Hindawi Publishing Corporation
Computational and Mathematical Methods in Medicine
Volume 2013, Article ID 628036, 11 pages
http://dx.doi.org/10.1155/2013/628036

Research Article

Structural Complexity of DNA Sequence

Cheng-Yuan Liou, Shen-Han Tseng, Wei-Chen Cheng, and Huai-Ying Tsai

Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan

Correspondence should be addressed to Cheng-Yuan Liou; cyliou@csie.ntu.edu.tw

Received 10 January 2013; Accepted 3 March 2013

Academic Editor: Carlo Cattani

In modern bioinformatics, finding an efficient way to allocate sequence fragments with biological functions is an important issue. This paper presents a structural approach based on context-free grammars extracted from original DNA or protein sequences. This approach is radically different from all those statistical methods. Furthermore, this approach is compared with a topological entropy-based method for consistency and difference of the complexity results.

1. Introduction

DNA sequence analysis becomes important part in modern molecular biology. DNA sequence is composed of four nucleotide bases—adenine (abbreviated A), cytosine (C), guanine (G), and thymine (T) in any order. With four different nucleotides, 2 nucleotides could only code for maximum of 4^2 amino acids, but 3 nucleotides could code for a maximum 4^3 amino acids. George Gamow was the first person to postulate that every three bases can translate to a single amino acid, called a codon. Marshall Nirenberg and Heinrich J. Matthaei were the first to elucidate the nature of a genetic code. A short DNA sequence can contain less genetic information, while lots of bases may contain much more genetic information, and any two nucleotides switch place may change the meaning of genetic messages.

Sequence arrangement can produce many different results, but only few codons exist in living bodies. Some sequences do not contain any information which is known as junk DNA. Finding an efficient way to analyze a sequence fragment corresponding to genetic functions is also a challenging problem.

In recent papers, methods broadly fall into two categories, sequence complexity [1, 2] and structural pattern analysis [3–8]. Koslicki [1] presented a method for computing sequence complexities. He redefined topological entropy function so that the complexity value will not converge toward zero for much longer sequences. With separate sequence into several segments, it can determine the segments where are exons or introns, and meaningful or meaningless. Hao et al. [7] given a graphical representation of DNA sequence, according to this paper, we can find some rare occurred subsequences. R. Zhang and C. T. Zhang [4] used four-nucleotide-related function drawing 3D curves graph to analyze the number of four-nucleotide occurrence probabilities. Liou et al. [9] had given a new idea in modeling complexity for music rhythms; this paper translated text messages into computable values, so computers can score for music rhythms.

In this paper, we propose a new method for calculating sequences different from other traditional methods. It holds not only statistical values but also structural information. We replace four nucleotides with tree structure presented in [9] and use mathematical tools to calculate complexity values of the sequences. So we can compare two sequences with values and determine dissimilarity between these two sequences. In biomedical section, we can use this technique to find the effective drugs for new virus with priority.

2. DNA Sequence Represented with Tree Structure

Our method uses Lindenmayer system [10–12] property among calculated complexities from tree structure [9]; it is a different way of computing complexities of sequences. At first, we introduce *DNA tree* and convert DNA sequence to

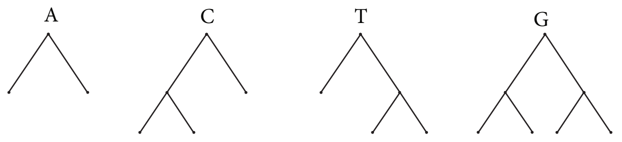


Figure 1: Nucleotide bases corresponding trees.

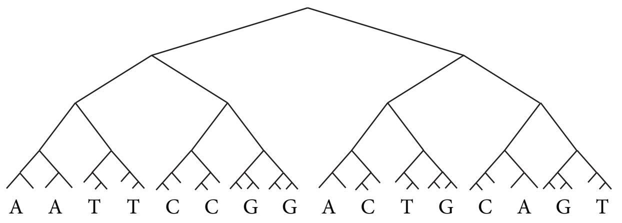


Figure 2: DNA sequence represented with tree structure.

tree structure. A DNA tree is a binary tree of which each subtree is also a DNA tree. Every tree node is either a terminal node or a node with two childrens (branches or descendants).

Lindenmayer system is a powerful rewriting system used to model the growth processes of plant development. We will introduce it in Section 2.2 in detail. Lindenmayer system uses some initial and rewriting rules to construct beautiful graphs. Since it can construct a tree from rewriting rules, it also can extract rewriting rules from a tree. In this section, we will use tools to generate the rules from tree.

We use 4 fixed *tree representations* for nucleotide bases A, T, C, and G (see Figure 1). When we apply this method to amino acid sequence, we can construct more tree representation for amino acids, respectively.

When we transfer a sequence to DNA tree, we will replace every word to tree elements step by step, and two consecutive trees can combine to a bigger tree. Following the previous steps, a DNA sequence will be transfer to a DNA tree (see Figure 2).

2.1. Bracketed Strings for a DNA Sequence. For computing complexity of our DNA tree, we need some rules for converting tree to another structure. We use a stack similarly structure to represent the hierarchy of DNA tree, called *bracketed string*. DNA tree can transfer to a unique bracketed string by the following symbols, and it can transfer back to the original tree:

(i) F: the current location of tree nodes; it can be replaced by any word or be omitted;

(ii) +: the following string will express the right subtree;

(iii) −: the following string will express the left subtree;

(iv) [: this symbol is pairing with]; "[⋯]" denotes a subtree where "⋯"; indicates all the bracketed strings of its subtree;

(v)]: see [description.

Following the previous symbols, Figure 3 shows that nucleotide base A and T represented tree can transfer to $[F[-F][+F]]$ and $[F[-F][+F[-F][+F]]]$, respectively.

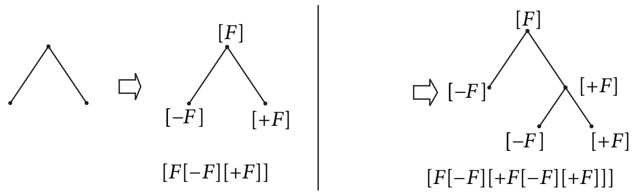


Figure 3: Bracketed strings representation for two trees.

And Figure 4 is the bracketed string of Figure 2. We can see that when the tree grows, string seems to be more redundant. Since we focus here only on DNA trees, we can simplify the bracketed string representations. First, our trees have only two subtrees. Second, the "F" notation for the tree is trivial. With these two characteristics, we may omit the "F" notation from the bracketed string and use only four symbols, {[,], −, +}, to represent trees. In our cases, "[⋯]" denotes a subtree where "⋯" indicates all the bracketed strings of its subtrees. "−" indicated the next "[⋯]" notation for a tree is a left subtree of current node, and "+" is a right subtree vice versa. Figure 5 is the simplified string of bracketed string shown in Figure 4.

