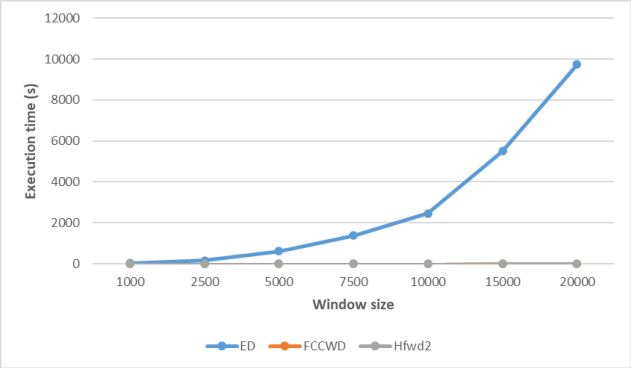

Fig. 10. Time comparison results.

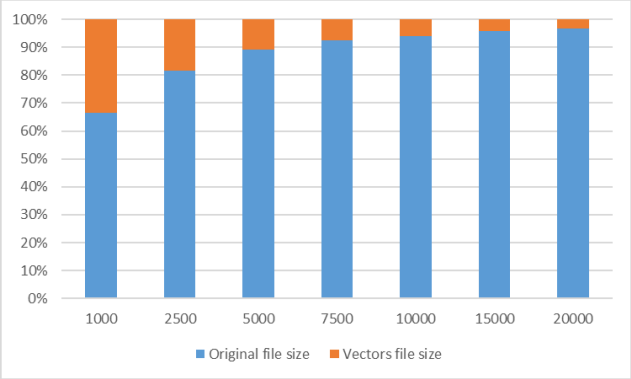

Fig. 11. Compare files size after transformation.

V. Conclusion

With the goal of improving the accuracy of similarity search, this paper propose an improved algorithm that transforms DNA sequences into numerical vectors using multiple feature parameters. These features are a blend of mathematical statistics and biological characteristics of genetic genes, allowing for a better representation of the original sequence information in the transformed vectors. Additionally, the method also enables efficient storage and reuse of the transformed data through reduction of its size. Experimental results demonstrate that the new algorithm improves similarity calculation accuracy while maintaining good performance. Although some cases of high noise still affect accuracy, the algorithm performs better in the case of long window length. In the future, we will continue to look for valuable parameters to further improve accuracy, as well as apply other indexing methods to process large data better.

Acknowledgment

This paper is supported by project CSCL02.06/22-23 of Institute of Information Technology (IoIT) - Vietnam Academy of Science and Technology (VAST), Hanoi, Vietnam.

References

[1] In-Seon Jeong, Kyoung-Wook Park, Seung-Ho Kang, Hyeong-Seok Lim, “An efficient similarity search based on indexing in large DNA databases,” Computational Biology and Chemistry, vol 34, pp.131-136, 2010.

[2] Zeyu Xia, Yingbo Cui, Ang Zhang, Tao Tang, Lin Peng, Chun Huang, Canqun Yang & Xiangke Liao, “A Review of Parallel Implementations for the Smith–Waterman Algorithm,” Interdisciplinary Sciences: Computational Life Sciences, vol 14, pp.1–14, 2022.

[3] Samer Mahmoud Wohoush, Mahmoud Hassan Saheb, “Indexing for Large DNA Database Sequenes,” International Journal of Biometrics and Bioinformatics (IJBB), vol 5, pp.202-215, 2011.

[4] T. Magoc and S. Salzberg, “FLASH: Fast length adjustment of short reads to improve genome assemblies,” Bioinformatics, pp.2957-2963, 2011.

[5] Xianyang Jiang, Peiheng Zhang, Xinchun Liu, Stephen S.-T.Yau, “Survey on index based homology search algorithms,” Springer Science + Business Media, LLC, pp.185-212, 2007.

[6] Hugh Williams, Justin Zobel, “Compression of nucleotide databases for fast searching,” Bioinformatics, pp.549–554,1997.

[7] Ooi BC, Pang HH, Wang H, Wong L, Yu C, “Fast filter-and-refine algorithms for subsequence selection,” Proceedings of the 6th international database engineering and applications symposium (IDEAS’02), Edmonton, Canada, pp.243–254, July 2002.

[8] Lee HP, Tsai YT, Sheu TF, Tang CT, “An IDC-based algorithm for efficient homology filtration with guaranteed seriate coverage,” Fourth IEEE symposium on bioinformatics and bioengineering (BIBE’04), Taichung, Taiwan, pp.395-402, 2004.

[9] Jim Dowling, KTH, “Reference Based Compression Algorithm”, Scalable, Secure Storage of Biobank Data, Work Package 2, pp.23 – 44, June 2014.

[10] Nahida Habib, Kawsar Ahmed , Iffat Jabin, Mohammad Motiur Rahman, “Modified HuffBit Compress Algorithm - An Application of R,” Journal of Integrative Bioinformatics, pp.1-13, Feb 2018 .

[11] Williams, Zobel, “Indexing and retrieval for genomic databases,” IEEE Transactions on Knowledge and Data Engineering Vol 14, pp.63–78, 2002.

[12] Ozturk, Ferhatosmanoglu, “Effective indexing and filtering for similarity search in large biosequence database,” Third IEEE Symposium on Bioinformatics and Bioengineering, Proceedings, pp.359–366, 2003.

[13] Kahveci, Singh, “An efficient index structure for string databases,” Proceedings of 27th International Conference on Very Large Data Base, Roma, Italy, pp.351–360, 2001.

[14] Yongkun Li, Lily He, Rong Lucy He & Stephen S.-T.Yau, “A novel fast vector method for genetic sequence comparison,” Scientific Reports, pp.1-11, 2017.

[15] Qing Zhou and Jun S. Liu, “Extracting sequence features to predict protein–DNA interactions: a comparative study,” Nucleic Acids Research, 36(12), pp.4137–4148, 2008.

[16] Stanley Maloy. Kelly Hughes, “Brenner's Encyclopedia of Genetics (Second Edition) ,” Elsevier, pp.242-245, 2013.

[17] Rui Dong, Lily He, Rong Lucy He, and Stephen S.-T. Yau1, “A Novel Approach to Clustering Genome Sequences Using Inter-nucleotide Covariance,” Frontiers in Genetics, pp.1-12, 2019.

[18] N. N. Kozlov, “The Study of the Secrets of the Genetic Code,” Journal of Computer and Communications, pp.64-83, 2018.