Doctoral Thesis

Analysis and Applications of the T-complexity

（T-complexityの解析と応用）

Kenji Hamano

Graduate School of Frontier Sciences

The University of Tokyo

Thesis Supervisor: Professor Hirosuke Yamamoto

December 2009

Abstract

T-codes are variable-length self-synchronizing codes introduced by Titchener in 1984. T-code codewords are constructed recursively from a finite alphabet using an algorithm called T-augmentation, resulting in excellent self-synchronization properties. An algorithm called T-decomposition parses a given sequence into a series of T-prefixes, and finds a T-code set in which the sequence is encoded to a longest codeword.

There are similarities and differences between T-decomposition and the conventional LZ78 incremental parsing. The LZ78 incremental parsing algorithm parses a given sequence into consecutive distinct subsequences (words) sequentially in such a way that each word consists of the longest matching word parsed previously and a literal symbol. Then, the LZ-complexity is defined as the number of words. By contrast, T-decomposition parses a given sequence into a series of T-prefixes, each of which consists of the recursive concatenation of the longest matching T-prefix parsed previously and a literal symbol, and it has to access the whole sequence every time it determines a T-prefix. Alike to the LZ-complexity, the T-complexity of a sequence is defined as the number of T-prefixes, however, the T-complexity of a particular sequence in general tends to be smaller than the LZ-complexity.

In the first part of the thesis, we deal with our contributions to the theory of T-codes. In order to realize a sequential determination of T-prefixes, we devise a new T-decomposition algorithm using forward parsing. Both the T-complexity profile obtained from the forward T-decomposition and the LZ-complexity profile can be derived in a unified way using a differential equation method. The method focuses on the increase of the average codeword length of a code tree. The obtained formulas are confirmed to coincide with those of previous studies.

The magnitude of the T-complexity of a given sequence s in general indicates the degree of randomness. However, there exist interesting sequences that have much larger T-complexities than any random sequences. We investigate the maximum T-complexity sequences and the maximum LZ-complexity sequences using various techniques including those of the test suite released by the National Institute of Standards and Technology (NIST) of the U.S. government, and find that the maximum T-complexity sequences are less random than the maximum LZ-complexity sequences.

In the second part of the thesis, we present our achievements in terms of application. We consider two applications—*data compression* and *randomness testing*.

First, we propose a new data compression scheme based on T-codes using a dictionary method such that all phrases added to a dictionary have a recursive structure similar to T-codes. Our data compression scheme can compress any of the files in the Calgary Corpus more efficiently than previous schemes based on T-codes and the UNIX *compress*, a variant of LZ78 (LZW).

Next, we introduce a randomness test based on the T-complexity. Recently, the Lempel-Ziv (LZ) randomness test based on the LZ-complexity was officially excluded from the NIST test suite. This is because the distribution of P-values for random sequences of length 10^6, the most common length used, is strictly discrete in the case of the LZ-complexity. Our test solves this problem because the T-complexity features an almost ideal uniform continuous distribution of P-values for random sequences of length 10^6. The proposed test outperforms the NIST LZ test, a modified LZ test proposed by Doganaksoy and Göloglu, and all other tests included in the NIST test suite, in terms of the detection of undesirable pseudo-random sequences generated by a multiplicative congruential generator (MCG) and non-random byte sequences $Y = Y_0, Y_1, Y_2, \cdots$, where Y_{3i} and Y_{3i+1} are random, but Y_{3i+2} is given by $Y_{3i} + Y_{3i+1} \bmod 2^8$.

Acknowledgments

I would like to thank my thesis supervisor, Professor Hirosuke Yamamoto, for accepting me as a doctoral student at the University of Tokyo and providing valuable discussions throughout this work. He gave me some important advice, and helped me to proceed with my research.

I appreciate the support and encouragement received from Dr. Ulrich Speidel (The University of Auckland). My work was inspired by his seminar regarding T-codes at the University of Tokyo in June 2007. When I participated in the 2008 International Symposium on Information Theory and its Applications (ISITA2008) held in Auckland, New Zealand, he kindly invited me to his office and explained his previous research on T-codes, especially the excellent relationships between T-code codewords and necklaces to me. Our commemorative photo at the Tamaki Campus will always remind me of my short and enjoyable research life as a doctoral student. When I participated in the 2009 IEEE International Symposium on Information Theory (ISIT2009), I was able to meet him again and discuss my research on T-codes with him. He gave me valuable feedback, and besides, introduced his current state of T-code research, which interested me.

Professor Toshinobu Kaneko (Tokyo University of Science) deserves thanks for his interest in my work. His comments on my research presented at a research talk hosted by the IEICE Technical Committee were a strong incentive for realizing an efficient data compression based on T-codes. His CRYPTREC technical report listing the problems of the NIST LZ test inspired me to develop a new randomness

test based on the T-complexity.

I am grateful to the executives of the Technical Research and Development Institute for the permission to concentrate on my research for a degree even though I was technically an employee. I also appreciate the help and support received from my colleagues. My special thank goes to Dr. Kazuyuki Joya, the chief of my laboratory at the time when I was a recruit, for his continuing advice and encouragement. I am also much obliged to Mr. Masaoki Ishikawa, the current chief of my laboratory. Occasional discussions with Dr. Kazuyuki Joya and Mr. Masaoki Ishikawa have greatly contributed to several research projects on randomness testing. I am grateful to Dr. Masaki Takeda and Dr. Fumio Sato, senior staff members of my laboratory, for their close guidance. My paper on randomness testing based on the linear complexity profile originated in discussions with Dr. Fumio Sato. I thank Mr. Akira Komori and Mr. Hideyuki Teranishi for making up for my absence from the office.

I am also indebted to Professor Masato Okada, Professor Tomoyuki Nishita, Associate Professor Ayumu Matani, and Associate Professor Noboru Kunihiro for their reading of the manuscript and comments on this work.

I thank Dr. Mitsugu Iwamoto, who was a classmate of mine in undergraduate days and a former member of Yamamoto Laboratory (Mathematical Information Laboratory #3), for his kind friendship throughout my doctoral studies. I am indebted to all members of the Yamamoto-Kunihiro Laboratory for my fulfilling research life.

Finally, I thank my family for their continuous support.

December 2009

Kenji Hamano

Contents

Abstract i

Acknowledgments iii

List of figures x

List of tables xii

1 Introduction 1

1.1 Overview of the Thesis 1

1.2 Kolmogorov Complexity 5

1.3 Computable Complexity Measures 6

1.3.1 LZ76-complexity 7

1.3.2 LZ-complexity 8

1.3.3 T-complexity 9

1.4 Data Compression and Parse Tree 10

1.5 Statistical Randomness Test 12

1.6 T-codes 14

1.7 LZ78 versus T-codes 21

1.8 Organization of the Thesis 23

1.9 Notation 26

2 Forward T-decomposition 28

2.1 Introduction 28
2.2 Forward T-decomposition Algorithm for Simple T-codes 29
2.3 Forward T-decomposition Algorithm for Generalized T-codes 33
2.4 Conclusions 37

3 Differential Equation Method for Derivation of the Formulas of the T-complexity and the LZ-complexity 39
3.1 Introduction 39
3.2 Previous Approach to the Derivation of the Maximum T-complexity Profile 40
3.3 Derivation of the T-complexity Profile 42
3.4 Derivation of the LZ-complexity Profile 50
3.5 Conclusions 58

4 Properties of the Maximum T-complexity Sequences 60
4.1 Introduction 60
4.2 Generation Algorithm of the Maximum T-complexity Sequences . . 61
4.3 Experiments 64
4.4 Conclusions 70

5 Application of the Forward T-decomposition to Data Compression 71
5.1 Introduction 71
5.2 LZ78 Family 72
5.3 Müller-Schimpfky Scheme 75
5.4 New Compression Scheme Based on T-codes 78
5.5 Experiments 84
5.6 Discussions 87
5.6.1 Comparison of Methods A, B, and C 87

5.6.2 Single-Pass Realization 88
5.6.3 Universality of the Proposed Scheme with Method C 89
5.7 Conclusions 92

6 Application of the T-complexity to Randomness Testing for Cryptography 94

6.1 Introduction 94
6.1.1 Importance of Randomness Testing in Cryptography 94
6.1.2 NIST Test Suite 96
6.1.3 Problems of the NIST LZ Test 101
6.2 Randomness Test Based on the T-complexity 103
6.3 Experiments 108
6.4 Conclusions 112