2.2. DNA Sequence Represented with L-System. When we obtain DNA tree and bracketed string representation, we need rewriting rules for analyzing tree structure. There are some types of rewriting mechanism such as Chomsky grammar and Lindenmayer system (*L-system* for short). The largest difference between two string rewriting mechanisms lies in the technique used to apply productions. Chomsky grammar is suitable for applying productions sequentially, while L-system is for parallel. In our structure, applying L-system to our representations is better than Chomsky grammar.

The L-system was introduced by the biologist Lindenmayer in 1968 [13]. The central concept of the L-system is rewriting. In general, rewriting is a technique used to define complex objects by successively replacing parts of a simple initial object, using a set of rewriting rules or productions. In the next section, we will present how we use L-system to our DNA tree. The L-system is defined as follows.

Definition 1. L-system grammars are very similar to the Chomsky grammar, defined as a tuple [14]:

$$G = (V, \omega, P), \tag{1}$$

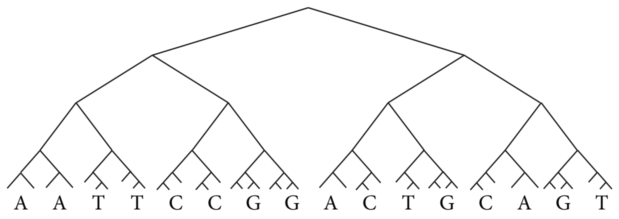
where

(i) $V = \{s_1, s_2, \ldots, s_n\}$ is an alphabet,

(ii) ω (start, axiom, or initiator) is a string of symbols from V defining the initial state of the system,

(iii) P is defined by a production map $P : V \rightarrow V^*$ with $s \rightarrow P(s)$ for each s in V. The identity production $s \rightarrow s$ is assumed. These symbols are called constants or terminals.

2.3. Rewriting Rules for DNA Sequences. As discussed earlier, we want to generate the rules from DNA trees. In this section,



$[[-F[-F[-F[-F[-F][+F]][+F[-F][+F]]][+F[-F[-F][+F[-F][+F]]][+F[-F][+F[-F][+F]]]]][+F[-F[-F[-F[-F][+F]]$

$[+F]][+F[-F[-F][+F]][+F]]][+F[-F[-F[-F][+F]][+F[-F][+F]]][+F[-F[-F][+F]][+F[-F][+F]]]]]][+F[-F[-F[-F$

$[-F][+F]][+F[-F[-F][+F]][+F]]][+F[-F[-F][+F[-F][+F]]][+F[-F[-F][+F]][+F[-F][+F]]]]][+F[-F[-F[-F[-F]$

$[+F]][+F]][+F[-F][+F]]][+F[-F[-F[-F][+F]][+F[-F][+F]]][+F[-F][+F[-F][+F]]]]]]]$

Figure 4: Bracketed strings representation for Figure 2.

$[-[-[-[-[-+]+[-+]]+[-[-+[-+]]+[-+[-+]]]]+[-[-[-[-+]+]+[-[-+]+]]+[-[-[-+]+[-+]]+[-[-+]+[-+]]]]]$

$+[-[-[-[-+]+[-[-+]+]]+[-[-+[-+]]+[-[-+]+[-+]]]]+[-[-[-[-+]+]+[-+]]+[-[-[-+]+[-+]]+[-+[-+]]]]]]$

Figure 5: More simply bracketed strings representation for Figure 2.

we will explain how we apply rewriting rules to those trees. We can apply distinct variables to each node. Since the technique described previously always generates two subtrees for each node, for every nonterminal node, they always can be explained in the following format:

$$P \longrightarrow LR, \tag{2}$$

where P denotes the current node, L denotes its left subtree, and R denotes its right subtree, respectively. We give an example shown in Figure 6; left tree has three nodes and only root is nonterminal node, it can be rewritten as $P \rightarrow LR$. Right tree has five nodes, root P with left subtree L and right subtree R. Left subtree is terminal, but right is not. R has two terminal subtrees R_L and R_R, so this tree can be rewritten as $P \rightarrow LR$ and $R \rightarrow R_L R_R$.

2.4. Rewriting Rules for Bracketed Strings. Similarly, we can also use rewriting rules to generate bracketed strings. In rewriting rules for DNA trees shown in Section 2.3, we write $P \rightarrow LR$ for a tree with left and right subtrees. Note that we call L and R as the nonterminals. In this section, terminal nodes will be separated from trees, and we use "null" to represent a terminal. Such tree will have a corresponding bracketed string as follows: $[[-F\cdots][+F\cdots]]$. "$[-F\cdots]$" represents the left subtree, while "$[+F\cdots]$" represents the right subtree. Therefore, we can replace the rewriting rules with

$$\begin{aligned} P &\longrightarrow [-FL]\,[+FR]\,, \\ F &\longrightarrow \cdots, \\ R &\longrightarrow \cdots, \end{aligned} \tag{3}$$

where "$\cdots$" is the rewriting rule for the bracketed string of each subtree. For the sake of readability, we replace the words such as "R_{R_L}" and "R_{R_R}". In Figure 7, we show the rewriting rules for the bracketed string of the tree in Figure 3.

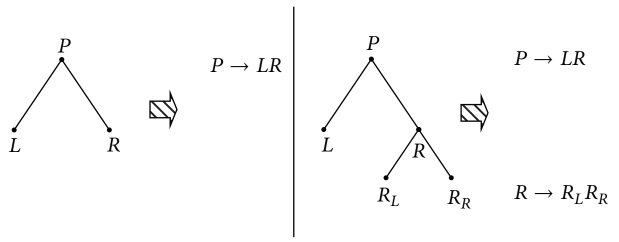


Figure 6: Example of rewriting rules for trees.

As we can see, there are "nulls" in the rules. Those "nulls" do not have significant effects to our algorithm, so we simply ignore the nulls. Now, Figure 3 can apply new rewriting rules without trivial nulls as Figure 8.

When tree grows up, the rewriting rules may generate identical rules. Assume that we have the following rules:

$$\begin{aligned} P &\longrightarrow [-FT_L]\,[+FT_R]\,, \\ T_L &\longrightarrow [-F]\,[+F]\,, \\ T_R &\longrightarrow [-F]\left[+FT_{R_R}\right], \\ T_{R_R} &\longrightarrow [-F]\left[+FT_{R_{R_R}}\right], \\ T_{R_{R_R}} &\longrightarrow [-F]\,. \end{aligned} \tag{4}$$

These rules can generate exactly one bracketed string and, thus, exactly one DNA tree. All these rules form a rule set that represents a unique DNA tree. When we look at $T_R \rightarrow [-F][+FT_{R_R}]$ and $T_{R_R} \rightarrow [-F][+FT_{R_{R_R}}]$, they have the same structure since they both have a right subtree and do not have a left subtree. The only difference is that one of the subtrees is T_{R_R} and that the other is $T_{R_{R_R}}$. We will define two terms to

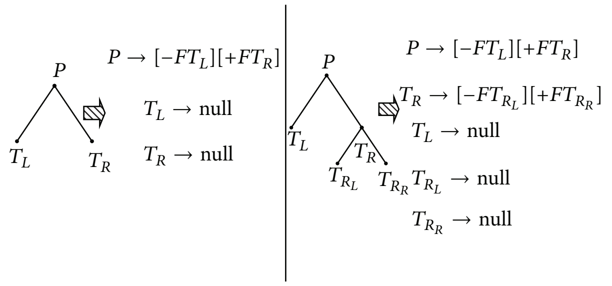


FIGURE 7: Rewriting rules for the bracketed string of trees.

