國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

上下文相關的符號序列複雜度

The Complexity of Context-Sensitive Sequences

林建旻

Chien-Min Lin

指導教授：劉長遠博士

Advisor: Cheng-Yuan Liou, Ph.D.

中華民國 103 年 1 月

January, 2014

國立臺灣大學碩士學位論文
口試委員會審定書

上下文相關的符號序列複雜度

The Complexity of Context-Sensitive Sequences

本論文係林建旻君（學號 R98922120）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 102 年 11 月 27 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

劉長遠

（指導教授）

呂育道　　[illegible]

[illegible]

系 主 任　許永真

致謝

此篇論文能夠完成，首先要感謝我的指導教授劉長遠老師，老師在我研究期間給了許多有用的建議和可能的研究方向，使我在研究過程中獲益良多。除此之外老師也十分關心學生的近況和未來的方向，不時給予叮嚀與幫助，真的十分感謝。也感謝百忙之中抽空擔任口試委員的呂育道教授、彭智楹教授和黃昭綺學姐，並且提出了寶貴的建議及指導，讓此篇論文的內容更趨完善。

另外感謝曾聖翰學長、黃柏翔和翁正勳同學整理的資料，讓我能接續其相關的研究內容。也感謝演算法實驗室同學、學長姐及友人們的多所協助，不論是研究、課業或是日常生活上，受大家照顧很多。特別感謝從大學一路互相扶持的葉士賢同學，在我整個研究所期間幫了相當多的忙，謝謝。

最後感謝我的家人，至始至終都堅定的支持我完成學業，並給與生活上必要的協助。感謝女友佩儀，在苦悶而緩慢的研究期間悉心照料，讓我能消化負面情緒持續進行研究。最後感謝參與我研究生涯的每一位親人及朋友，謝謝大家。

中文摘要

在此篇論文中，我們使用 L 系統建模複雜度方法來計算上下文相關文法的複雜度。由於在現有文獻中已被證明出目前並沒有一般性的演算法用以計算上下文相關文法的複雜度，所以我們選擇了幾個常見的上下文相關文法例子來實作，為此我們改進了先前的 L 系統建模複雜度方法，使其能夠處理任意長度的符號序列。

關鍵字：結構複雜度、樹狀結構表示法、序列、上下文相關文法、L 系統

Abstract

This article discusses how to apply the L-system modeling complexity method to context-sensitive sequences. Since it is proved that there exists no general calculation method to compute the entropy of context-sensitive languages, we choose some common context-sensitive languages and analyze them case by case. For that purpose, we extend the capability of the modeling complexity method in previous work. Our method can deal with arbitrary length sequences.

Key words: Structural complexity, Tree representation, Sequence, Context-sensitive grammar, L-system

Contents

口試委員會審定書 **i**

致謝 **ii**

中文摘要 **iii**

Abstract **iv**

Contents **v**

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**

1.1 The complexity of the L-system 1

1.2 Preliminary 2

1.3 Other Applications 12

2 The Complexity of Context-Sensitive Sequences **14**

3 Extension **31**

4 Summary **34**

A Code **36**

Bibliography **38**

List of Figures

1.1 The tree representations of words a, b, c and d. 3

1.2 From tree representations to a text tree. 3

1.3 Bracketed string representations for tree representations. 4

1.4 The bracketed string representation of T. 4

1.5 The rewiting rules of T. 5

1.6 The rewiting rules for the bracketed string of T. 5

1.7 The rewiting rules for the bracketed string of T ignored all nonterminals. 7

1.8 The converge test result of the generating function $V_1^m(z)$. 11

1.9 The rhythmic tree of jazz standard Blues For Alice. 12

1.10 Tree representations of nucleotide bases. 13

1.11 DNA sequence represented with text tree. 13

2.1 The text tree of Case 2.1 for $n = 4$. 15

2.2 The structure of minimum difficult tree. 16

2.3 The text tree of Case 2.1 for $n = 4$, each node is numbered in BFS order. . 17

2.4 The complexity diagram of Case 2.1 for $n = 1, 2, \ldots, 64$ on all isomorphic depths. 19

2.5 The complexity diagram of Case 2.1 for $n = 1, 2, \ldots, 64$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.17967} + 1$. 20

2.6 The text tree of Case 2.2 for $n = 3$. 21

2.7 The complexity diagram of Case 2.2 for $n = 1, 2, \ldots, 64$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.25} + 1$. 22

2.8 The complexity diagram of all sequences of Case 2.3 for $n = 1$ on isomorphic depth 0. 23
2.9 The complexity diagram of all sequences of Case 2.3 for $n = 2$ on isomorphic depth 0. 23
2.10 The complexity diagram of all sequences of Case 2.3 for $n = 3$ on isomorphic depth 0. 24
2.11 The complexity diagram of all sequences of Case 2.3 for $n = 4$ on isomorphic depth 0. 24
2.12 The complexity diagram of all sequences of Case 2.3 for $n = 5$ on isomorphic depth 0. 24
2.13 The complexity diagram of all sequences of Case 2.3 for $n = 6$ on isomorphic depth 0. 25
2.14 The complexity diagram of all sequences of Case 2.3 for $n = 7$ on isomorphic depth 0. 25
2.15 The complexity diagram of all sequences of Case 2.3 for $n = 8$ on isomorphic depth 0. 25
2.16 The complexity diagram of all sequences of Case 2.3 for $n = 9$ on isomorphic depth 0. 26
2.17 The complexity diagram of all sequences of Case 2.3 for $n = 10$ on isomorphic depth 0. 26
2.18 The complexity diagram of all sequences of Case 2.3 for $n = 1, 2, \ldots, 10$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.404} + 1$. 26
2.19 The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0. 27
2.20 The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0. 27
2.21 The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0. 28

2.22 The complexity diagram of all sequences of Case 2.4 for $n = 4$ on isomorphic depth 0. 28
2.23 The complexity diagram of all sequences of Case 2.4 for $n = 5$ on isomorphic depth 0. 28
2.24 The complexity diagram of all sequences of Case 2.4 for $n = 6$ on isomorphic depth 0. 29
2.25 The complexity diagram of all sequences of Case 2.4 for $n = 7$ on isomorphic depth 0. 29
2.26 The complexity diagram of all sequences of Case 2.4 for $n = 8$ on isomorphic depth 0. 29
2.27 The complexity diagram of all sequences of Case 2.4 for $n = 9$ on isomorphic depth 0. 30
2.28 The complexity diagram of all sequences of Case 2.4 for $n = 10$ on isomorphic depth 0. 30
2.29 The complexity diagram of all sequences of Case 2.4 for $n = 1, 2, \ldots, 10$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.05514} + 1$. 30

3.1 The ternary text tree of Case 2.1 for $n = 3$. 31

A.1 Code of previous work. 36
A.2 Paper code. 37

List of Tables

1.1 The rewriting rules of T. 7

1.2 Classification Based on the Similarity of Rewriting Rules of T. 8

1.3 The values for the class parameters of Table 1.2, on isomorphic depth 0. . 10

2.1 Rewriting Rules of the text tree in Figure 2.1. 15

2.2 Classification Based on the Similarity of Rewriting Rules in Table 2.1. (The star symbols “*” point out where the forward-referencing rules are.) 16

2.3 Classification Based on the Similarity of Rewriting Rules in Table 2.1 by applying Algorithm 1. 19

2.4 Rewriting Rules of the text tree in Figure 2.6. 21

2.5 Classification Based on the Similarity of Rewriting Rules in Table 2.4 by applying Algorithm 1. 22

3.1 Rewriting Rules of the ternary text tree in Figure 3.1. 31

3.2 Classification Based on the Similarity of Rewriting Rules in Table 3.1. . . 32

3.3 The values for the class parameters of Table 3.2, on isomorphic depth 0. . 32

Chapter 1

Introduction

It is proved that "there exists no general calculation method to compute the entropy of the language generated by a context-sensitive grammar" by Kaminger [1]. Furthermore, there is no procedure to decide whether the language of any context-sensitive grammar is finite or not [2]. Thus, we choose some common context-sensitive languages and analyze their complexity case by case. For that purpose, we introduce the L-system in the following section.

1.1 The complexity of the L-system

The Lindenmayer system, or L-system for short, was introduced by the biologist Aristid Lindenmayer in 1968 [3]. The L-system is a type of string-rewriting mechanism such as Chomsky grammars. 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. The L-system is defined as follows.

Definition 1.1. *The L-system grammars are very similar to the Chomsky grammars, defined as a tuple [4]:*

$$G = (V, \omega, P),$$

where

1. $V = \{s_1, s_2, \ldots, s_n\}$ *is an alphabet,*

2. *ω (start, axiom, or initiator) is a string of symbols from V defining the initial state of the system,*
3. *P is defined by a production map $P : V \to V^*$ with $s \to P(s)$ for each s in V. The identity production $s \to s$ is assumed. These symbols are called constants or terminals.*

The following definition is given by Liou et al. [5], with this definition we can compute the complexity of L-system.

Definition 1.2. *The complexity of L-system.*
The complexity of L-system is defined as

$$K_0 = -\ln R,$$

where R is the radius of convergence.

The radius of convergence R can be derived from rewriting rules of L-system. We will show how to compute R, and introduce some terms in the following section.

1.2 Preliminary

The major difference between L-system and Chomsky grammars is the technique used to apply productions. In L-systems, rewriting rules are applied parallel and simultaneously, whereas in Chomsky grammars, rewriting rules are applied sequentially to replace all the letters in a given word. Parallel production application has an essential impact on the formal properties of rewriting systems. For example, there are languages which can be generated by context-free L-systems, which means all the rewriting rules of the L-system are context-free, but not by context-free Chomsky grammars. In the remainder of this paper, whenever we say an L-system, we suppose it to be context-free.

The L-system uses some initial and rewriting rules to construct beautiful graphs. Since it can construct a tree from rewriting rules, it can also extract rewriting rules from a tree. For computing the complexity of a grammar, we use fixed tree presentations for the input sequence generated by the grammar. For example, if the input sequence is $abcd$, we tranfer each word of to an unique tree representation, i.e., a single node for "a", a node with only one left child for "b", a node with only one right child for "c" and a node with both left and right child for "d" (see Figure 1.1). After replacing every word to tree representations, we combine each two consecutive trees to a bigger tree by creating a new node as the root of these two subtrees. Repeat the process level by level, the input sequence will be transfer to a text tree T (see Figure 1.2).

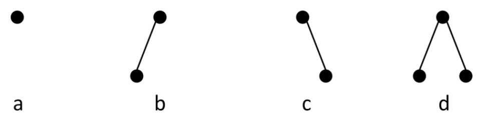


Figure 1.1: The tree representations of words a, b, c and d.

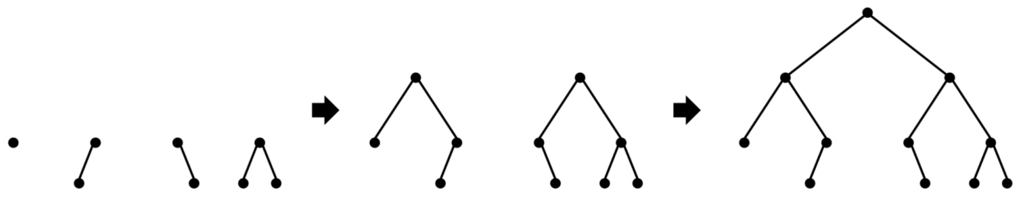


Figure 1.2: From tree representations to a text tree.

For computing the complexity of L-system, we need some rules for converting tree to another structure. We use a stack similarly structure to represent the hierarchy of tree, called *bracketed string*. The tree structure can transfer to a unique bracketed string by the following symbols, and it can transfer back to the original tree:

1. F : the current location of tree nodes; it can be replaced by any word or be omitted;
2. $+$: the following string will express the right subtree;
3. $-$: the following string will express the left subtree;
4. [: this symbol is pairing with]; "[. . .]" denotes a subtree where ". . ." indicates all the bracketed strings of its subtree;
5.] : see [description.

Following the previous symbols, Figure 1.3 shows that how to represent the tree representations by bracketed strings. And Figure 1.4 is the bracketed string representaion of T.