7 Conclusions 114

References 117

Publications 125

List of Figures

1.1 Example of a parse tree. 11

1.2 Updates of the incremental parse tree for "*abbbbababbaba*". 12

1.3 P-value 13

1.4 Intermediate T-code sets $\mathcal{S}_i,\ 0 \leq i \leq 4$. 16

1.5 $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$. 17

1.6 T-prefix matrix of 'obj2' included in the Calgary Corpus. 21

1.7 Relations between the LZ78 incremental parsing and the standard T-decomposition 22

1.8 Forward T-decomposition 22

1.9 Schematic flowchart of this thesis. 26

2.1 Trie growth for a sequence $s = 00111110011111000111000100$. . . . 31

2.2 Computation time of the T-complexity using Algorithm-B. 34

2.3 Trie growth for a sequence $s = 11000111101111111111111111$. 36

3.1 Illustration of code trees of simple T-code sets. 43

3.2 Maximum T-complexity profile and the graph of $n = \mathrm{li}(N \ln \#\mathcal{A})$ for the binary alphabet. 48

3.3 Graph of z_n for the T-complexity. 48

3.4 Graph of z_n for the T-complexity for the case of $p = 0.33$. 49

3.5 Graph of z_n for the T-complexity for the case of $p = 0.20$. 49

3.6 Empirical T-complexity profile and the graph of $n = \mathrm{li}((\ln \#\mathcal{A})H_T N)$ for the binary alphabet. 50
3.7 Illustration of LZ78 parse trees. 51
3.8 Comparison of Eq. (3.20) and our evaluation Eq. (3.19). 54
3.9 Graph of z_n for the LZ-complexity. 55
3.10 Graph of z_n for the LZ-complexity for the case of $p = 0.33$. 57
3.11 Graph of z_n for the LZ-complexity for the case of $p = 0.20$. 57
3.12 Empirical LZ-complexity profile and the graph computed from Eq. (3.21). 57
3.13 Graph of k for various p's. 58

4.1 Maximum T-complexity profile for the binary alphabet. 61
4.2 Pass ratios for the maximum T-complexity sequences. 65
4.3 Pass ratios for the maximum LZ-complexity sequences. 65
4.4 Spectrum of a maximum T-complexity sequence. 67
4.5 Autocorrelation function of a maximum T-complexity sequence. . . 67
4.6 Spectrum of a maximum LZ-complexity sequence. 68
4.7 Autocorrelation function of a maximum LZ-complexity sequence. . . 68
4.8 Empirical distributions of LZ-complexities of the maximum T-complexity sequences (solid line) and Marsaglia's random sequences (broken line). 69

5.1 Illustration of the Müller-Schimpfky scheme (I). 76
5.2 Illustration of the Müller-Schimpfky scheme (II). 76
5.3 Indexing leaf nodes in the code tree. 78
5.4 Distributions of ξ's and η's for 'progc' in the Calgary Corpus for Method A. 82
5.5 Distributions of ξ's and η's for 'progc' in the Calgary Corpus for Method C. 83

5.6 T-complexity (vertical axis) plotted against the number of parsed subsequences (horizontal axis). The broken line shows the result of linear regression of the data measurement values. 85
5.7 Dictionary size (vertical axis) plotted against the number of parsed subsequences (horizontal axis). The broken line shows the result of linear regression of the data measurement values. 86
5.8 Comparison of the dictionary size. 88
5.9 Comparison of $\#\xi$. 89

6.1 Empirical distribution of the LZ-complexity of sequences of length 10^6 (dots) and the normal distribution $\mathrm{N}(69588.2, 8.55788^2)$ (solid line). 104
6.2 Empirical distribution of the T-complexity of sequences of length 10^6 (dots) and the normal distribution $\mathrm{N}(38720.6, 58.2937^2)$ (solid line). 105
6.3 T-complexity profiles for $\psi = 0.0, 0.2, 0.4, 0.6, 0.8$. Wider broken lines correspond to larger ψ. 105
6.4 Magnified detail of Fig. 6.3. 106
6.5 Empirical distribution of η (solid line) and the theoretical distribution (broken line). 107
6.6 View of 10^4 triples generated from the MCG given by Eq. (6.2). . . 111

List of Tables

1.1 Character assignment to $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$. 18

1.2 Comparative table between LZ78 and T-codes. 23

2.1 Comparison of average time to compute the LZ-complexity and the T-complexity of a random sequence for various lengths. 33

3.1 Computation of k for the maximum T-complexity sequences. 45

4.1 Test Results of the DFT test and the Universal test for the maximum T-complexity sequences. 65

4.2 Test Results of the DFT test and the Universal test for the maximum LZ-complexity sequences. 66

5.1 Encoding steps in LZ78 for the sequence "$abracadabraabracadabra$". 73

5.2 Example of LZW for the sequence "$abracadabraabracadabra$". . . 74

5.3 Example of LZMW for the sequence "$alfeatsalfalfaalfeatsalfalfa$". 75

5.4 Compression ratios for the Calgary Corpus achieved by the UNIX *gzip*, *bzip2*, *compress*, the Müller-Schimpfky (M-S) scheme, and our scheme (A, B, and C) described in Section 5.4. 77

5.5 Example of Phase 1 with Method A for $s_1^7 = alfeats$. 81

5.6 Start-Step-Stop code with $start = 1, step = 2, stop = 7$. 82

5.7 Compressed file size [bytes] of 'alphabet.txt' in the case of the UNIX *gzip*, *bzip2*, *compress*, and our scheme (A, B, and C). 87

5.8 Compression ratios for the Calgary Corpus when LZW and LZMW are used in Phase 1 but the same coding is used in Phases 2 and 3. The compression ratios achieved by our scheme (A, B, and C) are shown again. 87

6.1 Characteristics of randomness tests included in the NIST test suite. 97

6.2 Breakdown of the 188 Statistical Tests 98

6.3 Probabilities of η. 107

6.4 Reject Ratios of the NIST test suite for MCG sequences when $\alpha = 0.01$. 111

6.5 Reject Ratios of the NIST test suite for sequences Y when $\alpha = 0.01$. 112

Chapter 1

Introduction

1.1 Overview of the Thesis

As our network society grows and grows, more and more information needs to be exchanged over the public net. In order to transmit information securely, we have to encrypt the transmitted messages by a reliable cipher system. Random or pseudo-random sequences play an important role in such cipher systems. For example, keystreams used in symmetric-key stream ciphers should not be distinguishable from truly random sequences in order to protect encrypted messages against eavesdroppers. In general, we use pseudo-random number generators in cryptosystems because a device to generate truly random sequences is expensive and it is often required that random sequences can be reproduced. Therefore, in order to establish a secure cryptosystem, we need a measure to evaluate how random a sequence is.

What is randomness? It is a mathematically and philosophically interesting deep question. The randomness of a sequence is often measured on the basis of statistical properties of the truly random sequences. However, from the perspective of information theory, random sequences can also be characterized as incompressible sequences. The ultimate form of this approach is Kolmogorov complexity [3, 36].

The Kolmogorov complexity of a sequence is defined as the length of the shortest computer program that can generate the sequence. If the shortest computer program is shorter than the length of the sequence, it is not random. Since compressed data with a decoding program can generate a sequence, its length gives an upper bound of Kolmogorov complexity. Hence, the complexity of a sequence is closely related to data compression. However, Kolmogorov complexity has been proved to be uncomputable in general [3].

In 1978, Lempel and Ziv [66] proposed a computable complexity called the LZ-complexity founded on the LZ78 universal compression scheme. Furthermore, a randomness test based on the LZ-complexity (LZ test) was included in the NIST test suite [42] released by the U.S. government. But, the LZ test was officially excluded from it in 2008 [43] because of a serious defect [29].

The NIST test suite is a package of randomness tests to evaluate random number generators and pseudo-random number generators for cryptographic applications. It has been used by various companies, organizations, and government agencies worldwide. The NIST test suite assumes that the P-value of a random sequence distributes uniformly in the range of 0 to 1. The LZ test cannot satisfy this assumption because the distribution of P-values for random sequences of length 10^6 is strictly discrete. If we do not pay attention to it, the type I error rate is unexpectedly increased, and hence we cannot evaluate (pseudo) random number sequences correctly. This is the reason why the LZ test was excluded from the NIST test suite. Since a randomness test based on a complexity measure is essential in the field of information security, the problem of the LZ test is expected to be solved. But, it is difficult to solve this problem whenever we use the LZ-complexity.