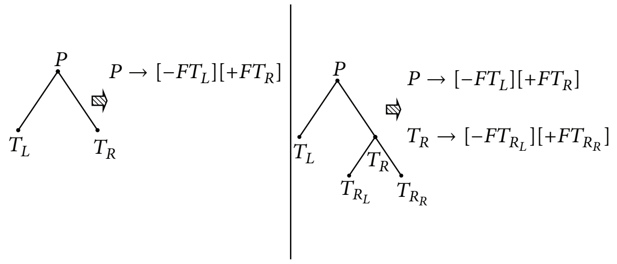


FIGURE 8: Rewriting rules for the bracketed string without nulls of trees.

express the similarity between two rewriting rules, and these terms can simplify complexity analysis.

2.5. Homomorphism and Isomorphism of Rewriting Rules. At the end of the previous section, we discussed that $T_R \to [-F][+FT_{R_R}]$ and $T_{R_R} \to [-F][+FT_{R_{R_R}}]$ are almost the same. How can we summarize or organize an effective feature to them? Liou et al. [9] gave two definitions to classify similar rewriting rules described before as follows.

Definition 2. Homomorphism in rewriting rules. We define that rewriting rule R_1 and rewriting rule R_2 are homomorphic to each other if and only if they have the same structure.

In detail, rewriting rule R_1 and rewriting rule R_2 in DNA trees both have subtrees in corresponding positions or both not. Ignoring all nonterminals, if rule R_1 and rule R_2 generate the same bracketed string, then they are homomorphic by definition.

Definition 3. Isomorphism on level X in rewriting rules. Rewriting rule R_1 and rewriting rule R_2 are isomorphic on depth X if they are homomorphic and their nonterminals are relatively isomorphic on depth $X-1$. Isomorphic on level 0 indicates homomorphism.

Applying to the bracketed string, we ignore all nonterminals in (4) as follows:

$$P \longrightarrow [-FT_L]\,[+FT_R] \longrightarrow [-F]\,[+F],$$
$$T_L \longrightarrow [-F]\,[+F] \longrightarrow [-F]\,[+F],$$
$$T_R \longrightarrow [-F]\left[+FT_{R_R}\right] \longrightarrow [-F]\,[+F],$$
$$T_{R_R} \longrightarrow [-F]\left[+FT_{R_{R_R}}\right] \longrightarrow [-F]\,[+F],$$
$$T_{R_{R_R}} \longrightarrow [-F] \longrightarrow [-F]. \tag{5}$$

We find that P, T_L, T_R, and T_{R_R} are homomorphic to each other; they generate the same bracketed string, $[-F][+F]$. But $T_{R_{R_R}}$ is not homomorphic to any of the other rules; its bracketed string is $[-F]$.

Let us recall DNA tree example in Figure 2; we will use this figure as an example to clarify these definitions. Now we marked some nodes shown in Figure 9; there are tree rooted at A, B, C, and D, respectively, tree A, tree B, tree C, and tree D. Tree A is isomorphic to tree C on depth 0 to 3, but they are not isomorphic on depth 4. Tree B is isomorphic to tree C on depth from 0 to 2, but they are not isomorphic on depth 3. D is not isomorphic to any other trees, nor is it homomorphic to any other trees.

After we define the similarity between rules by homomorphism and isomorphism, we can classify all the rules into different subsets, and every subset has the same similarity relation. Now we list all the rewriting rules of Figure 2 into Table 1 but ignore terminal rules such as "$\to$ null" and transfer rule's name to class name (or class number). For example, we can give terminal rewriting rule a class, "$C_3 \to$ null", and a rule link to two terminals; we can give them "$C_2 \to C_3C_3$"; here C_3 is the terminal class. After performing classification, we obtain not only a new rewriting rule set but also a context-free grammar, which can be converted to automata.

In Table 1, rules such as $T_{R_{L_{L_L}}} \to [-F][+F]$, and $T_{R_{R_{R_{L_L}}}} \to [-F][+F]$ and $T_{R_{L_{R_{L_R}}}} \to [-F][+F]$ are isomorphic on depth 1 and assigned to Class 4. There are twenty such rules before classification, so we write "$(20)C_4 \to [-F][+F]$". Similar rules such as $P \to [-FT_L][+FT_R]$, $T_{R_{L_{L_L}}} \to [-F][+F]$, and $T_{R_{R_{R_R}}} \to [-F][+FT_{R_{R_{R_{R_R}}}}]$ are isomorphic on depth 0, and there are 47 such rules. They are all assigned to Class 1 by following a similar classification procedure. The classification of the all rules is listed in Table 2. Note that this section also presents a new way to convert a context-sensitive grammar to a context-free one.

3. DNA Sequence Complexity

When we transfer the DNA sequence to the rewriting rules, and classify all those rules we attempt to explore the redundancy in the tree that will be the base for building the cognitive map [15]. We compute the complexity of the tree which those classified rules represent. We know that a classified rewriting rule set is also a context-free grammar, so there are some methods for computing complexity of rewriting rule as follows.

Definition 4. Topological entropy of a context-free grammar. The topological entropy K_0 of (context-free grammar) CFG can be evaluated by means of the following three procedures [16, 17].

Table 1: Rewriting rules for the DNA tree in Figure 2.