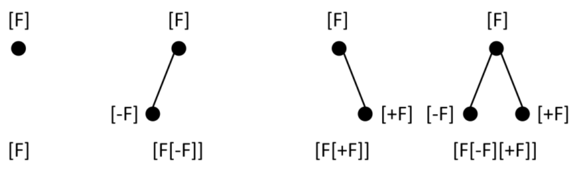


Figure 1.3: Bracketed string representations for tree representations.

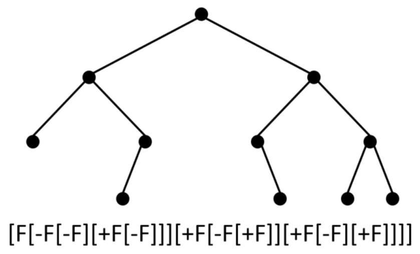


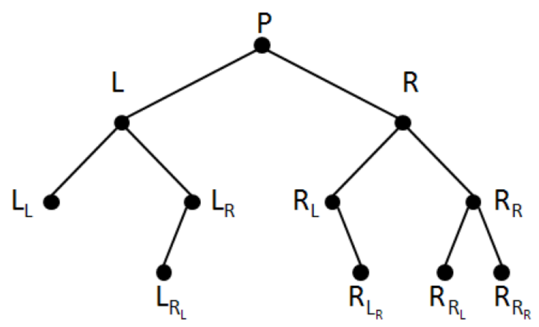
Figure 1.4: The bracketed string representation of T.

Now we explain how to generate rewriting rules from tree structures. We can apply distinct variables to each tree node. For any binary tree, for every tree node has left and right subtrees, they can always be explained in the following format:

$$P \to LR,$$

where P denotes the current node, L denotes its left subtree, and R denotes its right subtree, respectively. Thus, we can use rewriting rules to represent T (see Figure 1.5). We denote "ϕ" to be null.

Now, we know how to use rewriting rules to represent a tree, and we also know how to represent a tree with a bracketed string. We can also use rewriting rules to generate bracketed strings. In rewriting rules for tree structures, we write $P \to LR$ for a tree having left and right subtrees. Note that we call L and R the non-terminals. Such a tree will have a bracketed string as follows: $[[-F\ldots][+F\ldots]]$. It is clear that "$[-F\ldots]$" represents the left subtree, and that "$[+F\ldots]$" represents the right subtree. Therefore, we



$$\begin{aligned}
P &\rightarrow L\ R\\
L &\rightarrow L_L\, L_R\\
L_L &\rightarrow \phi\,\phi\\
L_R &\rightarrow L_{R_L}\phi\\
L_{R_L} &\rightarrow \phi\,\phi\\
R &\rightarrow R_L\, R_R\\
R_L &\rightarrow \phi\, R_{L_R}\\
R_{L_R} &\rightarrow \phi\,\phi\\
R_R &\rightarrow R_{R_L} R_{R_R}\\
R_{R_L} &\rightarrow \phi\,\phi\\
R_{R_R} &\rightarrow \phi\,\phi
\end{aligned}$$

Figure 1.5: The rewiting rules of T.

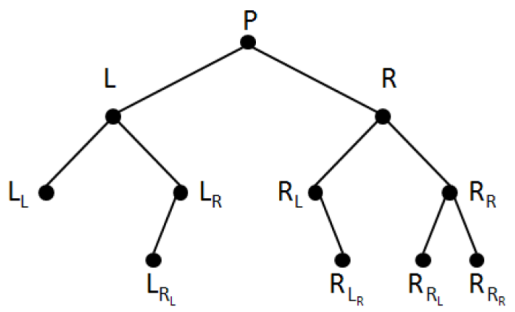
can replace the rewriting rules with

$$P \rightarrow [-FL][+FR]$$

$$L \rightarrow \ldots$$

$$R \rightarrow \ldots$$

where "…" is the rewriting rule for the bracketed string of each subtree. In this way, we do not have to write L for a left subtree and R for a right subtree; the orientation is already described in the bracketed string "$-F$" and "$+F$". For the sake of readabiliy, we replace the words such as "R_{R_L}" and "R_{R_R}" by "$T_{R_{R_L}}$" and "$T_{R_{R_R}}$" respectively. For those null rewriting rules "ϕ"s, we simply ignore the nulls. The new rewriting rules without trivial nulls are shown in Figure 1.6.



$$\begin{aligned}
P &\rightarrow [-FT_L][+FT_R]\\
T_L &\rightarrow [-FT_{L_L}][+FT_{L_R}]\\
T_{L_R} &\rightarrow [-FT_{L_{R_L}}]\\
T_R &\rightarrow [-FT_{R_L}][+FT_{R_R}]\\
T_{R_L} &\rightarrow [+FT_{R_{L_R}}]\\
T_{R_R} &\rightarrow [-FT_{R_{R_L}}][+FT_{R_{R_R}}]
\end{aligned}$$

Figure 1.6: The rewiting rules for the bracketed string of T.

When tree grows up, the rewriting rules may generate identical rules. These rules can generate exactly one bracketed string and, thus, exactly one tree structure. All these rules form a rule set, which represents a unique tree structure. In Figure 1.6, it is clear that $T_R \rightarrow [-FT_{R_L}][+FT_{R_R}]$ and $T_{R_R} \rightarrow [-FT_{R_{R_L}}][+FT_{R_{R_R}}]$ are almost the same. The only difference is that the subtrees of T_R are T_{R_L} and T_{R_R}, and the subtrees of T_{R_R} are $T_{R_{R_L}}$ and $T_{R_{R_R}}$. But they have the same structure: they both have a right subtree and do not have a left subtree. We will define two terms to express the similarity between two rewriting rules, and these terms can simplify complexity analysis.

Definition 1.3. *Homomorphism in rewriting rules.*
We define that rewriting rule R_1 and rewriting rule R_2 are homomorphisc if and only if they have the same structure.

In detail, rewriting rule R_1 and rewriting rule R_2 in tree structure 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 1.4. *Isomorphism on level X in rewriting rules.*
We define that rewriting rule R_1 and rewriting rule R_2 are isomorphisc on depth X if they are homomorphisc and their nonterminals are relatively isomorphic on depth $X-1$, denoted by $\mathsf{Iso}_X(R_1, R_2)$. Isomorphic on level 0 indicates homomorphism.

By using bracketed strings, we can obtain a much clearer definition of homomorphism. All homomorphic rewriting rules generating rules generate the same bracketed strings when all nonterminals are ignored. We ignore all nonterminals of the rewriting rules in Figure 1.7, we find P, T_L, T_R and T_{R_R} are homomorphic to each other; they generate the same bracketed string, $[-F][+F]$. T_{L_R} and T_{R_L} are not homomorphic to any of the other rules; its bracketed strings are $[-F]$ and $[+F]$.

After we define the similarity between rules by homomorphism and isomorphism, we can classify all the rules into different subsets based on their similarity. Now we list all the rewriting rules of T into Table 1.1 but ignore terminal rules such as "$\rightarrow \phi$" and transfer the name of each rule to class name (or class number). After performing classification,

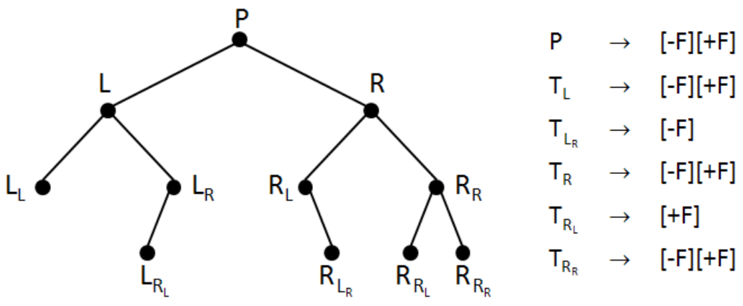


Figure 1.7: The rewiting rules for the bracketed string of T ignored all nonterminals.

we obtain not only a new rewriting rule set but also a context-free grammar, which can be converted to automata.

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

Table 1.1: The rewriting rules of T.

In Table 1.1, rules $P \to [-FT_L][+FT_R], T_L \to [-F][+FT_{L_R}], T_R \to [-FT_{R_L}][+FT_{R_R}]$ and $T_{R_R} \to [-F][+F]$ are isomorphic on depth 0 and assigned to C_1. For every rule T_i, we denote $H(T_i)$, the *height* of T_i, to be the longest distance in the text tree from T_i to the terminals derived from T_i. Since there are no isomorpic rules except terminal rules on depth 1, the class number of each rule is set by the order from highest rule to lowest rule and from left to right. That is, we first set the highest node P to C_1, and then set the second high rules T_L and T_R to C_2 and C_3 respectively. The class number of third high rules T_{L_R}, T_{R_L} and T_{R_R} are set C_4, C_5 and C_6 respectively, and the terminal rules T_{L_L}, $T_{L_{R_L}}$, $T_{R_{L_R}}$, $T_{R_{R_L}}$ and $T_{R_{R_R}}$ have the largest class number C_7. The complete table is shown in Table 1.2. Now we can define the complexity of a rewriting rule set as follows:

Definition 1.5. *Topological entropy of a context-free grammar.*
The topological entropy K_0 of a context-free grammar can be evaluated by means of the following three procedures [6, 7]:

Classification of Rules	Isomorphic Depth #0	Isomorphic Depth #1
Class #1	$(1)C_1 \to C_1C_1$	$(1)C_1 \to C_2C_3$
	$(1)C_1 \to C_3C_1$	
	$(1)C_1 \to C_4C_2$	
	$(1)C_1 \to C_4C_4$	
Class #2	$(1)C_2 \to C_4C_\phi$	$(1)C_2 \to C_7C_4$
Class #3	$(1)C_3 \to C_\phi C_4$	$(1)C_3 \to C_5C_6$
Class #4	$(5)C_4 \to C_\phi C_\phi$	$(1)C_4 \to C_7C_\phi$
Class #5		$(1)C_5 \to C_\phi C_7$
Class #6		$(1)C_6 \to C_7C_7$
Class #7		$(5)C_7 \to C_\phi C_\phi$

Table 1.2: Classification Based on the Similarity of Rewriting Rules of T.

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

$$V_i \to t_{i1}U_{i1}, t_{i2}U_{i2}, \ldots, t_{ik_i}U_{ik_i},$$

where $\{t_{i1}, t_{i2}, \ldots, t_{ik_i}\}$ are terminals and $\{U_{i1}, U_{i2}, \ldots, U_{ik_i}\}$ are non-terminals. The formal algebraic expression for each variable is

$$V_i = \sum_{j=1}^{k_i} t_{ij}U_{ij}$$

2. *By replacing every terminal t_ij with an auxiliary variable z, one obtains the generating function*

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

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 $N_i(n)$, $N(n) = \max\{N_i(n)$, over all $i\}$.*

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

$$K_0 = -\ln R.$$

The productions of Liou et al. [5] method have some difference from the aforementioned definitions. Therefore, they had given some slightly modifications in deriving the complexity of the L-system, which is rewritten as follows:

Definition 1.6. *Generating function of a conext-free grammar.*

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

$$\begin{aligned} U_{i1} &\rightarrow V_{a_{i11}} V_{a_{i12}} \\ U_{i2} &\rightarrow V_{a_{i21}} V_{a_{i22}} \\ \ldots &\rightarrow \quad \ldots \\ U_{in_i} &\rightarrow V_{a_{in_i1}} V_{a_{in_i2}} \end{aligned}$$

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

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

If V_i does not have any non-terminal, we set $V_i(z) = 1$.

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 text tree. After obtaining the largetst 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 rhythmic tree as $K_0 = -\ln R$.