The T-complexity [55], which is also the computable complexity measure based on T-codes [53] proposed by Titchener, has similarities with the LZ-complexity. The LZ78 is the encoding scheme in which a given sequence is parsed into sub-

sequences (words) using the LZ78 incremental parsing in such a way that each word consists of the longest matching word parsed previously and a literal symbol. The LZ-complexity is defined as the number of words obtained by the LZ78 incremental parsing. By contrast, the T-complexity is defined as the number of subsequences obtained by T-decomposition, which is the parsing algorithm that parses a given sequence into subsequences (T-prefixes) in such a way that each subsequence consists of the recursive concatenation of the longest matching T-prefix parsed previously and a literal symbol. Because of these characteristics, it is expected that the T-complexity can detect the recursive structure of a sequence better than the LZ-complexity.

The LZ78 encoding and the LZ-complexity have been well studied by many researchers. On the other hand, although T-codes have several desirable and attractive properties, T-codes have received little attention from anyone but some researchers at the University of Auckland. Furthermore, T-codes or the T-complexity has the following defects.

- The LZ78 incremental parsing is sequential, while T-decomposition is not a sequential algorithm because it requires the whole sequence to start the algorithm.

- There are many efficient universal compression schemes derived from the LZ78 scheme, while no efficient data compression scheme has been devised from T-codes.

- The maximum T-complexity is not attained by truly random sequences, and hence there exist sequences that have larger T-complexity than those of truly random sequences.

From the above background, this thesis mainly deals with T-codes and the T-complexity to clarify the following.

- Sequential T-decomposition can be realized by an algorithm proposed in Chaper 2.

- The T-complexity profile of a sequence can be derived theoretically by a differential equation method proposed in Chapter 3.

- The distribution of T-complexity can be characterized for random sequences, and properties of the maximum T-complexity sequences can be clarified as shown in Chapter 4.

- An efficient universal data compression scheme can be constructed on the basis of T-codes, and it outperforms the UNIX *compress* as shown in Chapter 5.

From these results, we can expect that the T-complexity is a good measure to evaluate the randomness of a sequence. Actually, in Chapter 6, we propose a randomness test based on the T-complexity (T-complexity test) that can solve the problem of the NIST LZ test and can be used as a supplement to the NIST test suite. We also demonstrate the power of the T-complexity test by showing some experimental results for non-random sequences that cannot be detected well by the NIST test suite, but can be detected well by the T-complexity test. From this fact, the T-complexity test can contribute considerably to accurate evaluation of cryptosystems.

In the following of this chapter, we introduce some basic concepts about complexity, data compression, statistical test, and T-codes. Furthermore, in the last of this chapter, we describe the organization of this thesis and the notation used in this thesis.

1.2 Kolmogorov Complexity

Suppose we are interested in the amount of information in an individual finite object that can be represented in the form of a finite sequence, such as a DNA sequence or a written text. We want to measure it just as mass and energy.

According to Shannon's information theory [48], the amount of information in a particular message x from an ensemble of possible messages $\mathcal{E}$, which is communicated between a sender and a receiver over a channel, is defined as $I(x) \equiv -\log_2 q(x)$, where $q(x)$ is the probability that x is selected from the ensemble $\mathcal{E}$. $I(x)$ measures the statistical unexpectedness of x. In other words, it measures the amount of surprise contained in x when x is received. The entropy of $\mathcal{E}$ is defined as

$$H(\mathcal{E}) \equiv \sum_{x \in \mathcal{E}} q(x) I(x).$$

$H(\mathcal{E})$ is the average amount of information gained by observing the outcome from $\mathcal{E}$. This approach requires á priori knowledge of the probability distribution over the set of possible messages. Hence, the concept of information based on Shannon's information theory is a probabilistic notion. We cannot rely on this approach to measure the amount of information in an individual finite sequence.

Kolmogorov complexity [3, 36] deals with quantifying the amount of information in an individual finite sequence. In 1965, Kolmogorov defined the Kolmogorov complexity of a finite sequence s as the length of the shortest binary computer program that can generate s. It is known as the invariance theorem that Kolmogorov complexity is independent of the type of computer up to a constant.

Kolmogorov complexity can express the notion of randomness of an individual sequence. A sequence is considered non-random when its Kolmogorov complexity is significantly shorter than its literal representation. Hence, a random sequence must be an incompressible sequence. Although initial sequences of decimal digits of

π will pass empirical randomness tests, those sequences are not considered random in terms of Kolmogorov complexity since π can be calculated by a short program.

As an example, consider the following three binary sequences.

1. 0101010101010101010101010101010101010101010101010101010101010101

2. 1100110011001111111001110111100110010010000100010110010111111

3. 0110001100010100000101111000100011101111100011101100010101010101

The first sequence is obviously regular, but the second and the third sequences seem to be irregular. Assume that the third sequence is truly random. We want to claim that the first sequence is not random. However, when each bit is an i.i.d. (independent and identically distributed) random variable taking values on $\{0, 1\}$ with equal probability, the probability of the outcome of the first sequence is the same as those of the others. So, we cannot say that the first sequence is less probable than the others. In this case, we can use Kolmogorov complexity. The first sequence can be simply described as 'write 01 thirty times'. The second sequence seems to be irregular, but it is actually described as the binary expansion of $\sqrt{2} - 1$. So, we can claim that both the first and the second sequences are not random on the basis of Kolmogorov complexity.

1.3 Computable Complexity Measures

Unfortunately, Kolmogorov complexity is impracticable because Kolmogorov complexity of an arbitrary sequence is non-computable [3]. Instead, the following two groups of computable complexity measures are used to estimate Kolmogorov complexity.

- The size of a specific machine that can generate a given sequence.
- The number of steps in which a given sequence can be generated from a given alphabet according to a specific predetermined rule.

The former group includes the linear complexity and the maximum order complexity. The linear complexity of a sequence s is defined as the length of the shortest linear feedback shift register (LFSR) that can generate s, and it is efficiently computed by the Berlekamp-Massey algorithm [38]. The maximum order complexity of a sequence s is defined as the length of the shortest (not necessarily linear) feedback shift register (FSR) that can generate s, and it is efficiently calculated using a directed acyclic word graph (DAWG) [26]. These complexities have been commonly used to evaluate stream ciphers, in which (pseudo) random keystreams are generated. In this thesis, we focus on the latter group.

1.3.1 LZ76-complexity

In 1976, Lempel and Ziv proposed their first computable complexity measure [35], which is referred to as the LZ76-complexity hereafter. A famous data compression algorithm known as LZ77 [65] is based on the LZ76-complexity. The LZ76-complexity of a sequence is linked to *the gradual buildup of new patterns along the given sequence* and defined as the number of subsequences obtained from a particular parsing of the sequence described below.

Let $s = s_0 s_1 \cdots s_{N-1}$ be a sequence of length N, and let us denote a subsequence $s_i s_{i+1} \cdots s_j$ of s by $s(i, j)$. We say that $s(0, b)$ is *reproducible* from $s(0, a)$, $a < b < N$, if there exists an integer $m \leq a$ such that $s_{m+k} = s_{a+1+k}$ for all $0 \leq k \leq b - a - 1$. This reproducibility is denoted by $s(0, a) \rightarrow s(0, b)$. We say that $s(0, b)$ is *producible* from $s(0, a)$, $a < b < N$, if $s(0, a) \rightarrow s(0, b - 1)$. This producibility is denoted by $s(0, a) \Rightarrow s(0, b)$. A history of s is the parsing of s such that $s(0, h_0)s(h_0 + 1, h_1) \cdots s(h_{n-1} + 1, h_n)$, where $h_0 = 0$, $h_n = N - 1$, $h_{i-1} < h_i$, and $s(0, h_{i-1}) \Rightarrow s(0, h_i)$, $1 \leq i \leq n$. $s(h_{i-1} + 1, h_i)$, $i = 1, 2, \cdots, a$, are called *words*. A word $s(h_{i-1} + 1, h_i)$ and the corresponding production step $s(0, h_{i-1}) \Rightarrow s(0, h_i)$ are called *exhaustive* if $s(0, h_{i-1}+1), s(0, h_{i-1}+2), \cdots, s(0, h_i - 1)$ are reproducible from $s(0, h_{i-1})$ but $s(0, h_i)$ is not. A history is called exhaustive if all the words are

exhaustive with a possible exception of the last word. Every sequence has a unique exhaustive history. The LZ76-complexity of s is defined as the number of words in the exhaustive history of s. For example, the exhaustive history of $s = 0101110110$ is given by $0 \cdot 1 \cdot 011 \cdot 10110\cdot$, where successive words are separated by dots. In this case, the last word is also exhaustive, and so there is a dot at the end of the history. Thus, the LZ76-complexity of this sequence is 4. The time complexity for computing the LZ76-complexity of a sequence of length N is $\mathrm{O}(N^2)$ because of the exhaustive search of the patterns. This is a demerit of the LZ76-complexity. The above parsing is called the LZ76 parsing hereafter.