$P \to [-FT_L]\,[+FT_R]$
$T_L \to [-FT_{L_L}]\,[+FT_{L_R}]$
$T_{L_L} \to [-FT_{L_{L_L}}]\,[+FT_{L_{L_R}}]$
$T_{L_{L_L}} \to [-FT_{L_{L_{L_L}}}]\,[+FT_{L_{L_{L_R}}}]$
$T_{L_{L_{L_L}}} \to [-F]\,[+F]$
$T_{L_{L_{L_R}}} \to [-F]\,[+F]$
$T_{L_{L_R}} \to [-FT_{L_{L_{R_L}}}]\,[+FT_{L_{L_{R_R}}}]$
$T_{L_{L_{R_L}}} \to [-F]\,[+FT_{L_{L_{R_{L_R}}}}]$
$T_{L_{L_{R_{L_R}}}} \to [-F]\,[+F]$
$T_{L_{L_{R_R}}} \to [-F]\,[+FT_{L_{L_{R_{R_R}}}}]$
$T_{L_{L_{R_{R_R}}}} \to [-F]\,[+F]$
$T_{L_R} \to [-FT_{L_{R_L}}]\,[+FT_{L_{R_R}}]$
$T_{L_{R_L}} \to [-FT_{L_{R_{L_L}}}]\,[+FT_{L_{R_{L_R}}}]$
$T_{L_{R_{L_L}}} \to [-FT_{L_{R_{L_{L_L}}}}]\,[+F]$
$T_{L_{R_{L_{L_L}}}} \to [-F]\,[+F]$
$T_{L_{R_{L_R}}} \to [-FT_{L_{R_{L_{R_L}}}}]\,[+F]$
$T_{L_{R_{L_{R_L}}}} \to [-F]\,[+F]$
$T_{L_{R_R}} \to [-FT_{L_{R_{R_L}}}]\,[+FT_{L_{R_{R_R}}}]$
$T_{L_{R_{R_L}}} \to [-FT_{L_{R_{R_{L_L}}}}]\,[+FT_{L_{R_{R_{L_R}}}}]$
$T_{L_{R_{R_{L_L}}}} \to [-F]\,[+F]$
$T_{L_{R_{R_{L_R}}}} \to [-F]\,[+F]$
$T_{L_{R_{R_R}}} \to [-FT_{L_{R_{R_{R_L}}}}]\,[+FT_{L_{R_{R_{R_R}}}}]$
$T_{L_{R_{R_{R_L}}}} \to [-F]\,[+F]$
$T_{L_{R_{R_{R_R}}}} \to [-F]\,[+F]$
$T_R \to [-FT_{R_L}]\,[+FT_{R_R}]$
$T_{R_L} \to [-FT_{R_{L_L}}]\,[+FT_{R_{L_R}}]$
$T_{R_{L_L}} \to [-FT_{R_{L_{L_L}}}]\,[+FT_{R_{L_{L_R}}}]$
$T_{R_{L_{L_L}}} \to [-F]\,[+F]$
$T_{R_{L_{L_R}}} \to [-FT_{R_{L_{L_{R_L}}}}]\,[+F]$
$T_{R_{L_{L_{R_L}}}} \to [-F][+F]$
$T_{R_{L_R}} \to [-FT_{R_{L_{R_L}}}]\,[+FT_{R_{L_{R_R}}}]$
$T_{R_{L_{R_L}}} \to [-F]\,[+FT_{R_{L_{R_{L_R}}}}]$
$T_{R_{L_{R_{L_R}}}} \to [-F][+F]$
$T_{R_{L_{R_R}}} \to [-FT_{R_{L_{R_{R_L}}}}]\,[+FT_{R_{L_{R_{R_R}}}}]$
$T_{R_{L_{R_{R_L}}}} \to [-F][+F]$
$T_{R_{L_{R_{R_R}}}} \to [-F]\,[+F]$
$T_{R_R} \to [-FT_{R_{R_L}}]\,[+FT_{R_{R_R}}]$
$T_{R_{R_L}} \to [-FT_{R_{R_{L_L}}}]\,[+FT_{R_{R_{L_R}}}]$
$T_{R_{R_{L_L}}} \to [-FT_{R_{R_{L_{L_L}}}}]\,[+F]$
$T_{R_{R_{L_{L_L}}}} \to [-F]\,[+F]$
$T_{R_{R_{L_R}}} \to [-F][+F]$
$T_{R_{R_R}} \to [-FT_{R_{R_{R_L}}}]\,[+FT_{R_{R_{R_R}}}]$
$T_{R_{R_{R_L}}} \to [-FT_{R_{R_{R_{L_L}}}}]\,[+FT_{R_{R_{R_{L_R}}}}]$
$T_{R_{R_{R_{L_L}}}} \to [-F][+F]$
$T_{R_{R_{R_{L_R}}}} \to [-F][+F]$
$T_{R_{R_{R_R}}} \to [-F]\,[+FT_{R_{R_{R_{R_R}}}}]$
$T_{R_{R_{R_{R_R}}}} \to [-F][+F]$

Table 1: Continued.

(1) For each variable V_i with productions (in Greibach form),

$$V_i \longrightarrow t_{i_1}U_{i_1}, t_{i_2}U_{i_2}, \ldots, t_{i_{k_i}}U_{i_{k_i}}, \tag{6}$$

where $\{t_{i_1}, t_{i_2}, \ldots, t_{i_{k_i}},\}$ are terminals and $\{U_{i_1}, U_{i_2}, \ldots, U_{i_{k_i}},\}$ are nonterminals. The formal algebraic expression for each variable is

$$V_i = \sum_{j=1}^{k_i} t_{i_j}U_{i_j}. \tag{7}$$

(2) By replacing every terminal t_{i_j} with an auxiliary variable z, one obtains the generating function

$$V_i(z) = \sum_{n=1}^{\infty} N_i(n)\, z^n, \tag{8}$$

where $N_i(n)$ is the number of words of length n descending from V_i.

(3) Let $N(n)$ be the largest one of $N_i(n)$, $N(n) = \max\{N_i(n)$, for all $i\}$. The previous series converges

TABLE 2: Classification based on the similarity of rewriting rules.

Classification of rules	Isomorphic Depth #0	Isomorphic Depth #1	Isomorphic Depth #2	Isomorphic Depth #3
Class #1	(19) $C_1 \rightarrow C_1C_1$ (4) $C_1 \rightarrow C_1C_2$ (4) $C_1 \rightarrow C_2C_1$ (20) $C_1 \rightarrow C_2C_2$	(8) $C_1 \rightarrow C_1C_1$ (1) $C_1 \rightarrow C_1C_3$ (1) $C_1 \rightarrow C_2C_2$ (1) $C_1 \rightarrow C_2C_4$ (1) $C_1 \rightarrow C_3C_1$ (1) $C_1 \rightarrow C_3C_3$ (1) $C_1 \rightarrow C_4C_2$ (5) $C_1 \rightarrow C_4C_4$	(3) $C_1 \rightarrow C_1C_1$ (1) $C_1 \rightarrow C_4C_2$ (1) $C_1 \rightarrow C_7C_5$ (1) $C_1 \rightarrow C_8C_8$ (1) $C_1 \rightarrow C_3C_1$ (1) $C_1 \rightarrow C_8C_6$	(1) $C_1 \rightarrow C_1C_1$ (1) $C_1 \rightarrow C_4C_3$ (1) $C_1 \rightarrow C_5C_2$
Class #2	(48) $C_2 \rightarrow$ null	(4) $C_2 \rightarrow C_4C_5$	(1) $C_2 \rightarrow C_8C_{10}$	(1) $C_2 \rightarrow C_8C_6$
Class #3		(4) $C_3 \rightarrow C_5C_4$	(1) $C_3 \rightarrow C_9C_9$	(1) $C_3 \rightarrow C_9C_7$
Class #4		(20) $C_4 \rightarrow C_5C_5$	(1) $C_4 \rightarrow C_9C_{11}$	(1) $C_4 \rightarrow C_{12}C_{10}$
Class #5		(48) $C_5 \rightarrow$ null	(1) $C_5 \rightarrow C_{10}C_8$	(1) $C_5 \rightarrow C_{13}C_{11}$
Class #6			(1) $C_6 \rightarrow C_{10}C_{10}$	(1) $C_6 \rightarrow C_{13}C_{13}$
Class #7			(1) $C_7 \rightarrow C_{11}C_9$	(1) $C_7 \rightarrow C_{13}C_{15}$
Class #8			(5) $C_8 \rightarrow C_{11}C_{11}$	(1) $C_8 \rightarrow C_{14}C_{14}$
Class #9			(4) $C_9 \rightarrow C_{11}C_{12}$	(1) $C_9 \rightarrow C_{14}C_{16}$
Class #10			(4) $C_{10} \rightarrow C_{12}C_{11}$	(1) $C_{10} \rightarrow C_{15}C_{13}$
Class #11			(20) $C_{11} \rightarrow C_{12}C_{12}$	(1) $C_{11} \rightarrow C_{15}C_{15}$
Class #12			(48) $C_{12} \rightarrow$ null	(1) $C_{12} \rightarrow C_{16}C_{14}$
Class #13				(5) $C_{13} \rightarrow C_{16}C_{16}$
Class #14				(4) $C_{14} \rightarrow C_{16}C_{17}$
Class #15				(4) $C_{15} \rightarrow C_{17}C_{16}$
Class #16				(20) $C_{16} \rightarrow C_{17}C_{17}$
Class #17				(48) $C_{17} \rightarrow$ null