For example, we compute the modeling complexity value of sequence "*abcd*" on isomorphic depth 0 now. According to the definition, the given values for the class parameters are listed in Table 1.3. There are 4 classes, so we obtain the formulas for $V_1(z')$,

	Classification of Rules	Isomorphic Depth #0
$(n = 4)$	Class #1	$(1)C_1 \rightarrow C_1\ \ C_1$
	$(n_1 = 4)$	$n_{11}\quad a_{111}a_{112}$
		$(1)C_1 \rightarrow C_3\ \ C_1$
		$n_{12}\quad a_{121}a_{122}$
		$(1)C_1 \rightarrow C_4\ \ C_2$
		$n_{13}\quad a_{131}a_{132}$
		$(1)C_1 \rightarrow C_4\ \ C_4$
		$n_{14}\quad a_{141}a_{142}$
	Class #2	$(1)C_2 \rightarrow C_4\ \ C_\phi$
	$(n_2 = 1)$	$n_{21}\quad a_{211}a_{212}$
	Class #3	$(1)C_3 \rightarrow C_\phi\ \ C_4$
	$(n_3 = 1)$	$n_{31}\quad a_{311}a_{312}$
	Class #4	$(5)C_4 \rightarrow C_\phi\ \ C_\phi$
	$(n_4 = 1)$	$n_{41}\quad a_{411}a_{412}$

Table 1.3: The values for the class parameters of Table 1.2, on isomorphic depth 0.

.

$V_2(z')$, $V_3(z')$ and $V_4(z')$. They are:

$$V_\phi(z') = 1$$

$$V_4(z') = 1$$

$$V_3(z') = \frac{\sum_{p=1}^{n_3} n_{3p} z' V_{a_{3p1}}(z') V_{a_{3p2}}(z')}{\sum_{q=1}^{n_3} n_{3q}} = \frac{z' \times (1 \times V_\phi(z') \times V_4(z'))}{1} = z'$$

$$V_2(z') = \frac{\sum_{p=1}^{n_2} n_{2p} z' V_{a_{2p1}}(z') V_{a_{2p2}}(z')}{\sum_{q=1}^{n_2} n_{2q}} = \frac{z' \times (1 \times V_4(z') \times V_\phi(z'))}{1} = z'$$

$$\begin{aligned} V_1(z') &= \frac{\sum_{p=1}^{n_1} n_{1p} z' V_{a_{1p1}}(z') V_{a_{1p2}}(z')}{\sum_{q=1}^{n_1} n_{1q}} \\ &= \frac{z' \times (1 \times V_1(z') \times V_1(z') + 1 \times V_3(z') \times V_1(z') + 1 \times V_4(z') \times V_2(z') + 1 \times V_4(z') \times V_4(z'))}{4} \\ &= \frac{z' V_1(z')^2 + z'^2 V_1(z') + z'^2 + z'}{4} \end{aligned}$$

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

$$z' V_1(z')^2 + (z'^2 - 4) V_1(z') + (z'^2 + z') = 0$$

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

$$V_1(z') = \frac{4 - z'^2}{2z'} \pm \frac{1}{2z'} \sqrt{z'^4 - 4z'^3 - 12z'^2 + 16}$$

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

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

$$V_i^0(z') = 1$$

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 $V_i^m(z')$ is convergent or divergent at a number z'. We use binary search to every real number between 0 and 1; in every test, when $V_i^m(z')$ converges, we set bigger z' next time, but when $V_i^m(z')$ diverges, we set smaller z' next time. Running more iterations will obtain more precise radius. Here, we set the iteration for 8 times.

The converge test result of generating function $V_1^m(z')$ is shown in Figure 1.8. The generating function $V_1^m(z')$ diverges when $z' > 0.369109$, that is, $z'^{max} = 0.369109$ and the complexity value is $K_0 = -ln(R) = -ln(z'^{max}) = 0.996663$.

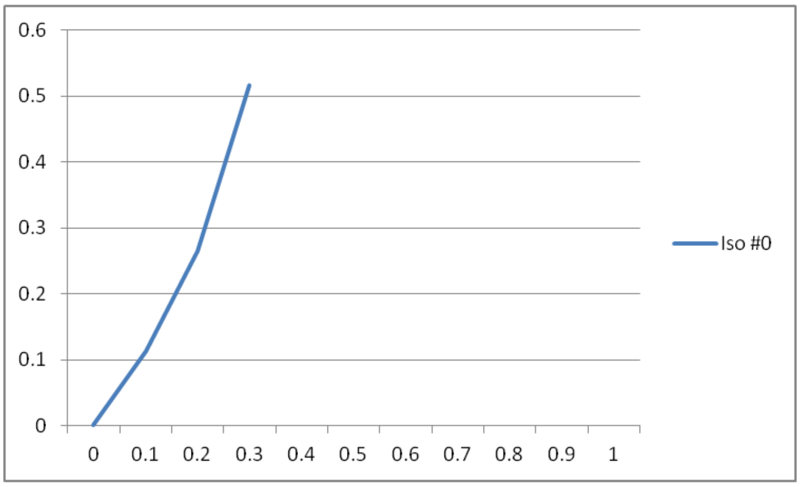


Figure 1.8: The converge test result of the generating function $V_1^m(z)$.

1.3 Other Applications

With Definition 1.6, Liou et al. gave a new idea in modeling complexity for music rhythms [5] and DNA sequences [8]. The former paper applies L-system to construct rhythmic trees, and analyzes the hierarchical characteristics of rhythm from the tree structure. A rhythmic tree is a tree of which each subtree is also a rhythmic tree. Each tree node represents the total beat duration that is equal to the sum of all those of its descendants. Given a rhythm, the direct way to represent it with an L-system is to construct rewriting rules that replace longer metrical units with shorter ones. An example is shown in Figure 1.9. Similarly, the latter paper applies L-system to construct DNA trees, and determines the dissimilarity between two given DNA sequences. 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 children. There are four fixed tree representations for nucleotide bases A, C, T and G (see Figure 1.10). When transfering a sequence to DNA tree, we replace every word to the corresponding tree presentations, and two consecutive trees can combine to a bigger tree. Following the previous steps, a DNA tree will be constructed (see Figure 1.11). With some slightly modifications, transfer text sequences to binary strings, this technique can be applied to literary works, even Chinese word sequences [9].

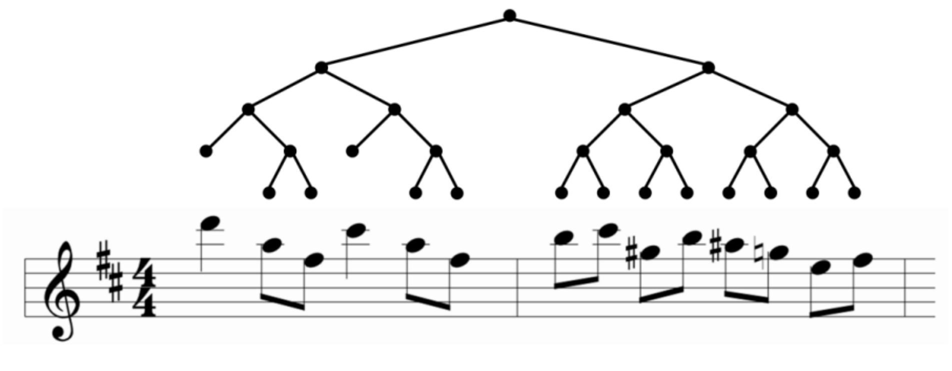


Figure 1.9: The rhythmic tree of jazz standard Blues For Alice.

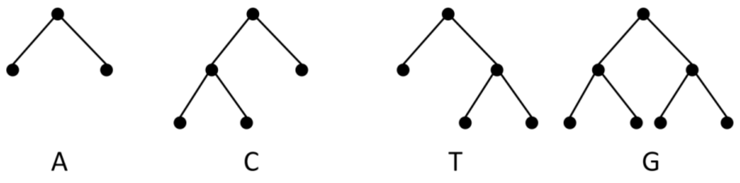


Figure 1.10: Tree representations of nucleotide bases.

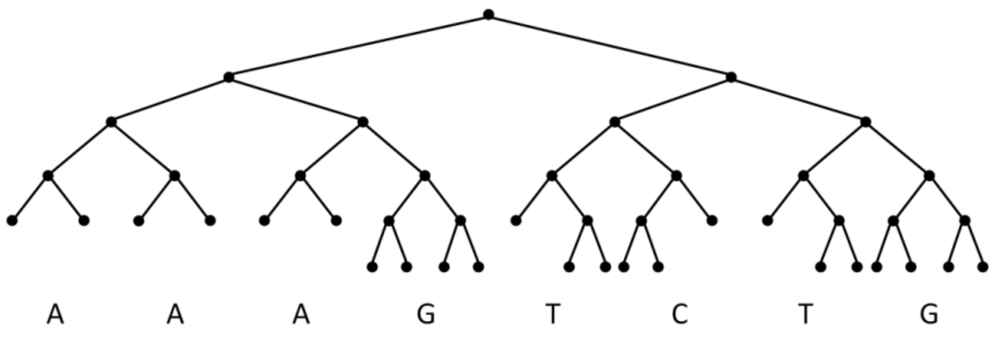


Figure 1.11: DNA sequence represented with text tree.

Chapter 2

The Complexity of Context-Sensitive Sequences

In this chapter, we apply the L-system modeling complexity method to some context-sensitive language cases. Different from the previous work we mentioned above, our method can deal with arbitrary length sequences. Basically, we have almost the same procedure with those previous work but the classification part. We made some modifications to the classifition part for avoiding forward-referencing rules, and we will show that in the following case. Note that for every cases in this chapter, we use the same tree representation for each word "a", "b", "c" and "d", as Figure 1.1 shows.

Case 2.1. *Given a context-sensitive grammar G_1 and the language generated by G_1 is*

$$L(G_1) = \{a^n b^n c^n | n \geq 1\}.$$

The language generated by G_1 is $aa \ldots abb \ldots bcc \ldots c$ where the number of a, b and c are the same. For example, we start at the sequence for $n = 4$, i.e., $aaaabbbbcccc$. To apply the L-system method, first, we tranfer each word of input sequence to corresponding tree representations, and combine all the tree representations to the text tree $T_1(4)$ of $L(G_1)$ for $n = 4$ (see Figure 2.1). Since we have the text tree, we can list all the rewriting rules of Figure 2.1 into Table 2.1 but terminal rules, and transfer the name of each rule to class numbers for listing the classification Table 2.2.

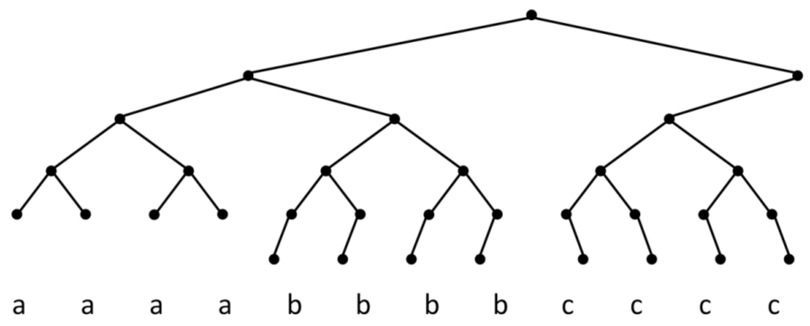


Figure 2.1: The text tree of Case 2.1 for $n = 4$.

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

Table 2.1: Rewriting Rules of the text tree in Figure 2.1.