1.3.2 LZ-complexity

In 1978, Ziv and Lempel proposed another computable complexity measure [66], which is referred to as the LZ-complexity. A famous data compression algorithm known as LZ78 [66] is based on the LZ-complexity. The LZ-complexity of a sequence is defined as the number of subsequences obtained from the LZ78 incremental parsing of the sequence described below.

The LZ78 incremental parsing is sequential and parses a given sequence immediately after a prefix of the unparsed part of the sequence that differs from all preceeding words. The resultant parsing is represented as $s = s(n_0 + 1, n_1)s(n_1 + 1, n_2) \cdots s(n_m + 1, n_{m+1})$, where $n_0 = -1,\ n_{m+1} = N - 1$. The first m words $s(n_{j-1} + 1, n_j),\ 1 \leq j \leq m$, are all distinct and for all $j,\ 0 \leq j \leq m$, there exists $i < j$ such that $s(n_i + 1, n_{i+1}) = s(n_j + 1, n_{j+1} - 1)$, where $s(n_{-1} + 1, n_0)$ is the empty string. But, the last word $s(n_m + 1, n_{m+1})$ may not be distinct from the first m words. For example, the resultant parsing of $s = 0101110110$ is given by $0\cdot1\cdot01\cdot11\cdot011\cdot0$, where successive words are separated by dots. The LZ-complexity of this sequence is 6. The time complexity for computing the LZ-complexity of a sequence of length N is $\mathrm{O}(N \log N)$. While the LZ76 parsing searches the entire string occurred before, the LZ78 incremental parsing restricts the starting points

of its pattern searches to those of previously parsed words in order to reduce time complexity of the LZ76 parsing.

1.3.3 T-complexity

In 1984, Titchener introduced variable-length self-synchronizing codes called T-codes [53]. The codewords of a T-code set are constructed from a finite alphabet using a recursive hierarchical pattern copying algorithm called T-augmentation. An algorithm called T-decomposition parses a given sequence into a series of parameters for T-augmentation—a codeword called T-prefix and an integer called T-expansion parameter, and finds a T-code set in which the sequence is encoded to a longest codeword.

In 1998, Titchener proposed a new complexity measure called T-complexity [55]. The T-complexity[*1] of a sequence s is the number of T-augmentation steps required to represent s as a longest codeword in a T-code set. T-information is defined as the inverse logarithmic integral[*2] of the T-complexity, and T-entropy is defined as T-information divided by the sequence length. The normalized T-entropy has been shown to be closely related to the Kolmogorov-Sinai entropy of the logistic map [6]. Several applications of the T-complexity have been introduced, such as evaluation of the entropy of language texts [56], network event detection [7], and similarity detection [63]. The time complexity for computing the T-complexity of a sequence of length N is $\mathrm{O}(N \log N)$ and it ostensibly corresponds to $\mathrm{O}(N)$ in an implementation on a fixed word size computer [62]. Details of T-codes are given in Section 1.6.

[*1] Strict definition is given in Section 1.6.

[*2] The logarithmic integral function is defined as $\mathrm{li}(\eta) \equiv \int_0^\eta \frac{\mathrm{d}t}{\ln t}$.

1.4 Data Compression and Parse Tree

Data compression is the conversion of an input data stream into another shorter data stream. Data compression is called lossless when the original input stream can be completely reproduced from the compressed stream. Lossless data compression can be classified into entropy coding (statistical coding) and universal coding. The former uses probability models explicitly, but the latter does not. Furthermore, universal coding is broadly classified into dictionary methods and transforms. The former includes LZ77 [65], LZ78 [66], LZW [61], etc., and the latter includes block-sorting compression [1], grammar-based compression [31], etc.

A given long data sequence is usually parsed into subsequences and each subsequence is encoded into the corresponding codeword. Depending on the lengths of subsequences and codewords, codes can be classified into four types: fixed-to-fixed-length (FF) codes, fixed-to-variable-length (FV) codes, variable-to-fixed-length (VF) codes, and variable-to-variable-length (VV) codes. In the case of VF codes, a data sequence is parsed into subsequences with variable length, but the length of codewords is fixed. A VF encoder uses a parse tree to parse a data sequence. In a parse tree, each branch is labeled with one of source symbols, each leaf node is assigned to a codeword, and every internal node has r child nodes, where r is the alphabet size. Following the unparsed part of a data sequence, the VF encoder traverses the parse tree from the root to a leaf. When a leaf node is reached, a parsed subsequence is given by the path from the root to the leaf, and the codeword is given by the codeword assigned to the leaf. This process continues until all data sequence is encoded. As an example of VF parsing, consider a sequence over the alphabet $\{a, b\}$: "*bbbbbbbbbabbabbabbb*", and a parse tree shown in Fig. 1.1. Since this parse tree has four leaf nodes, each codeword can be represented by a 2-bit string. The sequence is parsed as $bbb \cdot bbb \cdot bbb \cdot ba \cdot bba \cdot bba \cdot bbb \cdot$, and encoded into $3, 3, 3, 1, 2, 2, 3$, which means 11 11 11 01 10 10 11.

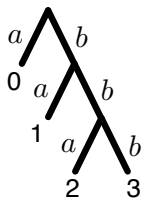


Figure 1.1. Example of a parse tree.

VF codes can be classified into static ones and adaptive ones. A VF code is called static when its parse tree is fixed, and called adaptive when its parse tree is dynamically changed as a data sequence is processed. The optimized static VF code (and parse tree) can be constructed by the Tunstall algorithm [60]. Furthermore, adaptive VF codes can be classified into two types: one is based on statistical coding and the other is based on non-statistical coding. Examples of the former and the latter are the adaptive Tunstall code [51] and LZ78 code, respectively. Usually, LZ78 falls into the category of dictionary methods, however, it can also be regarded as a non-statistical adaptive VF code because the LZ78 incremental parsing can be implemented by the incremental parse tree [64] as follows. Let r be the alphabet size. The LZ78 parse tree starts with a tree with a root and r child nodes. Following the unparsed part of a data sequence, the encoder traverses the current LZ78 parse tree from the root to a leaf. When a leaf is reached, a parsed subsequence is given by the path from the root to the leaf, and the subsequence is encoded into the codeword assigned to the leaf. Then, the LZ78 parse tree is updated by adding r child nodes to the leaf. This process continues until all data sequence is encoded. For example, consider a sequence over the alphabet $\{a, b\}$ ($r = 2$): "*abbbbababbaba*". The sequence can be parsed as $a \cdot b \cdot bb \cdot ba \cdot bab \cdot baba \cdot$ according to the LZ78 incremental parsing. For each parsing, the LZ78 parse tree is updated as shown in Fig. 1.2.

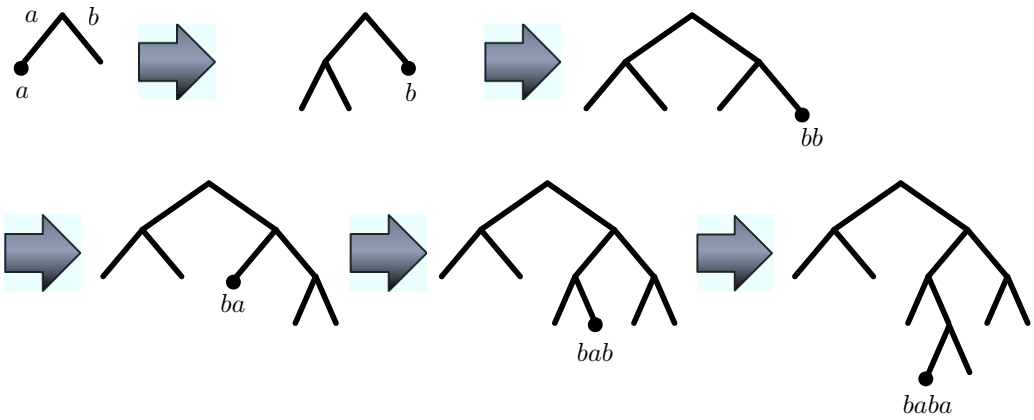


Figure 1.2. Updates of the incremental parse tree for "*abbbbababbbaba*".