when $z < R = e^{-K_0}$. The topological entropy is given by the radius of convergence R as

$$K_0 = -\ln R. \tag{9}$$

Our productions have some difference from the aforementioned definitions. First, our productions are written in Chomsky-reduced form instead of Greibach form. Second, DNA is finite sequence; it generates finite tree, but the previous formulas are applied on infinite sequences. For convenience in the DNA tree case, we rewrite the definition as follows [9].

Definition 5. Topological entropy of context free grammar for DNA tree.

(1) Assume that there are n classes of rules and that each class C_i contains n_i rules. Let $V_i \in \{C_1, C_2, \ldots, C_n\}$, $U_{ij} \in \{R_{ij}, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n_i\}$, and $a_{ijk} \in \{x : x = 1, 2, \ldots, n\}$, where each U_{ij} has the following form:

$$\begin{aligned} U_{i1} &\longrightarrow V_{a_{i11}} V_{a_{i12}}, \\ U_{i2} &\longrightarrow V_{a_{i21}} V_{a_{i22}}, \\ \cdots &\longrightarrow \cdots, \\ U_{in_i} &\longrightarrow V_{a_{in_i1}} V_{a_{in_i2}}. \end{aligned} \tag{10}$$

(2) The generating function of V_i, $V_i(z)$ has a new form as follows:

$$V_i(z) = \frac{\sum_{p=1}^{n_i} n_{ip} z V_{a_{ip1}}(z) V_{a_{ip2}}(z)}{\sum_{q=1}^{n_i} n_{iq}}. \tag{11}$$

If V_i does not have any nonterminal variables, we set $V_i(z) = 1$.

(3) After formulating the generating function $V_i(z)$, we intend to find the largest value of $z, z^{\max}$, at which $V_1(z^{\max})$ converges. Note that we use V_1 to denote the

rule for the root node of the DNA tree. After obtaining the largest value, $z^{\max}$, of $V_1(z)$, we set $R = z^{\max}$, the radius of convergence of $V_1(z)$. We define the complexity of the DNA tree as

$$K_0 = -\ln R. \tag{12}$$

Now we can do some examples of computation procedure for the complexity. According to our definition, the given values for the class parameters are listed in Table 3. There are five classes, so we obtain the formulas for $V_5(z'), V_4(z'), V_3(z'), V_2(z')$, and $V_1(z')$ successively. They are

$$\begin{aligned}
V_5(z') &= 1 \,(\text{by definition}),\\
V_4(z') &= \frac{\sum_{p=1}^{n_4} n_{4p} z' V_{a_{4p1}}(z') V_{a_{4p2}}(z')}{\sum_{q=1}^{n_i} n_{iq}}\\
&= \frac{z' \times (20 \times V_5(z') \times V_5(z'))}{20} = z',\\
V_3(z') &= \frac{\sum_{p=1}^{n_3} n_{3p} z' V_{a_{3p1}}(z') V_{a_{3p2}}(z')}{\sum_{q=1}^{n_i} n_{iq}}\\
&= \frac{z' \times (4 \times V_5(z') \times V_4(z'))}{4} = z'^2,\\
V_2(z') &= \frac{\sum_{p=1}^{n_2} n_{2p} z' V_{a_{2p1}}(z') V_{a_{2p2}}(z')}{\sum_{q=1}^{n_i} n_{iq}}\\
&= \frac{z' \times (4 \times V_4(z') \times V_5(z'))}{4} = z'^2,\\
V_1(z') &= \frac{\sum_{p=1}^{n_1} n_{1p} z' V_{a_{1p1}}(z') V_{a_{1p2}}(z')}{\sum_{q=1}^{n_i} n_{iq}}\\
&= \frac{8z' \times V_1(z')^2 + 2(z')^3 \times V_1(z')}{19}\\
&\quad + \frac{(2(z')^5 + 2(z')^4 + 5(z')^3)}{19}.
\end{aligned} \tag{13}$$

Rearranging the previous equation for $V_1(z')$, we obtain a quadratic for $V_1(z')$:

$$\begin{aligned}
&\frac{8}{19}(z') \times V_1(z') + \left(1 - \frac{2}{19}(z')^3\right) \times V_1(z')\\
&\quad + \frac{1}{19}\left(2(z')^5 + 2(z')^4 + 5(z')^3\right) = 0.
\end{aligned} \tag{14}$$

Solving $V_1(z')$, we obtain the formula

$$V_1(z') = \left(\frac{(z')^2}{4} - \frac{19}{8z'}\right) \pm \frac{19}{8z'}\sqrt{B^2 - A}, \tag{15}$$

where

$$\begin{aligned}
A &= \frac{32}{361}\left(2(z')^6 + 2(z')^5 + 5(z')^4\right),\\
B &= 1 - \frac{2}{19}(z')^3.
\end{aligned} \tag{16}$$

TABLE 3: The values for the class parameters of Table 2.