Unfortunately, in this case, we can not apply the L-system method to compute the complexity directly since there are forward-referencing rules "$C_2 \to C_1C_\phi$" and "$C_3 \to C_2C_\phi$" in the classification Table 2.2. By the formula of definition 1.6, we know $V_i(z)$ can be computed by $V_j(z)$ only if $i \leq j$. The original classification method causes forward-referencing rules only if the text tree is a *difficult tree*. We define a four-nodes tree stucture where the root s has left child t only and t has both left child u and right child v to be a *minimum difficult tree* and t to be a *stopper* (see Figure 2.2). A difficult tree is a tree contains at least one minimum difficult tree as its subtree. Note that stoppers, or minimum difficult trees, only appear on the path from root to the rightmost leaf. Also, tree representations should not be difficult trees. By traveling all nodes on the path, we can downwardly find all of the stoppers $t_1, t_2, \ldots, t_m$. Also, we give every node x of the text tree a *priority value*, denoted by $P(x)$, initially set to 0. For each stoppers t_i we set $P(t_i) = i$ and set

Classification of Rules	Isomorphic Depth #0	Isomorphic Depth #1	Isomorphic Depth #2, 3, 4
Class #1	(4)$C_1 \to C_1C_1$ (2)$C_1 \to C_1C_2$ (2)$C_1 \to C_2C_2$ (2)$C_1 \to C_3C_3$ (1)$C_1 \to C_4C_4$	(1)$C_1 \to C_2C_3$	(1)$C_1 \to C_2C_3$
Class #2	*(1)$C_2 \to C_1C_\phi$ (4)$C_2 \to C_4C_\phi$	(1)$C_2 \to C_2C_2$ (1)$C_2 \to C_4C_4$ (1)$C_2 \to C_5C_5$ (1)$C_2 \to C_6C_6$	(1)$C_2 \to C_6C_4$
Class #3	(4)$C_3 \to C_\phi C_4$	*(1)$C_3 \to C_2C_\phi$	(1)$C_3 \to C_5C_\phi$
Class #4	(12)$C_4 \to C_\phi C_\phi$	(2)$C_4 \to C_7C_7$	(1)$C_4 \to C_7C_7$
Class #5		(2)$C_5 \to C_8C_8$	(1)$C_5 \to C_8C_8$
Class #6		(2)$C_6 \to C_9C_9$	(1)$C_6 \to C_9C_9$
Class #7		(4)$C_7 \to C_9C_\phi$	(2)$C_7 \to C_{10}C_{10}$
Class #8		(4)$C_8 \to C_\phi C_9$	(2)$C_8 \to C_{11}C_{11}$
Class #9		(12)$C_9 \to C_\phi C_\phi$	(2)$C_9 \to C_{12}C_{12}$
Class #10			(4)$C_{10} \to C_{12}C_\phi$
Class #11			(4)$C_{11} \to C_\phi C_{12}$
Class #12			(12)$C_{12} \to C_\phi C_\phi$

Table 2.2: Classification Based on the Similarity of Rewriting Rules in Table 2.1. (The star symbols "*" point out where the forward-referencing rules are.)

the priority value for every descendant of t_i to $P(t_i)$, for $i = 1, 2, \ldots, m$. That makes the desendants of lower stoppers have larger priority value.

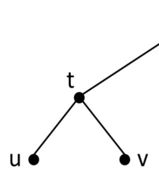


Figure 2.2: The structure of minimum difficult tree.

For example, we name each node of $T_1(4)$ by a number increasly in breadth-first search order (BFS for short, see Figure 2.3). First, we search for stoppers on the path from node $\#0$ to node $\#31$. In this case, there are only one stopper $\#5$ in $T_1(4)$ and we set $P(\#5) = 1$. Also, the priority value of all the descendants of node $\#5$ are set to be $P(\#5)$.

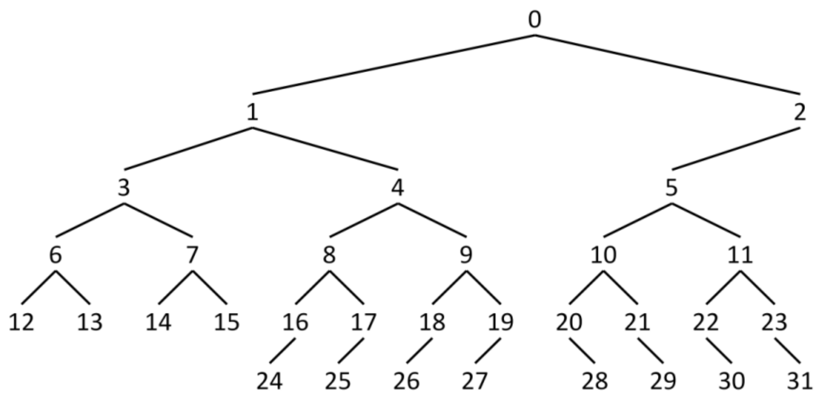


Figure 2.3: The text tree of Case 2.1 for $n = 4$, each node is numbered in BFS order.

Algorithm 1 New classification algorithm.

```
let $k$ be the isomorphic depth
let $c = 1$ be current class number
for $P = 0 \to$ largest priority value do
    for each node $x$ of $T$ in BFS order do
        if $H(x) > k$ and $C_x = 0$ and $P(x) = P$ then
            $C_x = c$
            $c = c + 1$
            for each node $y > x$ of $T$ in BFS order do
                if $H(y) > k$ and $C_y = 0$ and $P(y) = P$ and $\mathsf{Iso}_k(x, y)$ then
                    $C_y = C_x$
                end if
            end for
        end if
    end for
end for
for $l = k \to 0$ do
    for each node $x$ of $T$ in BFS order do
        if $H(x) = l$ and $C_x = 0$ then
            $C_x = c$
            $c = c + 1$
            for each node $y > x$ of $T$ in BFS order do
                if $C_y = 0$ and $\mathsf{Iso}_k(x, y)$ then
                    $C_y = C_x$
                end if
            end for
        end if
    end for
end for
```

Now we introduce the new classification method Algorithm 1. Applying it on difficult trees ensures no forward-referencing rules appear. The basic idea of this algorithm is dividing the classification into two parts. Let k be the isomorphic depth. The first part deal with each node x of the text tree where $H(x) > k$, in BFS order. When we classify these nodes, we have to make sure that for any x, y of the text tree, $C_x < C_y$ only if $P(x) < P(y)$. That ensures the upper text tree nodes classified properly, because the lower stoppers and their descendants will always have larger class number. The detail of the first part is shown in line 3 to line 15 of Algorithm 1. The second part deal with each node x of the text tree where $H(x) \leq k$, in BFS order. When we classify these nodes, we have to make sure that for any x, y of the text tree, $C_x < C_y$ only if $H(x) > H(y)$. That ensures the lower text tree nodes classified properly, and basically it is the classification method of those previous work. The detail of the second part is shown in line 16 to line 28 of Algorithm 1. Note that for each node x of the text tree, $C_x = 0$ initailly.

Now we show how Algorithm 1 works on $T_1(4)$, on isomorphic depth 1. The first part classify the nodes with height more than 1 in BFS order, i.e., $\#0$, $\#1$, $\#2$, $\#3$, $\#4$, $\#5$, $\#8$, $\#9$, $\#10$ and $\#11$. Start from the nodes with lower priority value, $\#0$, $\#1$, $\#2$, $\#3$, $\#4$, $\#8$ and $\#9$ are classified to be C_1, C_2, C_3, C_2, C_2, C_4 and C_4 respectively. Then the nodes with higher priority value, $\#5$, $\#10$ and $\#11$, are classified to be C_5, C_6 and C_6 respectively. The second part classify the rest of nodes. Start from the nodes with height equal to 1 in BFS order, $\#6$, $\#7$, $\#16$, $\#17$, $\#18$, $\#19$, $\#20$, $\#21$, $\#22$ and $\#23$ are set to be C_7, C_7, C_8, C_8, C_8, C_8, C_9, C_9, C_9 and C_9 respectively. Then the nodes with height less than 1, terminal nodes, are classified to be C_{10}.

By applying this algorithm on all isomorphic depths 0, 1, 2, 3 and 4, we have the new classification Table 2.3. With this new table, now we can apply the modeling complexity method. We show the result of L-system modeling complexity computations of Case 2.1 for $n = 1, 2, \ldots, 64$, on all isomorphic depths in Figure 2.4. The maximum isomorphic depth of each sequence of $L(G_1)$ is $H(T_1(n)) - 1$. For example, the maximum isomorphic depth of $L(G_1)$ for $n = 4$ is $H(T_1(4)) - 1 = 4$.

Classification of Rules	Isomorphic Depth #0	Isomorphic Depth #1	Isomorphic Depth #2, 3, 4
Class #1	$(3)C_1 \to C_1C_1$ $(2)C_1 \to C_1C_2$ $(2)C_1 \to C_2C_2$ $(1)C_1 \to C_5C_5$	$(1)C_1 \to C_2C_3$	$(1)C_1 \to C_2C_3$
Class #2	$(1)C_2 \to C_3C_\phi$ $(4)C_2 \to C_5C_\phi$	$(1)C_2 \to C_2C_2$ $(1)C_2 \to C_4C_4$ $(1)C_2 \to C_7C_7$	$(1)C_2 \to C_6C_4$
Class #3	$(1)C_3 \to C_3C_3$ $(2)C_3 \to C_4C_4$	$(1)C_3 \to C_5C_\phi$	$(1)C_3 \to C_5C_\phi$
Class #4	$(4)C_4 \to C_\phi C_5$	$(2)C_4 \to C_8C_8$	$(1)C_4 \to C_7C_7$
Class #5	$(12)C_5 \to C_\phi C_\phi$	$(1)C_5 \to C_6C_6$	$(1)C_5 \to C_8C_8$
Class #6		$(2)C_6 \to C_9C_9$	$(1)C_6 \to C_9C_9$
Class #7		$(2)C_7 \to C_{10}C_{10}$	$(2)C_7 \to C_{10}C_{10}$
Class #8		$(4)C_8 \to C_{10}C_\phi$	$(2)C_8 \to C_{11}C_{11}$
Class #9		$(4)C_9 \to C_\phi C_{10}$	$(2)C_9 \to C_{12}C_{12}$
Class #10		$(12)C_{10} \to C_\phi C_\phi$	$(4)C_{10} \to C_{12}C_\phi$
Class #11			$(4)C_{11} \to C_\phi C_{12}$
Class #12			$(12)C_{12} \to C_\phi C_\phi$

Table 2.3: Classification Based on the Similarity of Rewriting Rules in Table 2.1 by applying Algorithm 1.

In Figure 2.4 we can see, for those short sequences generated by G_1, the modeling complexity method does not work properly on high isomorphic depths. The short sequences have no self-referencing rules (like $C_1 \to C_1C_1$) on high isomorphic depths, which means the generating function of short sequences on high isomorphic depths never diverge when $0 < z < 1$. In other word, the modeling complexity method works properly on isomorphic depth 0 even for very short sequences. For this reason, we only list the results on isomorphic depth 0 for all the following cases.

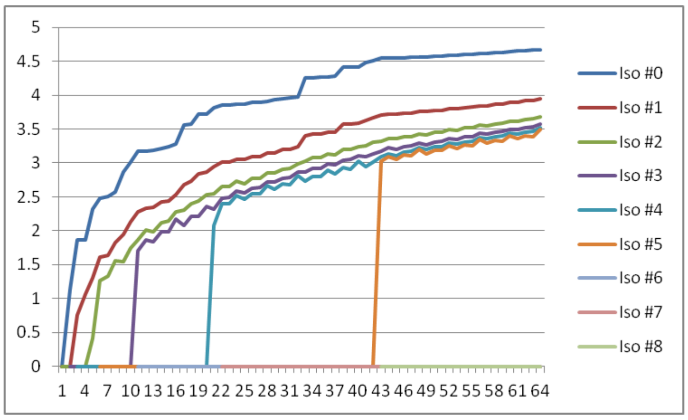


Figure 2.4: The complexity diagram of Case 2.1 for $n = 1, 2, \ldots, 64$ on all isomorphic depths.

On the other hand, we can use a single value α to represent the *grammar complexity* of any context-sensitive grammar. By the method of least squares, we can find a best-fit curve $H = \ln n^{\alpha} + 1$ of all the modeling complexity values. For those faster growing generating functions, they have larger grammar complexity. With this value, we can compare the complexity of different context-sensitive grammars easily. We show the result in Figure 2.5 and the grammar complexity of G_1 is $\alpha_1 = 1.17967$.

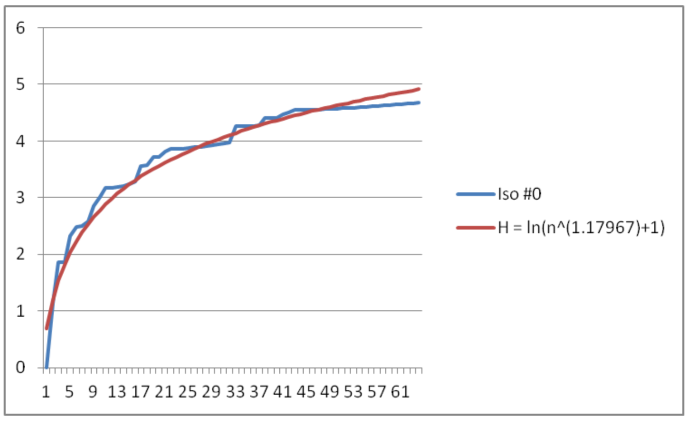


Figure 2.5: The complexity diagram of Case 2.1 for $n = 1, 2, \ldots, 64$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.17967} + 1$.