1.5 Statistical Randomness Test

This section describes how a statistical randomness test measures the quality of a pseudo-random sequence.

A binary sequence $\boldsymbol{\varepsilon} = \varepsilon_0\varepsilon_1\cdots\varepsilon_{N-1}$ that satisfies the following ideal properties for $0 \le i \le N-1$ is simply called a random sequence of length N.

- $\Pr(\varepsilon_i = 0) = \Pr(\varepsilon_i = 1) = \frac{1}{2}$
- $\Pr(\varepsilon_i \mid \varepsilon_0, \ldots, \varepsilon_{i-1}) = \Pr(\varepsilon_i)$

Let H_0 be a statistical hypothesis that a given sequence is random. A statistical randomness test is a procedure for evaluating the hypothesis H_0 for a given sequence based on its statistical properties. The test has two errors. A Type I error occurs when the test rejects sequences that were in fact produced by a random bit generator. A Type II error occurs when the test accepts sequences even though they were not produced by a random bit generator. The significance level α is defined as the probability that H_0 is rejected when it is true, and hence α is equal to the Type I error probability. Let x be an observed value of a specified random variable X obtained by applying a statistical test to a random sample. X is called a test statistic. When X is expected to take on larger (smaller) values for non-random sequences, the P-value of x is defined as the probability that X is

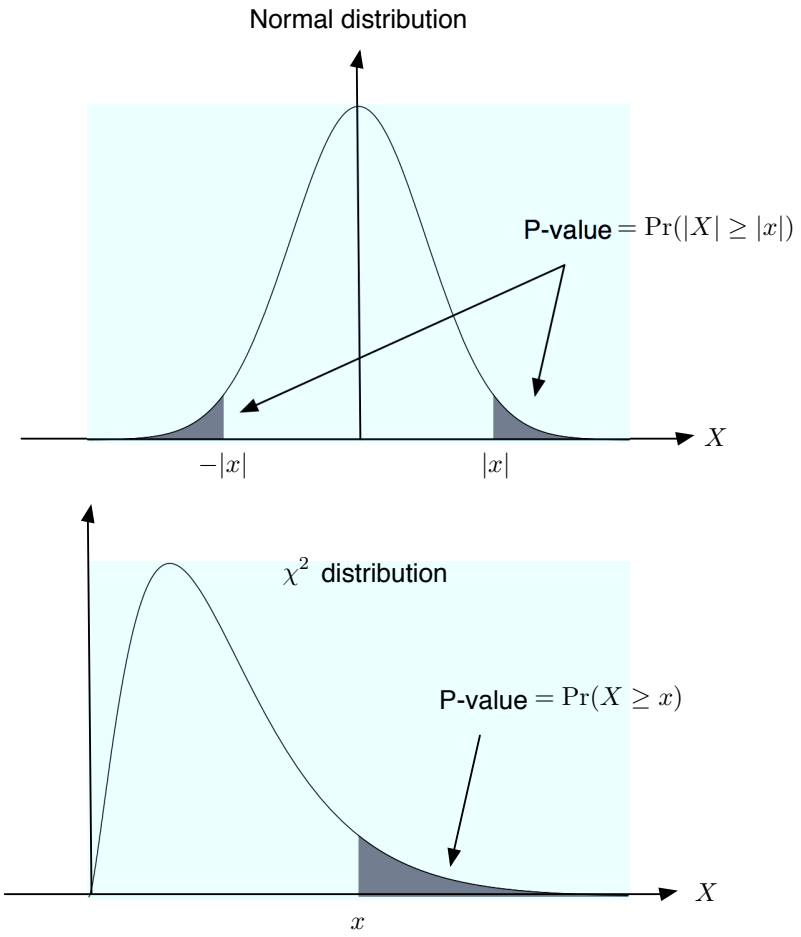


Figure 1.3. P-value

larger (smaller) than x in the case where H_0 is true. On the other hand, when X is expected to take on both larger and smaller values for non-random sequences, the P-value of x is defined as the probability that $|X|$ is larger than $|x|$ in the case where H_0 is true. The former and the latter cases correspond to the one tailed test and the two tailed test, respectively. Figure 1.3 shows P-values as gray area in the case where X follows normal distribution or χ^2 distribution. A small P-value gives evidence that a given sequence is non-random. Hence, we treat that if P-value $< \alpha$, then the given sequence fails the test and H_0 is rejected. Otherwise, the given sequence can be regarded as a random sequence and H_0 is accepted.

The P-value of a statistical randomness test distributes uniformly in the range of 0 to 1 if H_0 is true [33]. Let $F(X)$ be the cumulative distribution function of

a random variable X that is expected to take on larger values for non-random sequences. Let $G(Y)$ be the cumulative distribution function of a random variable Y where the observed value y of Y is the P-value of x. By the definition of y, $y = \Pr(X \geq x) = 1 - F(x)$. Hence, $\Pr(Y \leq y) = \Pr(1 - F(X) \leq 1 - F(x)) = \Pr(F(X) \geq F(x)) = \Pr(X \geq x) = 1 - F(x) = y$. Therefore, the P-value is uniformly distributed in the range of 0 to 1.

Famous packages of statistical randomness tests include:

- The NIST test suite described in NIST SP 800-22 [42, 43] released by the National Institute of Standards and Technology (NIST) of the U.S. government.
- The randomness tests listed by Knuth [33].
- The Diehard tests [37] developed by Marsaglia.

In most applications such as games, gambling, and computer simulations, random numbers are required to have the following representative properties.

- Long period
- Balance of symbols (uniform distribution)
- Low correlation

1.6 T-codes

T-codes are codes with variable-length codewords like Huffman codes [47]. Let $\mathcal{A}$ be a finite alphabet. Let uv represent the concatenation of two strings u and v, and let us denote k successive copies of u by u^k. A series of T-code sets $\mathcal{S}_i,\ i = 1, 2, \ldots,$ is constructed using the following recursive formula called T-augmentation.

$$\mathcal{S}_i = \bigcup_{j=0}^{k_i} \{p_i^j s \mid s \in \mathcal{S}_{i-1} \backslash \{p_i\}\} \cup \{p_i^{k_i+1}\}, \tag{1.1}$$

where $\mathcal{S}_0 = \mathcal{A}$ and a string p_i is selected from $\mathcal{S}_{i-1}$ and $k_i \in \mathbb{N} \equiv \{1, 2, 3, \cdots\}$. A string p_i and an integer k_i are called T-prefix and T-expansion parameter, respectively. $\mathcal{S}_i$ is called a T-code set at T-augmentation level i. $\mathcal{S}_i$ is also represented as $\mathcal{S}^{(k_1,k_2,\ldots,k_i)}_{(p_1,p_2,\ldots,p_i)}$.

Several kinds of T-augmentation are defined as follows [12].

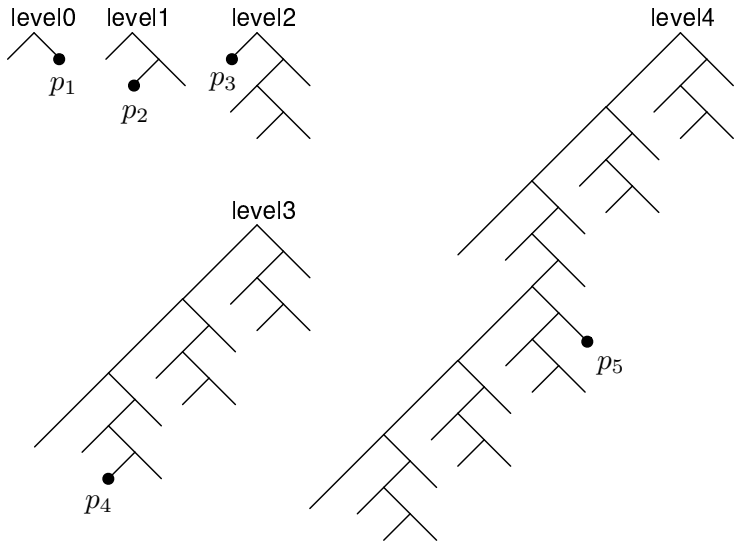
- T-augmentation is called *simple* when all T-expansion parameters are restricted to one, i.e., $k_1 = k_2 = \cdots = 1$.
- T-augmentation is called *strictly minimal* when each T-prefix p_i is chosen from one of the shortest codewords in the set.
- T-augmentation is called *systematic* if it is simple and strictly minimal.