	Classification of rules	Isomorphic depth #1
($n = 5$)	Class #1 ($n_1 = 8$)	n_{11} $n_{111}n_{112}$ (8) $C_1 \to C_1C_1$
		n_{12} $n_{121}n_{122}$ (1) $C_1 \to C_1C_3$
		n_{13} $n_{131}n_{132}$ (1) $C_1 \to C_2C_2$
		n_{14} $n_{141}n_{142}$ (1) $C_1 \to C_2C_4$
		n_{15} $n_{151}n_{152}$ (1) $C_1 \to C_3C_1$
		n_{16} $n_{161}n_{162}$ (4) $C_1 \to C_3C_3$
		n_{17} $n_{171}n_{172}$ (1) $C_1 \to C_4C_2$
		n_{18} $n_{181}n_{182}$ (5) $C_1 \to C_4C_4$
	Class #2 ($n_2 = 1$)	n_{21} $n_{211}n_{212}$ (4) $C_2 \to C_4C_5$
	Class #3 ($n_3 = 1$)	n_{31} $n_{311}n_{312}$ (4) $C_3 \to C_5C_4$
	Class #4 ($n_4 = 1$)	n_{41} $n_{411}n_{412}$ (20) $C_4 \to C_5C_5$
	Class #5 ($n_5 = 1$)	n_{51} $n_{511}n_{512}$ (48) $C_5 \to$ null

TABLE 4: Test data with topological entropy method and our method.

Type	Name	Koslicki method	Our method
DNA	E. coli[a]	Available	Available
	EV71[b]	Available	Available
	H1N1[c]	Available	Available
	H5N1[d]	Available	Available
	SARS[e]	Available	Available
Amino acid	Abrin	Too short	Available
	Ricin	Too short	Available
	BSE[f]	Too short	Available
	CJD[g]	Too short	Available

[a] *Escherichia coli* O157:H7.
[b] Enterovirus 71.
[c] Influenza A virus subtype H1N1.
[d] Influenza A virus subtype H5N1.
[e] Severe acute respiratory syndrome.
[f] Bovine spongiform encephalopathy.
[g] Creutzfeldt-Jakob disease.

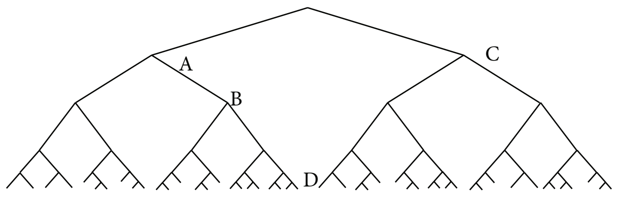


Figure 9: Example of homomorphism and isomorphism.

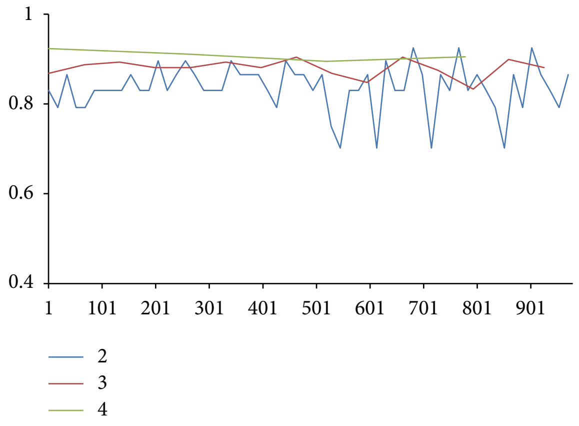


Figure 10: Koslicki method (topological entropy method, TE for short) example.

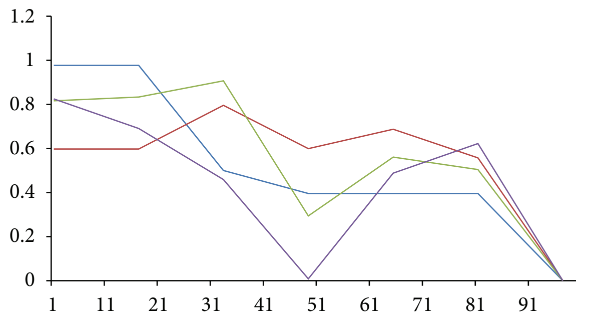
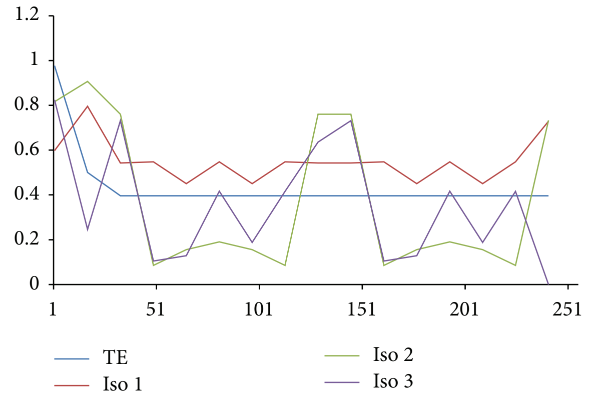



Figure 11: Our method compared with TE using test sequences.

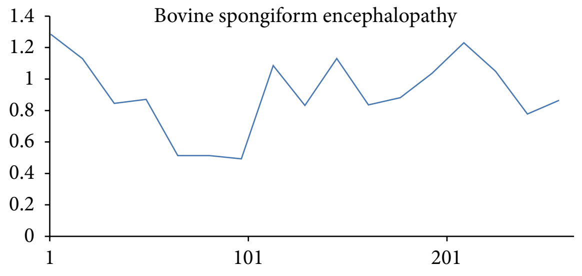


Figure 12: An amino acid sequence example, Bovine spongiform encephalopathy.

Finally, the radius of convergence, R, and complexity, $K_0 = -\ln R$, can be obtained from this formula. But, computing the $z^{\max}$ directly is difficult, so we use iterations and region tests to approximate the complexity; details are as follows.

(1) Rewrite the generating function as

$$V_i^m(z') = \frac{\sum_{p=1}^{n_i} n_{ip} z' V_{a_{ip1}}^{m-1}(z') V_{a_{ip2}}^{m-1}(z')}{\sum_{q=1}^{n_i} n_{iq}}, \quad (17)$$
$$V_i^0(z') = 1.$$

(2) The value from $V_i^0(z')$ to $V_i^m(z')$. When $V_i^{m-1}(z') = V_i^m(z')$ for all rules, we say that $V_i^m(z')$ reach the convergence, but z' is not the $z^{\max}$ we want. Here, we set $m = 1000$ for each iteration.

(3) Now we can test whether $V_i(z')$ is convergent or divergent at a number z'. We use binary search to test every real number between 0 and 1; in every test, when $V_i(z')$ converges, we set bigger z' next time, but when $V_i(z')$ diverges, we set smaller z' next time. Running more iterations will obtain more precise radius.

4. Results

In 2011, Koslicki [1] gave an efficient way to compute the topological entropy of DNA sequence. He used fixed length depending on subword size to compute topological entropy of sequence. For example, in Figure 10 (all DNA and amino acid data can be found in NCBI website, http://www.ncbi.nlm.nih.gov/), the sequence length is 1027 characters, and there are three subword sizes 2, 3, and 4 with blue, red, and green lines, respectively. For larger subword size, much larger fragment is required for complexity computation. The required fragment size grows exponentially, while the length of sequence is not dependent on the growth rate of subword size, so it is not a good method for us overall.