Case 2.2. *Given a context-sensitive grammar G_2 and the language generated by G_2 is*

$$L(G_2) = \{a^n b^n c^n d^n | n \geq 1\}.$$

Similar to Case 2.1, the language generated by G_2 is $aa \ldots abb \ldots bcc \ldots cdd \ldots d$ where the number of a, b, c and d are the same. With more complicated grammars can be used to parse other languages with even more letters. For example, we start at the sequence for $n = 3$. With the similar processes, we have the tree representations trasfered from each word of the input sequence, and the text tree $T_2(3)$ is shown in Figure 2.6. Then, we list all the rewriting rules from $T_2(3)$ into Table 2.4, and the classification Table 2.5. The result of L-system modeling complexity computations and the best-fit curve for Case 2.2 for $n = 1, 2, \ldots, 64$, on isomorphic depth 0 are shown in Figure 2.7. The grammar complexity of G_2 is $\alpha_2 = 1.25$.

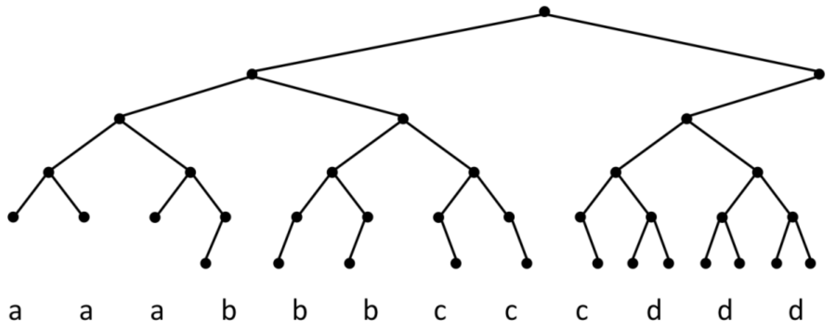


Figure 2.6: The text tree of Case 2.2 for $n = 3$.

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

$T_R \to [-FT_{R_L}]$
$T_{R_L} \to [-FT_{R_{L_L}}][+FT_{R_{L_R}}]$
$T_{R_{L_L}} \to [-FT_{R_{L_{L_L}}}][+FT_{R_{L_{L_R}}}]$
$T_{R_{L_{L_L}}} \to [+F]$
$T_{R_{L_{L_R}}} \to [-F][+F]$
$T_{R_{L_R}} \to [-FT_{R_{L_{R_L}}}][+FT_{R_{L_{R_R}}}]$
$T_{R_{L_{R_L}}} \to [-F][+F]$
$T_{R_{L_{R_R}}} \to [-F][+F]$

Table 2.4: Rewriting Rules of the text tree in Figure 2.6.

Classification of Rules	Isomorphic Depth #0
Class #1	$(3)C_1 \to C_1C_1$ $(1)C_1 \to C_1C_2$ $(1)C_1 \to C_2C_2$ $(1)C_1 \to C_3C_3$ $(1)C_1 \to C_6C_2$ $(1)C_1 \to C_6C_6$
Class #2	$(1)C_2 \to C_4C_\phi$ $(3)C_2 \to C_6C_\phi$
Class #3	$(2)C_3 \to C_\phi C_6$
Class #4	$(2)C_4 \to C_4C_4$ $(1)C_4 \to C_5C_4$ $(3)C_4 \to C_6C_6$
Class #5	$(1)C_5 \to C_\phi C_6$
Class #6	$(15)C_6 \to C_\phi C_\phi$

Table 2.5: Classification Based on the Similarity of Rewriting Rules in Table 2.4 by applying Algorithm 1.

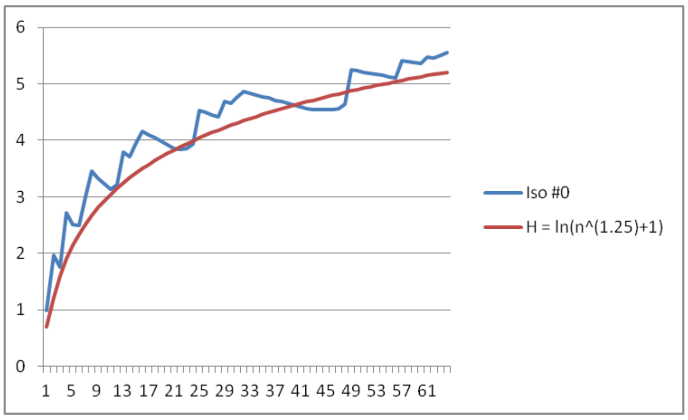


Figure 2.7: The complexity diagram of Case 2.2 for $n = 1, 2, \ldots, 64$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.25} + 1$.

Case 2.3. *Given a context-sensitive grammar G_3 and the language generated by G_3 is*

$$L(G_3) = \{ww|w \in \{a, d\}^n, n > 0\}.$$

Case 2.3 is known as copy language or duplicate language. Copy language is also a typical type of context-sensitive language. In this case, we list all sequences in $L(G_3)$ of length $n \times 2$, i.e., for $n = 3$ we list $aaaaaa$, $daadaa$, $adaada$, $ddadda$, $aadaad$, $daddad$, $addadd$, $dddddd$. We apply L-system modeling complexity method to all sequences for $n = 1, 2, \ldots, 10$, all diagram are shown in Figure 2.8 to 2.17. For every n, we compute their mean complexity value of all enumerated sequences. Then, we have the best-fit curve for Case 2.3 and the grammar complexity of G_3 is $\alpha_3 = 1.404$.

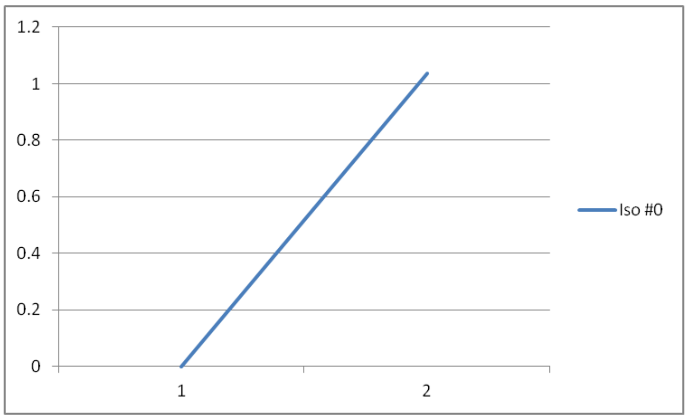


Figure 2.8: The complexity diagram of all sequences of Case 2.3 for $n = 1$ on isomorphic depth 0.

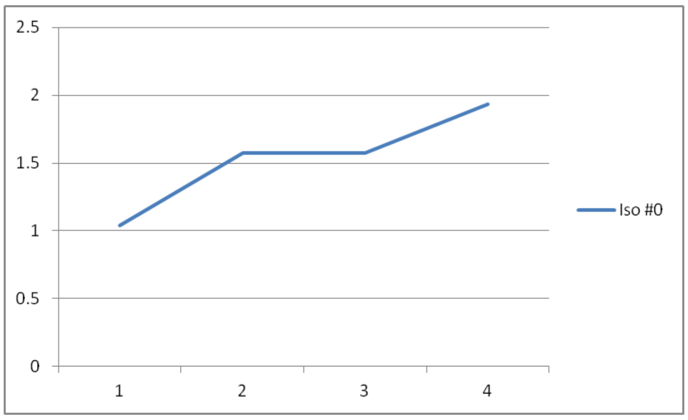


Figure 2.9: The complexity diagram of all sequences of Case 2.3 for $n = 2$ on isomorphic depth 0.

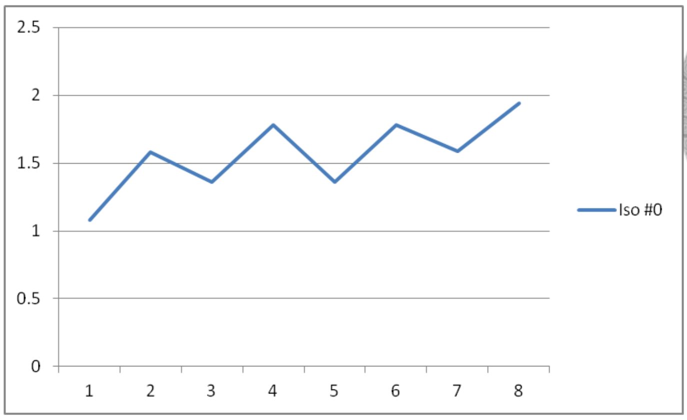


Figure 2.10: The complexity diagram of all sequences of Case 2.3 for $n = 3$ on isomorphic depth 0.

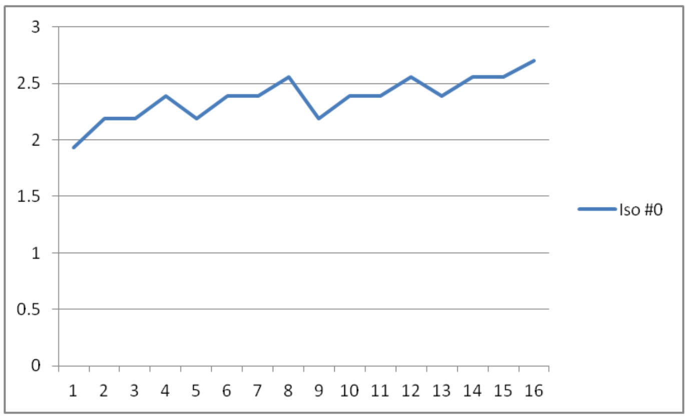


Figure 2.11: The complexity diagram of all sequences of Case 2.3 for $n = 4$ on isomorphic depth 0.

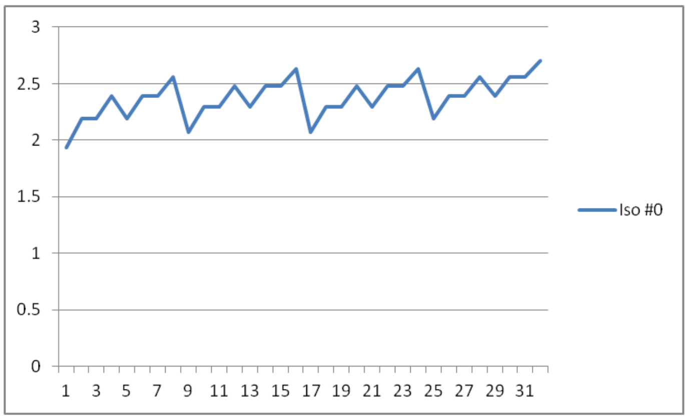


Figure 2.12: The complexity diagram of all sequences of Case 2.3 for $n = 5$ on isomorphic depth 0.

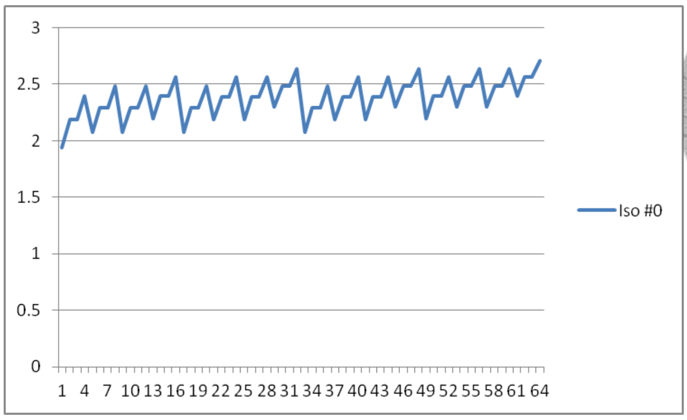


Figure 2.13: The complexity diagram of all sequences of Case 2.3 for $n = 6$ on isomorphic depth 0.