T-code sets that can be generated entirely by simple T-augmentation are called simple T-code sets. Strictly minimal T-code sets and systematic T-code sets are defined in the same way. T-code sets generated according to Eq. (1.1) are also called *generalized* T-code sets [54] because simple T-code sets were first proposed by Titchener in 1984. Figure 1.4 shows code trees corresponding to T-code sets $\mathcal{S}_i$ for the case of $\mathcal{S}_5 = \mathcal{S}^{(1,1,2,1,1)}_{(1,10,0,001010,00101011)}$ with $\mathcal{A} = \{0, 1\}$. Left and right branches are labeled 0 and 1, respectively. Each node x in the code tree of $\mathcal{S}^{(k_1,k_2,\ldots,k_i)}_{(p_1,p_2,\ldots,p_i)}$ can be uniquely represented as

$$x = p_n^{k'_n} p_{n-1}^{k'_{n-1}} \cdots p_1^{k'_1} k'_0,$$

where $0 \le k'_i \le k_i$ for $i = 1, 2, \ldots, n$ and $k'_0 \in \mathcal{A}$.

T-codes have desirable properties for character synchronization to occur automatically on decoding. Let us explain this with an example. Figure 1.5 shows the code tree corresponding to a systematic T-code set $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$. $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$ can be used in variable-length encoding in such a way that shorter codewords are assigned to more frequent characters and longer codewords are assigned to less fre-



T-prefix T-expansion parameter

p_1	=	1	k_1	=	1
p_2	=	10	k_2	=	1
p_3	=	0	k_3	=	2
p_4	=	001010	k_4	=	1
p_5	=	00101011			

Figure 1.4. Intermediate T-code sets $\mathcal{S}_i,\ 0 \leq i \leq 4$.

quent characters as shown in Table 1.1. A string "KOLMOGOROV" is encoded to the following binary string.

$$010010000110110101000000111100000011001010011110111 \tag{1.2}$$

When a data loss or corruption occurs in a string that is encoded with variable length codes, the decoder can lose track of the correct codeword boundaries, and hence a large number of subsequent characters can also be corrupted. To overcome this problem, variable length codes usually have certain bit sequences occurring at the end of codewords. However, T-codes do not have such specific synchronizing bit sequences. In the case of T-codes, the synchronization information is spread throughout the codewords owing to the T-augmentation algorithm. When the first bit of the string (1.2) is missing, it is decoded to "(sp)(sp)OLMOGO

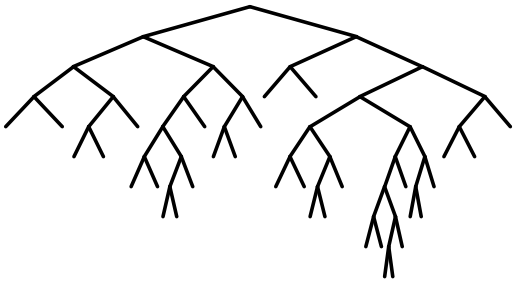


Figure 1.5. $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$.

ROV". Besides, when the first bit of the string (1.2) is inverted, it is decoded to "JOLMOGOROV". As can be noted from these examples, T-codes exhibit quick automatic synchronization. The self-synchronization mechanism is well understood [14].

T-decomposition [14], described below as Algorithm-A, is the inverse operation of T-augmentation, which can parse any sequence s into the following form

$$s = p_n^{k_n} p_{n-1}^{k_{n-1}} \cdots p_1^{k_1} k_0, \tag{1.3}$$

where $k_0 \in \mathcal{A}$ is a literal symbol and $k_i \in \mathbb{N},\ 1 \leq i \leq n$. The sequence s is related to one of the longest codewords in the T-code set $\mathcal{S}_n = \mathcal{S}^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)}$. Furthermore, each p_i can be uniquely represented as

$$p_i = p_{i-1}^{k_{i-1}^{(i)}} p_{i-2}^{k_{i-2}^{(i)}} \cdots p_1^{k_1^{(i)}} k_0^{(i)}, \tag{1.4}$$

where $k_0^{(i)} \in \mathcal{A}$ and $0 \leq k_j^{(i)} \leq k_j$ for $j > 0$. Algorithm-A can be implemented with $\mathrm{O}(N \log N)$ time and space complexities when the length of s is N, and it can be implemented with $\mathrm{O}(N)$ ostensibly on a fixed word size computer [62].

Algorithm-A

A1 Let s be a given sequence.

$i := 0$.

Table 1.1. Character assignment to $\mathcal{S}^{(1,1,1,1,1)}_{(0,1,00,01,11)}$.

Code	Character
100	<space>
101	E
0000	T
0001	A
0011	O
0101	I
0111	N
1111	S
00100	H
00101	R
01100	D
01101	L
11100	C
11101	U
010000	M
010001	W
010011	F
110000	G
110001	Y
110011	P
110101	B
110111	V
0100100	K
0100101	X
1100100	J
1100101	Q
1101100	Z
1101101	<.>
11010000	<,>
11010001	<?>
11010011	capsoff
110100100	capson
110100101	fill char

$\mathcal{S}_0 := \mathcal{A}$.

$s := sa$, where a is an arbitrary symbol in $\mathcal{A}$.

A2 Parse s into codewords in $\mathcal{S}_i$.

A3 If s becomes a single codeword in $\mathcal{S}_i$, i.e., $s \in \mathcal{S}_i$, then exit.

A4 Let p_{i+1} be the second to last codeword of s.

A5 If l adjacent copies of p_{i+1} are found immediately to the left of and including the second to last codeword, let $k_{i+1} := \ell$.

A6 $i := i + 1$. Generate $\mathcal{S}_i$ by Eq.(1.1). Go back to A2.

We show an example of how Algorithm-A processes $s = 00101000101$ for $\mathcal{A} = \{0, 1\}$. First, an arbitrary symbol $a \in \mathcal{A}$ is appended to s, i.e., $s := sa$. Then s is parsed into codewords in $\mathcal{S}_0$ $(= \mathcal{A})$. We obtain $s = 0.0.1.0.1.0.0.0.1.0.1.a$, where boundaries are indicated by dots. p_1 is given as the second to last codeword "1" and $k_1 = 1$. Then, $\mathcal{S}_1 = \{0, 10, 11\}$ is constructed from p_1, k_1, and $\mathcal{S}_0$. For $i = 1$, s is parsed into codewords in $\mathcal{S}_1$ as $s = 0.0.10.10.0.0.10.1a$, we obtain $p_2 = 10$, $k_2 = 1$, $\mathcal{S}_2 = \{0, 11, 100, 1010, 1011\}$. For $i = 2$, s is parsed into codewords in $\mathcal{S}_2$ as $s = 0.0.1010.0.0.101a$, and we obtain $p_3 = 0$. Since p_3 occurs twice adjacently including the second to last codeword, we have $k_3 = 2$, and

$$\begin{aligned} \mathcal{S}_3 \;=\; & \{11, 100, 1010, 1011, 011, 0100, 01010, 01011, \\ & 000, 0011, 00100, 001010, 001011\}. \end{aligned}$$

For $i = 3$, s is parsed into codewords in $\mathcal{S}_3$ as $s = 001010.00101a$, and we obtain $p_4 = 001010$ and $k_4 = 1$. For $i = 4$, s is parsed into codewords in $\mathcal{S}_4$ as $s = 00101000101a$. Finally s satisfies $s \in \mathcal{S}_4$, and the algorithm terminates. Then, we have $s = 00101000101a = p_4^{k_4} p_3^{k_3} p_2^{k_2} p_1^{k_1} a$, where s is one of the longest codewords in $\mathcal{S}_4$ (See the code tree of $\mathcal{S}_4$ in Fig. 1.4).

The T-complexity of s is defined as

$$t = \sum_{i=1}^{n} \log_2(k_i + 1),$$

when s is given by Eq. (1.3). It is worth noting that $\mathcal{S}_n$ has $2^t = \prod_{i=1}^{n}(k_i + 1)$ internal nodes. Hence, the T-complexity coincides with the number of bits required to address every internal node in its code tree where s corresponds to one of the longest codewords. For simple T-codes, the T-complexity t is equal to the number of T-prefixes, namely n. The T-information of s is defined as $\mathrm{li}^{-1}(t)$, and the T-entropy of s is defined as $\mathrm{li}^{-1}(t)/(|s| \ln(\#\mathcal{A}))$, where $|s|$ is the length of s.