We present a new method called structural complexity in previous sections, and there are several benefits from using our method instead of Koslicki method, described as follows.

(1) Our results are very different from those obtained by the topological entropy method; see the colored lines in Figures 11~14. These figures showed that our method is much sensitive to certain arrangements of the elements in the sequence.

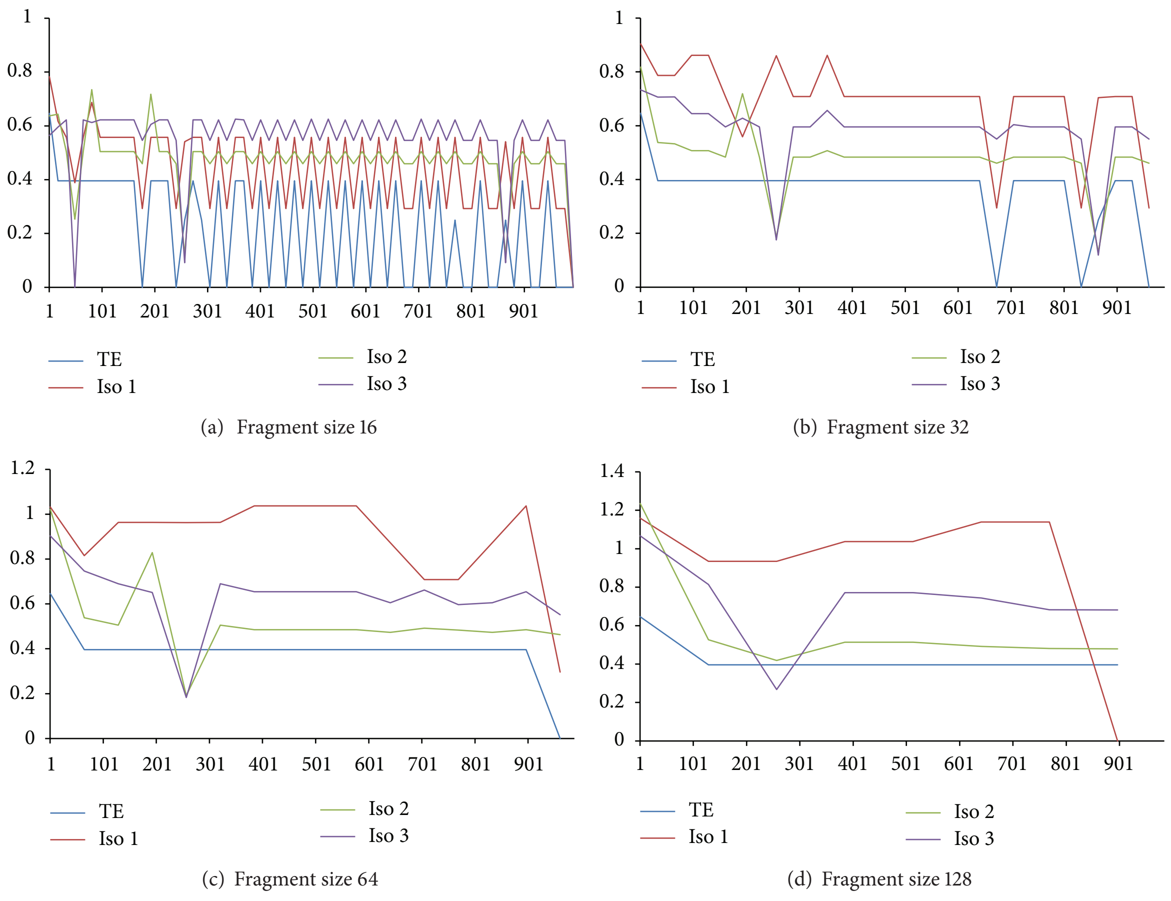


Figure 13: Compare with different methods.

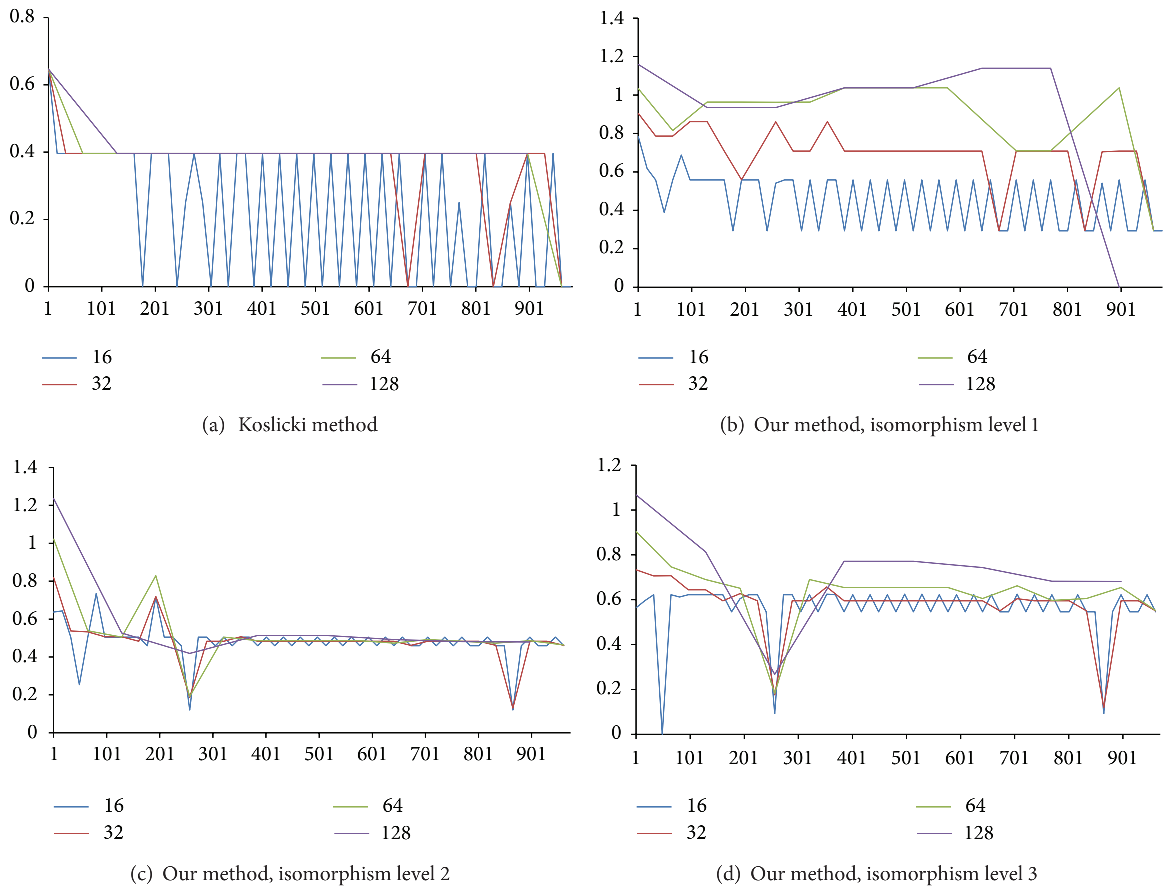
(2) Two different characters that exchange position will change value since Koslicki method just calculates the statistical values without structural information. Result was shown in Figure 11 bottom chart; the test sequence repeats the same subword several times. For blue line, all complexity values from topological entropy are equal within the region of repeated subwords. For red line, complexity values depend on the structure of subword. When the fragment of sequence is different from each other, our method will evaluate to different values.