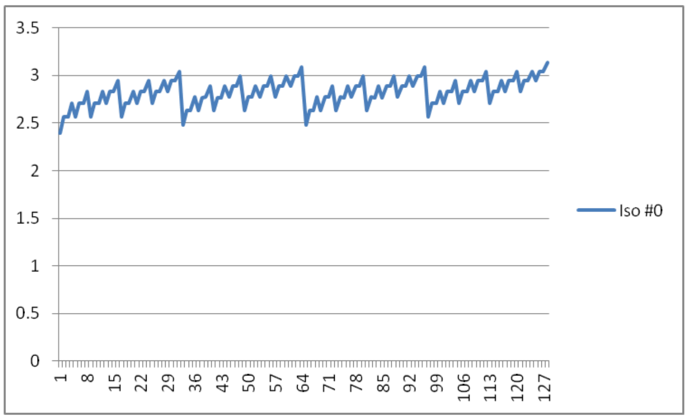


Figure 2.14: The complexity diagram of all sequences of Case 2.3 for $n = 7$ on isomorphic depth 0.

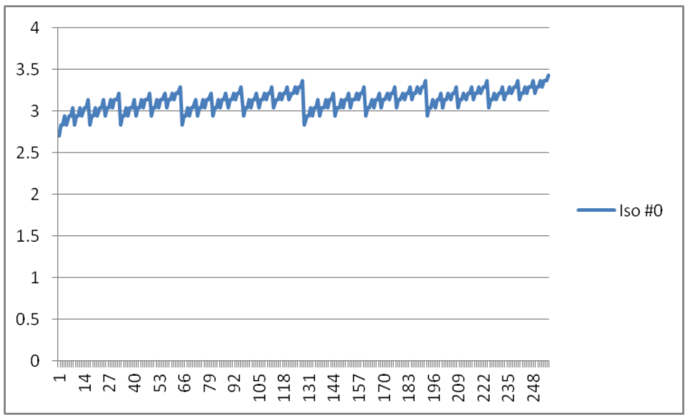


Figure 2.15: The complexity diagram of all sequences of Case 2.3 for $n = 8$ on isomorphic depth 0.

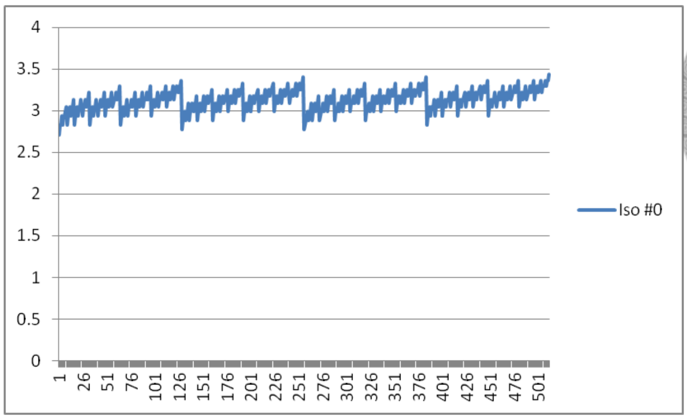


Figure 2.16: The complexity diagram of all sequences of Case 2.3 for $n = 9$ on isomorphic depth 0.

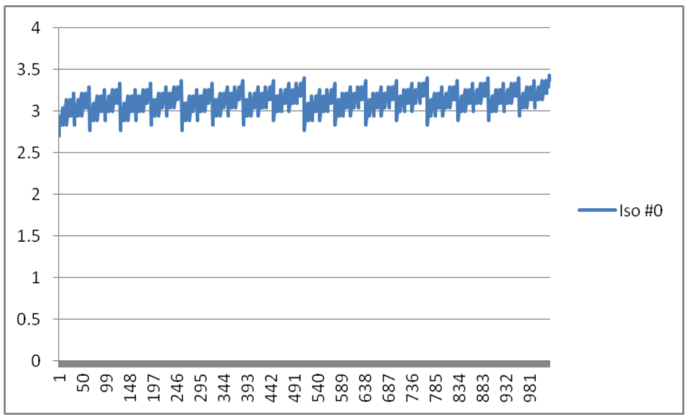


Figure 2.17: The complexity diagram of all sequences of Case 2.3 for $n = 10$ on isomorphic depth 0.

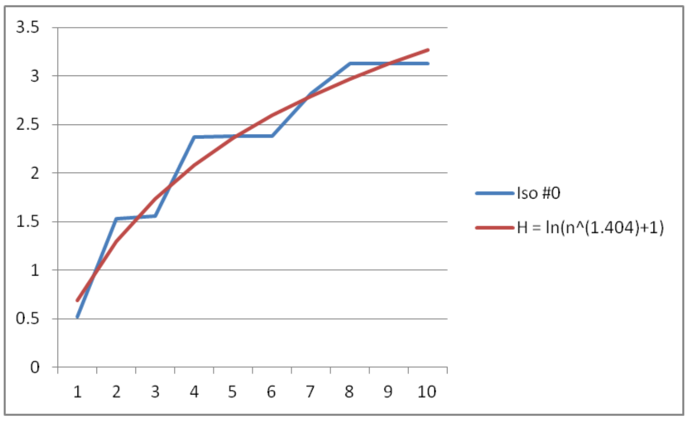


Figure 2.18: The complexity diagram of all sequences of Case 2.3 for $n = 1, 2, \ldots, 10$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.404} + 1$.

Case 2.4. *Given a context-sensitive grammar G_4 and the language generated by G_4 is*

$$L(G_4) = \{ww | w \in \{b, c\}^n, n > 0\}.$$

The only difference between of Case 2.3 and Case 2.4 is the words they generate. With different tree representations, the text tree combined from them is different too. Thus, the complexity derived from text tree is also dfferent. In this case, we list all sequences in $L(G_4)$ of length $n \times 2$, i.e., for $n = 3$ we list *bbbbbb*, *cbbcbb*, *bcbbcb*, *ccbccb*, *bbcbbc*, *cbccbc*, *bccbcc*, *cccccc*. We apply L-system modeling complexity method to all sequences for $n = 1, 2, \ldots, 10$, all diagram are shown in Figure 2.19 to 2.28. With similar processes, we have the best-fit curve for Case 2.4 and the grammar complexity of G_4 is $\alpha_4 = 1.05514$.

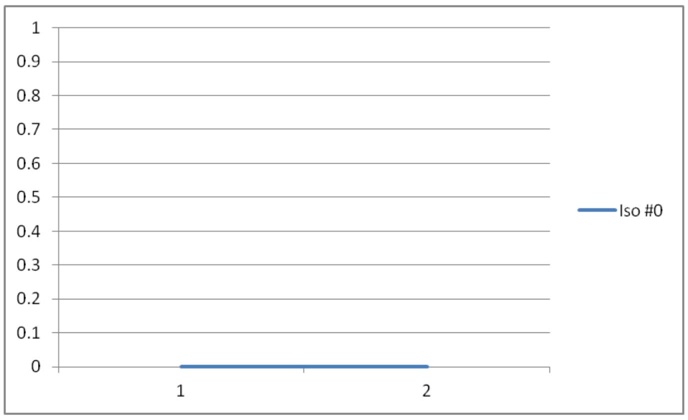


Figure 2.19: The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0.

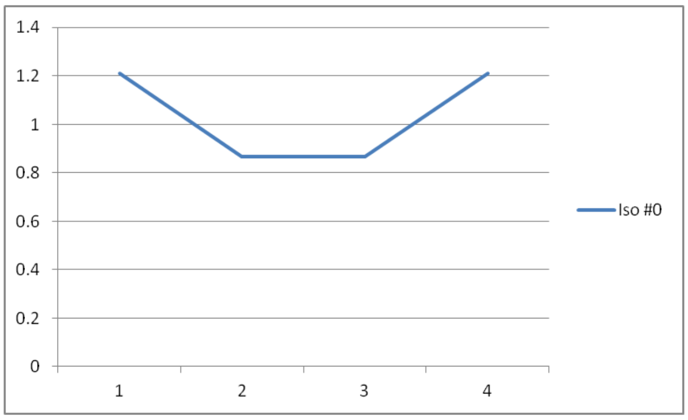


Figure 2.20: The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0.

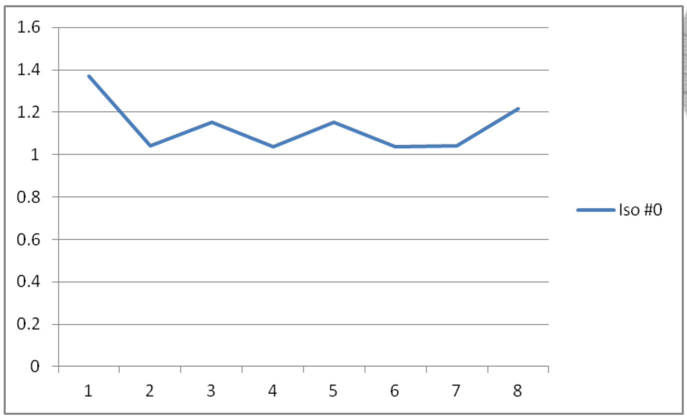


Figure 2.21: The complexity diagram of all sequences of Case 2.4 for $n = 3$ on isomorphic depth 0.

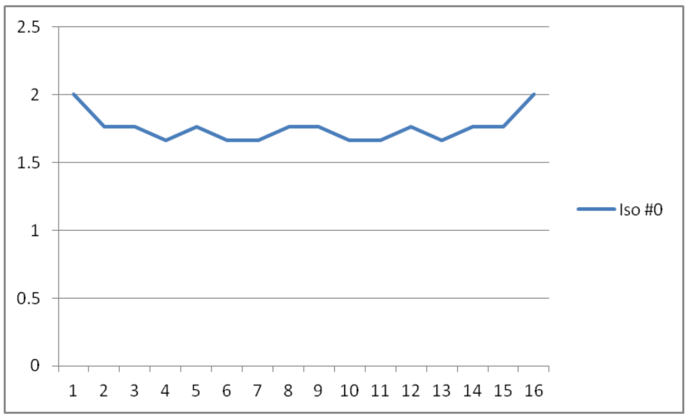


Figure 2.22: The complexity diagram of all sequences of Case 2.4 for $n = 4$ on isomorphic depth 0.

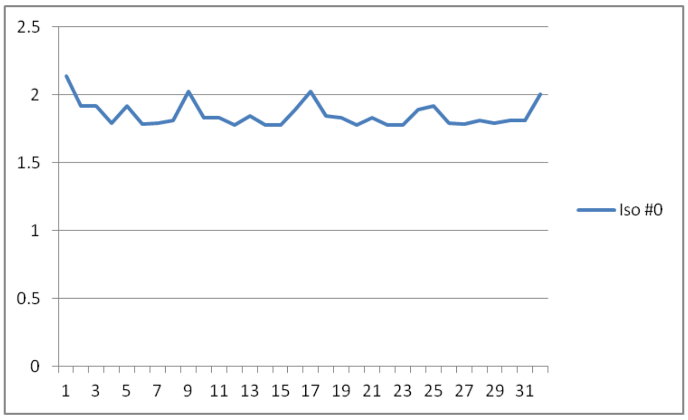


Figure 2.23: The complexity diagram of all sequences of Case 2.4 for $n = 5$ on isomorphic depth 0.

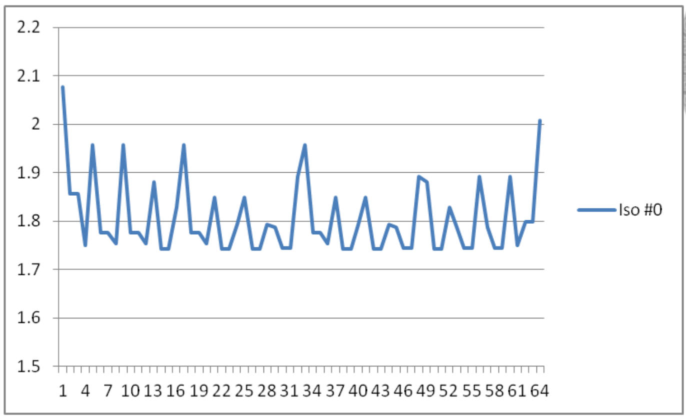


Figure 2.24: The complexity diagram of all sequences of Case 2.4 for $n = 6$ on isomorphic depth 0.