On the basis of Eq. (1.4), p_i is represented as $(k_{i-1}^{(i)}, k_{i-2}^{(i)}, \ldots, k_1^{(i)}, k_0^{(i)})$. Furthermore, $(k_0^{(1)}, k_0^{(2)}, \ldots, k_0^{(n)})$ is called a literal vector, and $(k_{i-1}^{(i)}, k_{i-2}^{(i)}, \ldots, k_1^{(i)})$, $i = 2, \ldots, n$, can be represented as the following lower triangular matrix called a T-prefix matrix.

$$\begin{pmatrix} k_1^{(2)} & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ k_1^{(n-1)} & \cdots & k_{n-2}^{(n-1)} & 0 \\ k_1^{(n)} & \cdots & k_{n-2}^{(n)} & k_{n-1}^{(n)} \end{pmatrix}$$

Figure 1.6 shows the T-prefix matrix of 'obj2' included in the Calgary Corpus, which is obtained from a forward T-decomposition algorithm for simple T-codes[*3]. In the figure, white and black pixels represent "0" and "1", respectively. A column of a T-prefix matrix is called *occupied* if it contains at least one non-zero element. Otherwise, it is called *unoccupied* or *empty*. As can be noted from the figure, in general, T-prefix matrices are very sparse. A T-prefix matrix has lots of empty columns, and few non-zero elements are unevenly distributed in it. These characteristics of T-prefix matrices were used for data compression by C. Müller and

[*3] A forward T-decomposition algorithm for simple T-codes is described in Section 2.2.

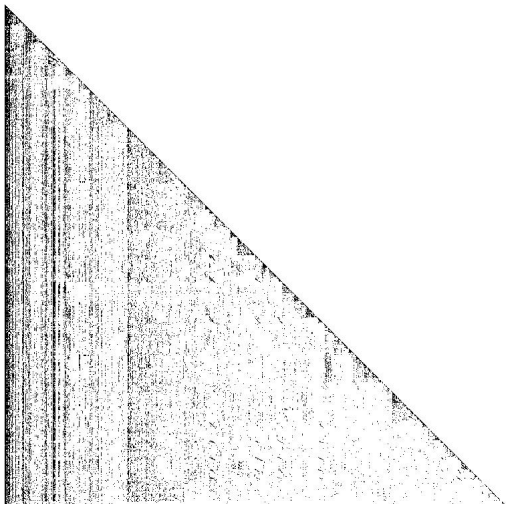


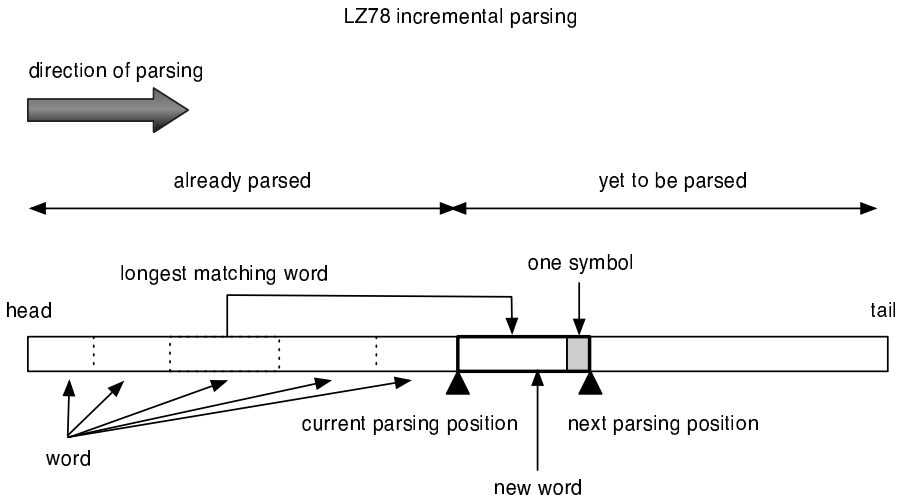
Figure 1.6. T-prefix matrix of 'obj2' included in the Calgary Corpus.

R. Schimpfky (See Section 5.3).

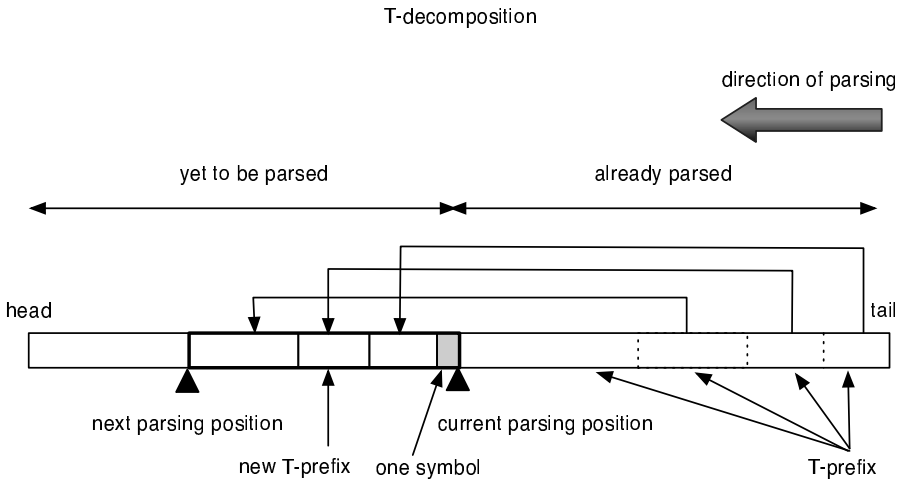
1.7 LZ78 versus T-codes

The LZ78 incremental parsing and the standard T-decomposition are related as shown in Fig. 1.7. The LZ78 incremental parsing parses a given sequence s into words in such a way that each word consists of the longest matching word parsed previously and a literal symbol. The T-decomposition parses s to T-prefixes, each of which consists of the recursive concatenation of the longest matching T-prefix parsed previously and a literal symbol. However, while the LZ78 incremental parsing is sequential, the standard T-decomposition has to access the whole sequence every time it determines a T-prefix, as described in Algorithm-A. On the other hand, a forward T-decomposition algorithm, as shown in Fig. 1.8, is suitable for on-line applications because of forward parsing.

The T-complexity of a particular sequence tends to be smaller than the respective LZ-complexity [58]. Therefore, data compression based on T-codes may be expected to achieve a better compression performance than the so-called LZ78 family. In addition, Titchener showed that the T-complexity is more sensitive to



(a) LZ78 incremental parsing



(b) Standard T-decomposition

Figure 1.7. Relations between the LZ78 incremental parsing and the standard T-decomposition

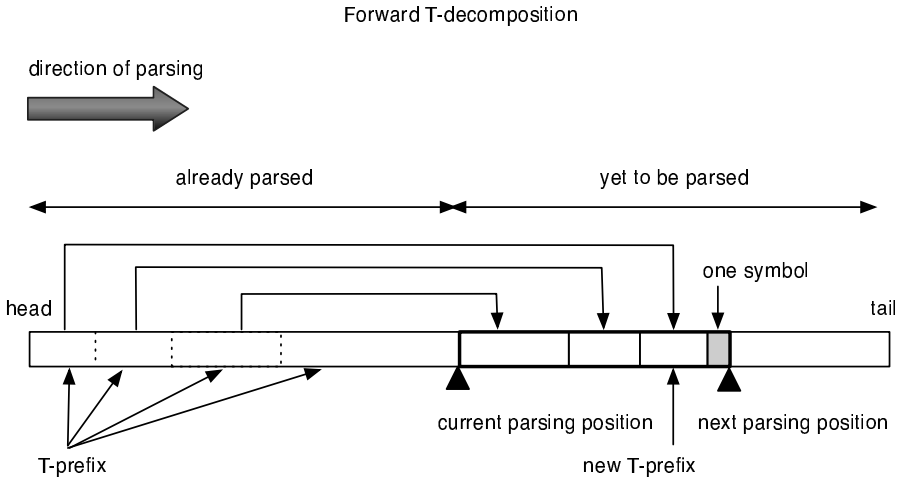


Figure 1.8. Forward T-decomposition

Table 1.2. Comparative table between LZ78 and T-codes.