(3) Our method can also calculate amino acid sequences. The Koslicki method depends on alphabet size and subword size, for example, in the basic length 2 substring calculation; since standard amino acid types have up to 20, it requires a minimum length of $20^2 + 2-1$ to calculate, but the amino acid strings are usually very short. Sometimes, Koslicki method cannot compute the amino acid sequence efficiently. Figure 12 shows that complexity of amino acid sequence can also be calculated by our method.

We also did experiments with lots of data, including fixed fragment size and fixed method on test sequences (see Figures 13 and 14). Here, we redefine the Koslicki method; the fragment size is no longer dependent on subword size. Instead, fixed length fragment like our method is applied. This change allows us to compare the data easier, and not restricted to the exponentially growing fragment size anymore. In Figure 13, we found that for larger fragment, the complexity curve will become smoothly because fragments for each data point contain more information. And we note that there is a common local peak value of those figures; the *simple sequence region* is big enough that our fragment size still contains the same simple sequence.

When we compare with the same method shown in Figure 14, we found the same situation more obviously. Thus, if we have many complexity values with different sizes, we have the opportunity to restore the portion of the DNA.

4.1. Application to Virus Sequences Database and Other Sequences. Now we can apply our technique to Chinese word sequences. Togawa et al. [18] gave a complexity of Chinese words, but his study was based on the number of strokes, which is different from our method. Here we use Big5 encoding for our system. Since the number of Chinese words is larger than 10000, we cannot directly use words as alphabet, so we need some conversion. We read a Chinese word into four hexadecimal letters so that we can replace the sequence with tree representation and compute the complexity.



(a) Koslicki method
(b) Our method, isomorphism level 1
(c) Our method, isomorphism level 2
(d) Our method, isomorphism level 3

FIGURE 14: Compare with different fragment sizes.

When it comes to biomedical section, we can create virus comparison database. Once a new virus or prion has been found, it will be easy to select corresponding drugs at the first time, according to cross comparison with each other by complexity in the database. We focus on most important viruses in recent years, such as *Escherichia coli* O157:H7 (*E. coli* o157), Enterovirus 71 (EV71), Influenza A virus subtype H1N1 (H1N1), Influenza A virus subtype H5N1 (H5N1), and severe acute respiratory syndrome (SARS). In recent years, these viruses have a significant impact and threat on the human world. We test these viruses and prions listed in Table 4. Here we can see that all prion regions cannot be analyzed by Koslicki method, but we can do it.

Finally, if any object can be written as a sequence, and there exists tree representation with alphabet of sequence, we can compute the complexity of the object.

5. Summary

In this paper, we give a method for computing complexity of DNA sequences. The traditional method focused on the statistical data or simply explored the structural complexity without value. In our method, we transform the DNA sequence to DNA tree with tree representations at first.

Then we transform the tree to context-free grammar format, so that it can be classified. Finally, we use redefined generating function and find the complexity values. We give a not only statistical but also structural complexity for DNA sequences, and this technique can be used in many important applications.

Acknowledgment

This work was supported by the National Science Council under project NSC 100-2221-E-002-234-MY3.

References

[1] D. Koslicki, "Topological entropy of DNA sequences," *Bioinformatics*, vol. 27, no. 8, Article ID btr077, pp. 1061–1067, 2011.

[2] C. Cattani, G. Pierro, and G. Altieri, "Entropy and multifractality for the myeloma multiple tet 2 gene," *Mathematical Problems in Engineering*, vol. 2012, Article ID 193761, 14 pages, 2012.

[3] S. Manna and C. Y. Liou, "Reverse engineering approach in molecular evolution: simulation and case study with enzyme proteins," in *Proceedings of the International Conference on Bioinformatics & Computational Biology (BIOCOMP '06)*, pp. 529–533, 2006.

[4] R. Zhang and C. T. Zhang, "Z curves, an intutive tool for visualizing and analyzing the DNA sequences," *Journal of*

Biomolecular Structure and Dynamics, vol. 11, no. 4, pp. 767–782, 1994.

[5] P. Tiño, "Spatial representation of symbolic sequences through iterative function systems," *IEEE Transactions on Systems, Man, and Cybernetics A*, vol. 29, no. 4, pp. 386–393, 1999.

[6] C. K. Peng, S. V. Buldyrev, A. L. Goldberger et al., "Long-range correlations in nucleotide sequences," *Nature*, vol. 356, no. 6365, pp. 168–170, 1992.

[7] B. L. Hao, H. C. Lee, and S. Y. Zhang, "Fractals related to long DNA sequences and complete genomes," *Chaos, solitons and fractals*, vol. 11, no. 6, pp. 825–836, 2000.

[8] C. Cattani, "Fractals and hidden symmetries in DNA," *Mathematical Problems in Engineering*, vol. 2010, Article ID 507056, 31 pages, 2010.

[9] C. Y. Liou, T. H. Wu, and C. Y. Lee, "Modeling complexity in musical rhythm," *Complexity*, vol. 15, no. 4, pp. 19–30, 2010.

[10] P. Prusinkiewicz, "Score generation with lsystems," in *Proceedings of the International Computer Music Conference*, pp. 455–457, 1986.

[11] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer, New York, NY, USA, 1996.

[12] P. Worth and S. Stepney, "Growing music: musical interpretations of L-systems," in *Applications of Evolutionary Computing*, vol. 3449 of *Lecture Notes in Computer Science*, pp. 545–550, Springer, Berlin, Germany, 2005.

[13] A. Lindenmayer, "Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 300–315, 1968.

[14] "Wikipedia: L-system—Wikipedia, the free encyclopedia," 2012.

[15] H. Barlow, "Unsupervised learning," *Neural Computation*, vol. 1, no. 3, pp. 295–311, 1989.

[16] R. Badii and A. Politi, *Complexity: Hierarchical Structures and Scaling in Physics*, vol. 6, Cambridge University Press, Cambridge, UK, 1999.

[17] W. Kuich, "On the entropy of context-free languages," *Information and Control*, vol. 16, no. 2, pp. 173–200, 1970.

[18] T. Togawa, K. Otsuka, S. Hiki, and H. Kitaoka, "Complexity of chinese characters," *Forma*, vol. 15, pp. 409–414, 2001.