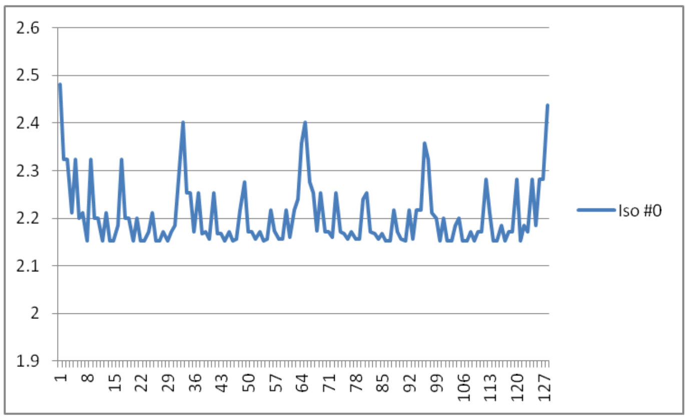


Figure 2.25: The complexity diagram of all sequences of Case 2.4 for $n = 7$ on isomorphic depth 0.

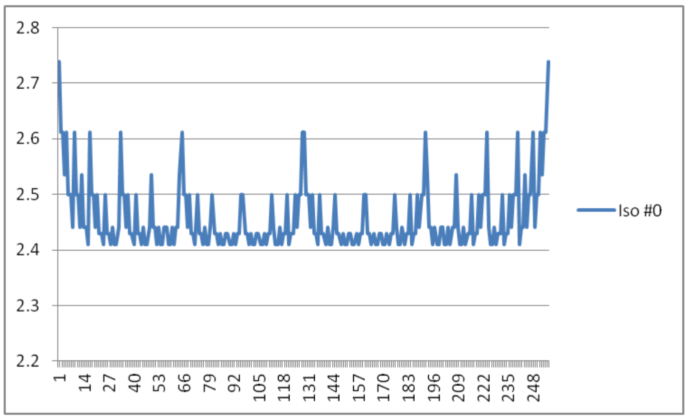


Figure 2.26: The complexity diagram of all sequences of Case 2.4 for $n = 8$ on isomorphic depth 0.

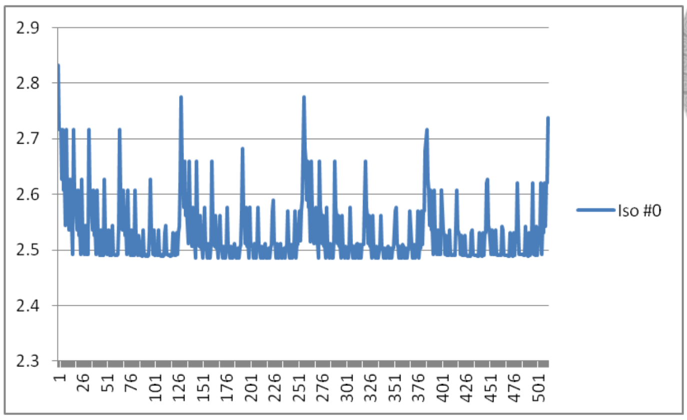


Figure 2.27: The complexity diagram of all sequences of Case 2.4 for $n = 9$ on isomorphic depth 0.

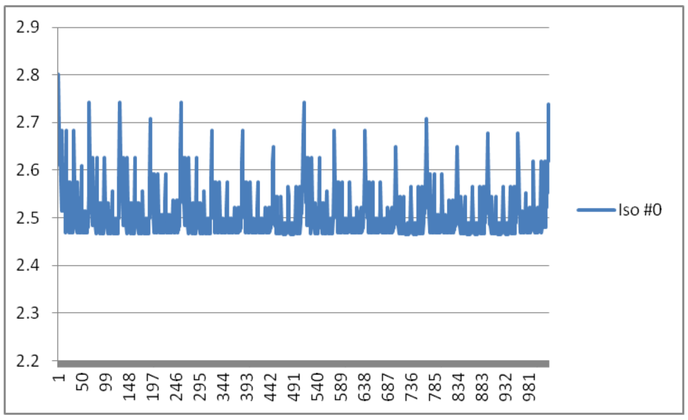


Figure 2.28: The complexity diagram of all sequences of Case 2.4 for $n = 10$ on isomorphic depth 0.

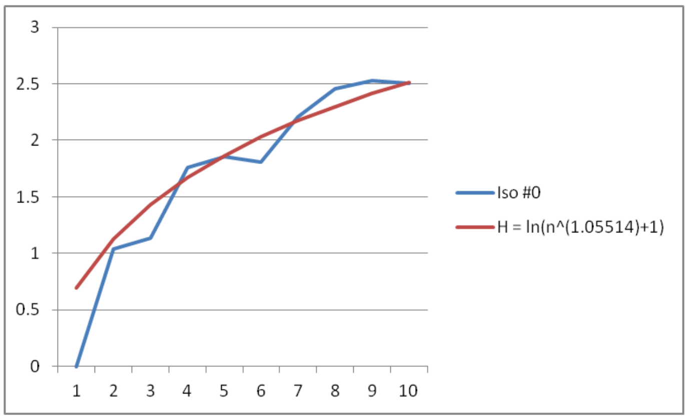


Figure 2.29: The complexity diagram of all sequences of Case 2.4 for $n = 1, 2, \ldots, 10$ on isomorphic depth 0 and the least square error curve of $H = \ln n^{1.05514} + 1$.

Chapter 3

Extension

In previous work, the text trees are binarily constructed. In fact, the modeling complexity method can be applied to ternary text trees cases. Here we take Case 2.1 for n = 3 as an example below.

With the similar processes but build the text tree ternarily, we have the text tree $T_1'(3)$ shown in Figure 3.1. Then, we list all the rewriting rules from $T_1'(3)$ into Table 3.1. Here we use "M" and "=" to denote the middle subtree rule. The classification is shown in Table 3.2.

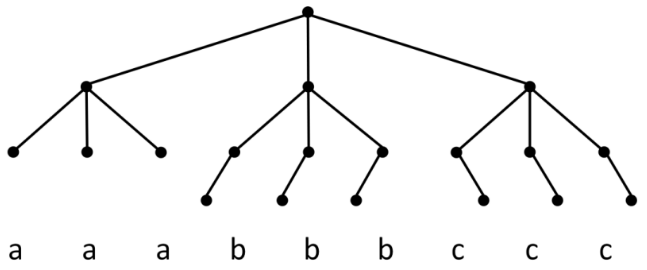


Figure 3.1: The ternary text tree of Case 2.1 for $n = 3$.

$P \to [-FT_L][=FT_M][+FT_R]$	
$T_L \to [-FT_{L_L}][=FT_{L_M}][+FT_{L_R}]$	$T_R \to [-FT_{R_L}][=FT_{R_M}][+FT_{R_R}]$
$T_M \to [-FT_{M_L}][=FT_{M_M}][+FT_{M_R}]$	$T_{R_L} \to [+FT_{R_{L_R}}]$
$T_{M_L} \to [-FT_{M_{L_L}}]$	$T_{R_M} \to [+FT_{R_{M_R}}]$
$T_{M_M} \to [-FT_{M_{M_L}}]$	$T_{R_R} \to [+FT_{R_{R_R}}]$
$T_{M_R} \to [-FT_{M_{R_L}}]$	

Table 3.1: Rewriting Rules of the ternary text tree in Figure 3.1.

Classification of Rules	Isomorphic Depth #0	Isomorphic Depth #1	Isomorphic Depth #2
Class #1	$(1)C_1 \to C_1C_1C_1$	$(1)C_1 \to C_2C_3C_4$	$(1)C_1 \to C_2C_3C_4$
	$(1)C_1 \to C_2C_2C_2$		
	$(1)C_1 \to C_3C_3C_3$		
	$(1)C_1 \to C_4C_4C_4$		
Class #2	$(3)C_2 \to C_4C_\phi C_\phi$	$(1)C_2 \to C_5C_5C_5$	$(1)C_2 \to C_5C_5C_5$
Class #3	$(3)C_3 \to C_\phi C_\phi C_4$	$(1)C_3 \to C_6C_6C_6$	$(1)C_3 \to C_6C_6C_6$
Class #4	$(9)C_4 \to C_\phi C_\phi C_\phi$	$(1)C_4 \to C_7C_7C_7$	$(1)C_4 \to C_7C_7C_7$
Class #5		$(1)C_5 \to C_7C_\phi C_\phi$	$(1)C_5 \to C_7C_\phi C_\phi$
Class #6		$(2)C_6 \to C_\phi C_\phi C_7$	$(2)C_6 \to C_\phi C_\phi C_7$
Class #7		$(2)C_7 \to C_\phi C_\phi C_\phi$	$(2)C_7 \to C_\phi C_\phi C_\phi$

Table 3.2: Classification Based on the Similarity of Rewriting Rules in Table 3.1.

By slightly modifications, the complexity formula can be applied to ternary cases:

$$V_i(z) = \frac{\sum_{p=1}^{n_i} n_{ip} z V_{a_{ip1}}(z) V_{a_{ip2}}(z) V_{a_{ip3}}(z)}{\sum_{q=1}^{n_i} n_{iq}}$$

For example, we compute the ternary modeling complexity value of sequence "$aaabbbccc$" on isomorphic depth 0 now. According to the definition, the given values for the class parameters are listed in Table 3.3.

	Classification of Rules	Isomorphic Depth #0
$(n = 7)$	Class #1	$(1)C_1 \to C_1\ C_1\ C_1$
	$(n_1 = 1)$	$n_{11}\quad a_{111}a_{112}a_{113}$
		$(1)C_1 \to C_2\ C_2\ C_2$
		$n_{12}\quad a_{121}a_{122}a_{123}$
		$(1)C_1 \to C_3\ C_3\ C_3$
		$n_{13}\quad a_{131}a_{132}a_{133}$
		$(1)C_1 \to C_4\ C_4\ C_4$
		$n_{14}\quad a_{141}a_{142}a_{143}$
	Class #2	$(3)C_2 \to C_4\ C_\phi\ C_\phi$
	$(n_2 = 1)$	$n_{21}\quad a_{211}a_{212}a_{213}$
	Class #3	$(3)C_3 \to C_\phi\ C_\phi\ C_4$
	$(n_3 = 1)$	$n_{31}\quad a_{311}a_{312}a_{313}$
	Class #4	$(9)C_4 \to C_\phi\ C_\phi\ C_\phi$
	$(n_4 = 1)$	$n_{41}\quad a_{411}a_{412}a_{413}$

Table 3.3: The values for the class parameters of Table 3.2, on isomorphic depth 0.

There are 4 classes, so we obtain the formulas for $V_1(z')$, $V_2(z')$, $V_3(z')$ and $V_4(z')$. They are:

$$
\begin{aligned}
V_\phi(z') &= 1 \\
V_4(z') &= 1 \\
V_3(z') &= \frac{\sum_{p=1}^{n_3} n_{3p} z' V_{a_{3p1}}(z') V_{a_{3p2}}(z') V_{a_{3p3}}(z')}{\sum_{q=1}^{n_3} n_{3q}} = \frac{z' \times (1 \times V_\phi(z') \times V_\phi(z') \times V_4(z'))}{1} = z' \\
V_2(z') &= \frac{\sum_{p=1}^{n_2} n_{2p} z' V_{a_{2p1}}(z') V_{a_{2p2}}(z') V_{a_{2p3}}(z')}{\sum_{q=1}^{n_2} n_{2q}} = \frac{z' \times (1 \times V_4(z') \times V_\phi(z') \times V_\phi(z'))}{1} = z' \\
V_1(z') &= \frac{\sum_{p=1}^{n_1} n_{1p} z' V_{a_{1p1}}(z') V_{a_{1p2}}(z') V_{a_{1p3}}(z')}{\sum_{q=1}^{n_1} n_{1q}} \\
&= \frac{z' \times (1 \times V_1(z')^3 + 1 \times V_2(z')^3 + 1 \times V_3(z')^3 + 1 \times V_4(z')^3)}{4} \\
&= \frac{z' V_1(z')^3 + 2z'^4 + z'}{4}
\end{aligned}
$$

Solving $V_1^m(z')$, $m = 1000$, the radius of convergence R, and complexity $K_0 = -\ln R$, can be obtained from this formula.

Also, the difficult tree causes forward-referencing rules in classification as well. Similar to binary text tree cases, the stoppers only appear on the path from root to the rightmost leaf. Generally, finding stoppers, are finding the nodes who have more children than its parent has. With this concept, we can classify the rewriting rules properly on arbitrary ternary text tree, or even on n-ary tree cases for arbitrary positive integer n.

Chapter 4

Summary

In this artical, we deal with arbitrary length context-sensitive languages and compute their L-system modeling complexity. Different from previous work, our input sequence is not divided into fixed length shorter sequences but whole. For any given grammar G, longer language $L(G)$ generated by G has larger modeling complexity value. That is because in longer sequences, the generating functions grow much faster than shorter ones. After computing modeling complexity values of many ns, we can use a single value α to represent grammar complexity by finding the best-fit curve $H = \ln n^{\alpha}$. Generating functions on high isomorphic depth never diverge, so we use isomorphic depth 0 for all cases. Also that is why modeling complexity results on higher isomorphic depth are smaller. For any sequnece, using different tree representations also derives different L-system modeling complexity values. There are many possible combinations between input sequences and tree representations to build an identical text tree. If combination of distinct input sequences and distinct tree presentations derives the same text tree, then they have the same L-system modeling complexity values. That is, for the tree representations shown in Figure 1.1, sequence "$aaaaaaaa$" and sequence "$dddd$" have the same text tree and complexity.

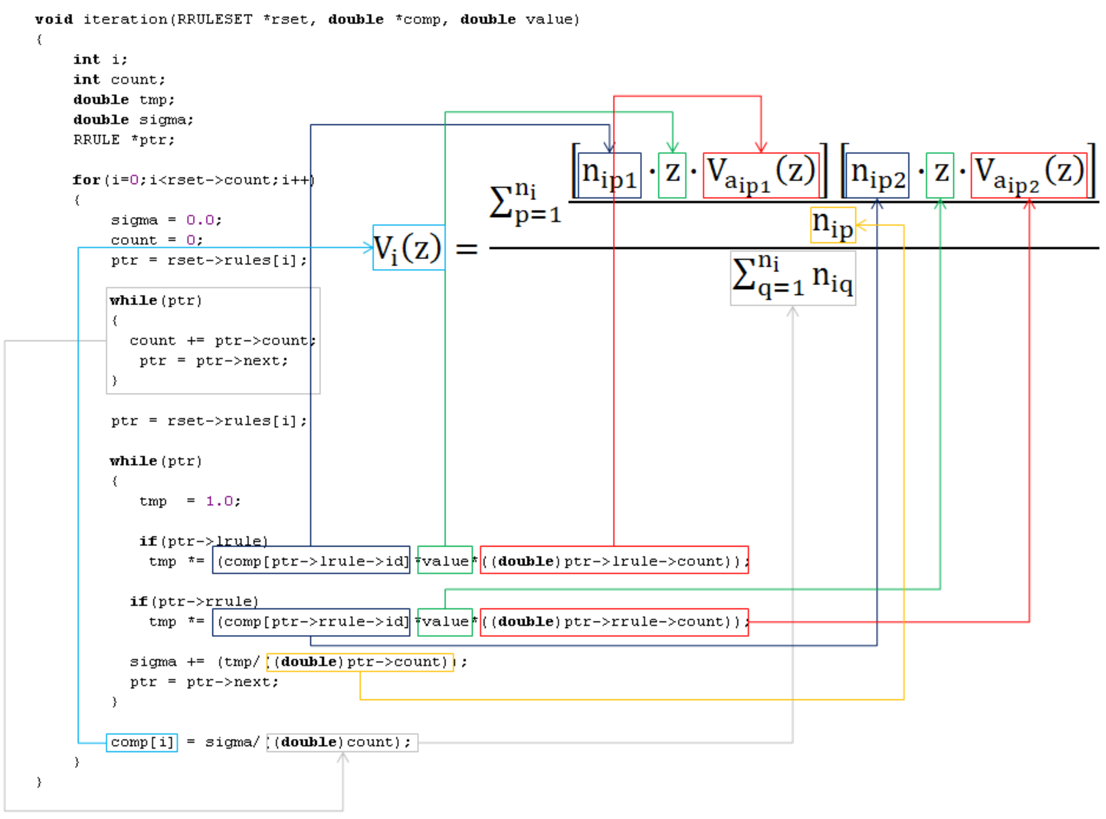
We may also use the extension formula in Chapter 3 to compute the L-system complexity of *penrose tiling* [10]. A Penrose tiling is a non-periodic tiling generated by an aperiodic set of prototiles. Penrose tilings are named after mathematician and physicist Roger Penrose who investigated these sets in the 1970s. L-system can implement penrose tiling properly and Malsys.cz [11] is a website to demostrate it.

Acknowledgements

本論文取材自國科會計畫 NSC 100-2221-E-002-234-MY3，題目解答為指導教授所授，全文為類神經網路實驗室學長與同學協力完成。

Appendix A

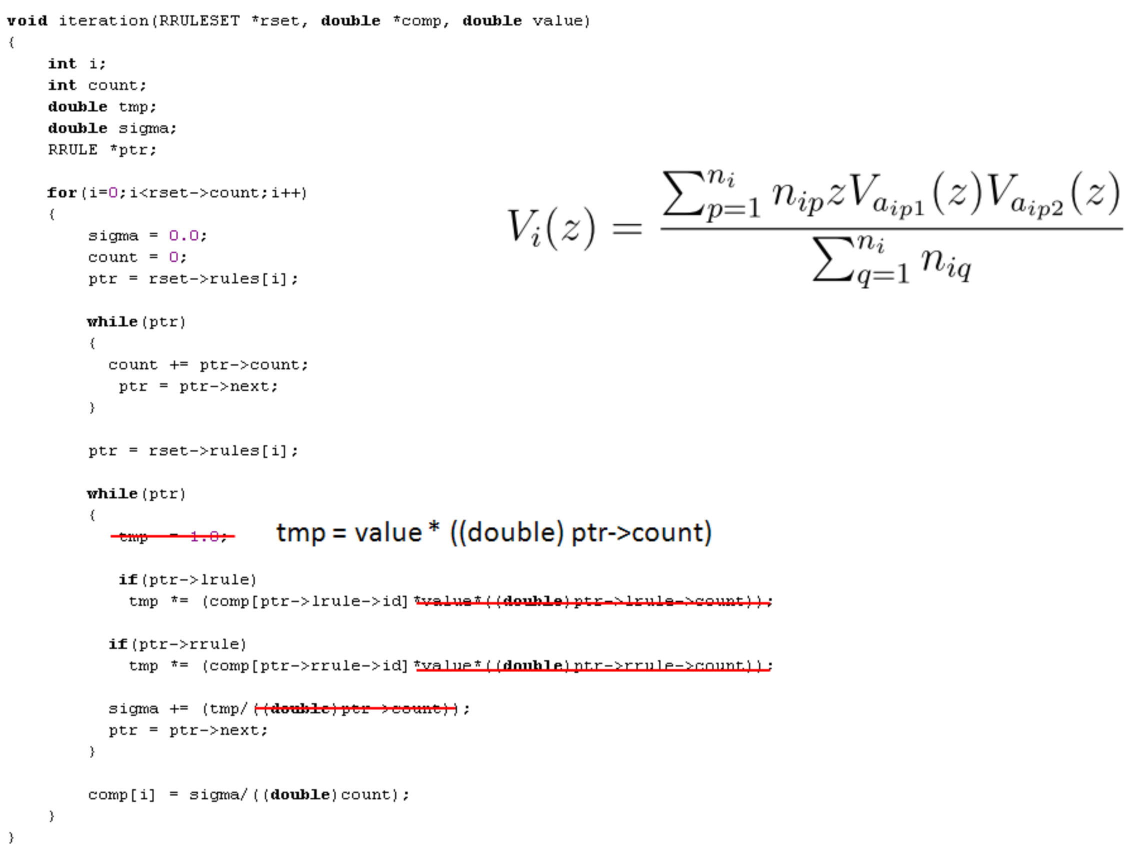
Code



```
void iteration(RRULESET *rset, double *comp, double value)
{
    int i;
    int count;
    double tmp;
    double sigma;
    RRULE *ptr;

    for(i=0;i<rset->count;i++)
    {
        sigma = 0.0;
        count = 0;
        ptr = rset->rules[i];

        while(ptr)
        {
          count += ptr->count;
           ptr = ptr->next;
        }

        ptr = rset->rules[i];

        while(ptr)
        {
           tmp  = 1.0;

           if(ptr->lrule)
            tmp *= (comp[ptr->lrule->id]*value*((double)ptr->lrule->count));

          if(ptr->rrule)
            tmp *= (comp[ptr->rrule->id]*value*((double)ptr->rrule->count));

          sigma += (tmp/((double)ptr->count));
          ptr = ptr->next;
        }

        comp[i] = sigma/((double)count);
    }
}
```

$$V_i(z) = \frac{\sum_{p=1}^{n_i} \frac{\left[n_{ip1} \cdot z \cdot V_{a_{ip1}}(z)\right]\left[n_{ip2} \cdot z \cdot V_{a_{ip2}}(z)\right]}{n_{ip}}}{\sum_{q=1}^{n_i} n_{iq}}$$

Figure A.1: Code of previous work.



```
void iteration(RRULESET *rset, double *comp, double value)
{
    int i;
    int count;
    double tmp;
    double sigma;
    RRULE *ptr;

    for(i=0;i<rset->count;i++)
    {
        sigma = 0.0;
        count = 0;
        ptr = rset->rules[i];

        while(ptr)
        {
          count += ptr->count;
           ptr = ptr->next;
        }

        ptr = rset->rules[i];

        while(ptr)
        {
           ~~tmp  = 1.0;~~     tmp = value * ((double) ptr->count)

           if(ptr->lrule)
            tmp *= (comp[ptr->lrule->id] ~~*value*((double)ptr->lrule->count));~~

          if(ptr->rrule)
            tmp *= (comp[ptr->rrule->id] ~~*value*((double)ptr->rrule->count));~~

          sigma += (tmp/ ~~((double)ptr->count)~~);
          ptr = ptr->next;
        }

        comp[i] = sigma/((double)count);
    }
}
```

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

Figure A.2: Paper code.

Bibliography

[1] F. P. Kaminger. The noncomputability of the channel capacity of context-sensitive languages. *Information and Control*, 17(175), 1970.

[2] P. Landweber. Decision problems on phrase structure grammars. *IEEE Trans. Electron. Comput.*, 13(354), 1964.

[3] Aristid Lindenmayer. Mathematical models for cellular interactions in development. Part I and II. *Journal of Theoretical Biology*, 18(3):280–315, 1968.

[4] Wikipedia. L-system — Wikipedia, the free encyclopedia, 2012. (http://en.wikipedia.org/wiki/L-system).

[5] Cheng-Yuan Liou, Tai-Hei Wu, and Chia-Ying Lee. Modeling complexity in musical rhythm. *Complexity*, 15(4):19–30, 2010.

[6] R. Badii and A. Politi. *Complexity: Hierarchical structures and scaling in physics*, volume 6. Cambridge University Press, 1999.

[7] Werner Kuich. On the entropy of context-free languages. *Information and Control*, 16(2):173–200, 1970.

[8] Cheng-Yuan Liou, Shen-Han Tseng, Wei-Chen Cheng, and Huai-Ying Tsai. Structural complexity of dna sequence. *Comp. Math. Methods in Medicine*, 2013.

[9] Cheng-Yuan Liou, Daw-Ran Liou, Alex A. Simak, and Bo-Shiang Huang. Syntactic sensitive complexity for symbol-free sequence. *IScIDE Bejing, LNCS*, 8261:15–22, 2013.

[10] Roger Penrose. Pentaplexity a class of non-periodic tilings of the plane. *The Mathematical Intelligencer*, 2:32–37, 1979.

[11] Marek Fiser. Marek's l-systems, 2014. (http://malsys.cz).