	LZ78	T-codes
parsing method	The LZ78 incremental parsing, which is a sequential algorithm.	The known T-decomposition, which is not a sequential algorithm.
complexity measure	LZ-complexity, which is obtained by applying the LZ78 incremental parsing to a sequence.	T-complexity, which is obtained by applying the T-decomposition to a sequence.
data compression	so-called LZ78 family [47]	Nothing except for [41], which is not so efficient.
randomness test	The LZ test, which was excluded from the NIST test suite.	Nothing.

the variation between sources than the LZ-complexity [58]. Thus, a randomness test based on the T-complexity is likely to outperform a randomness test based on the LZ-complexity.

Table 1.2 shows a comparison between LZ78 and T-codes. We note from the table that compared with LZ78, T-codes has no counterparts for parsing method, efficient data compression scheme, and randomness test. In this thesis, we show that T-codes can have good counterparts that defeat the LZ78.

1.8 Organization of the Thesis

In Chapter 2, we first devise a forward T-decomposition algorithm, which can parse a given sequence and determine T-prefixes and T-expansion parameters sequentially. We propose two T-decomposition algorithms. One is for simple T-codes and the other is for generalized T-codes. It is experimentally confirmed that the computation time is about $O(N^{1.2})$, where N is the length of a random sequence. These sequential algorithms are suitable for on-line applications.

In Chapter 3, the T-complexity profile and the LZ-complexity profile are derived in a unified way using the same differential equation method. Here, the

complexity profile of a sequence s is defined as the sequence of $c_1, c_2, \cdots, c_n, \cdots$, where c_n is the complexity of the n-symbol prefix of s, i.e., $s(0, n-1)$. We focus on incremental quantities of average codeword length to formulate the differential equation. First, the maximum T-complexity profile, i.e., the T-complexity profile of a maximum T-complexity sequence obtained from the consecutive concatenation of T-prefixes of a systematic T-code set, is derived using the differential equation method. Then, we derive the T-complexity profile for random sequences on the basis of the derivation method of the maximum T-complexity profile. After that, the maximum LZ-complexity profile and the LZ-complexity profile for random sequences are derived following an argument similar to the case of the T-complexity profile. Our differential equation technique shows how the logarithmic integral function necessarily appears in the expression of the maximum T-complexity profile. Our expressions are confirmed to agree with the ones in previous studies.

Although the empirical mean of the T-complexity of binary random sequences of length 10^6 is 38720.6, we can easily generate the binary maximum T-complexity sequences of length 10^6 with T-complexity 56170. In Chapter 4, some properties of the maximum T-complexity sequences are investigated experimentally using various techniques including the NIST test suite [42, 43] and compared with those of the maximum LZ-complexity sequences. The experimental results show that the maximum T-complexity sequences are less random than the maximum LZ-complexity sequences, and some spikes are observed in the spectrum of a maximum T-complexity sequence, while no spike is observed in that of a maximum LZ-complexity sequence.

In Chapter 5, we propose a data compression based on a dictionary method such that all phrases added to a dictionary have a recursive structure of T-codes. We examine three dictionary updating rules. Since the T-complexity of a given sequence in general tends to be smaller than the respective LZ-complexity, our data compression scheme is expected to be more efficient than the UNIX *compress*, a

variant of LZ78. After we briefly summarize the known data compression scheme based on T-codes [41], we compare compression ratios of our proposed scheme, the known scheme based on T-codes, and the UNIX *compress*. As expected, our proposed scheme is more efficient than the known scheme based on T-codes and the UNIX *compress*, however, it is inferior to the UNIX *bzip2* and *gzip* on the whole.

Finally, a randomness test based on the T-complexity, called the T-complexity test in this thesis, is proposed in Chapter 6. A randomness test based on the LZ-complexity (LZ test) originally included in the NIST test suite was excluded from it in 2008 because the distribution of P-values for random sequences of length 10^6, the most common length used, is strictly discrete in the case of the LZ-complexity. It is undesirable because the NIST test suite assumes that the P-value of a random sequence distributes uniformly in the range of 0 to 1 and not paying attention to this assumption results in an unexpected increase in the Type I error rate. On the other hand, the T-complexity features an almost ideal continuous distribution of P-values for random sequences of length 10^6. Hence, the T-complexity test can solve the problem of the NIST LZ test. Moreover, we show that the T-complexity test outperforms the NIST LZ test, all other tests included in the NIST test suite, and a modified LZ test proposed by Doganaksoy and Göloglu [5] in terms of the detection of undesirable pseudo-random sequences generated by a multiplicative congruential generator (MCG) and non-random byte sequences $Y = Y_0, Y_1, Y_2, \cdots$, where Y_{3i} and Y_{3i+1} are random, but Y_{3i+2} is given by $Y_{3i} + Y_{3i+1} \bmod 2^8$.

The schematic flowchart of this thesis is shown in Fig. 1.9.

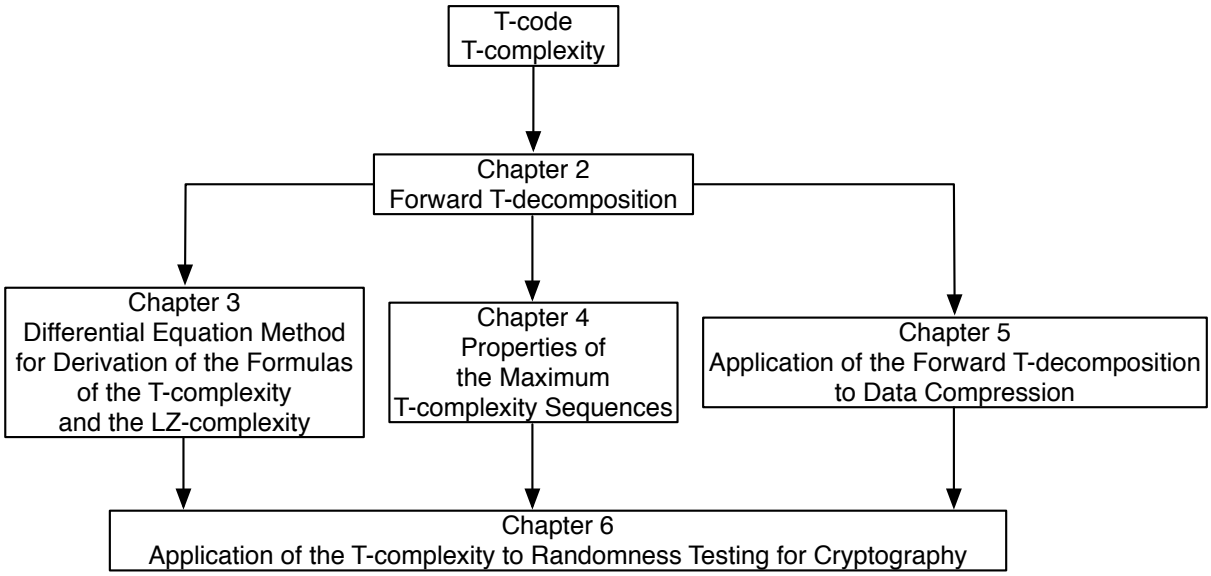


Figure 1.9. Schematic flowchart of this thesis.

1.9 Notation

This thesis uses the following notation.

- $\mathcal{A}$: A finite alphabet set.
- uv: The concatenation of two strings u and v.
- u^k: k successive copies of string u.
- $|u|$: The length of string u.
- λ: The null string.
- $\#\mathcal{B}$: The cardinality of a set $\mathcal{B}$.
- $\mathcal{S}_i$: A T-code set at T-augmentation level i.
- p_i: The i-th T-prefix.
- k_i: The i-th T-expansion parameter.
- s: A sequence fed into various algorithms described in this thesis.

- N: The length of a sequence s.
- $\mathrm{erf}(\cdot)$: The error function defined as $\mathrm{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_x^\infty \exp(-t^2)\mathrm{d}t$.
- $\mathrm{igamc}(\cdot,\cdot)$: The incomplete gamma function defined as $\mathrm{igamc}(a,x) \equiv \frac{\Gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_x^\infty \exp(-t)t^{a-1}\mathrm{d}t$, where $\mathrm{igamc}(a,0) = 1,\ \mathrm{igamc}(a,\infty) = 0$.
- $\mathrm{N}(\mu,\sigma^2)$: The normal distribution with mean μ and variance σ^2.
- $\mathrm{li}(\cdot)$: The logarithmic integral function defined as $\mathrm{li}(\eta) \equiv \int_0^\eta \frac{\mathrm{d}t}{\ln t}$.

Chapter 2

Forward T-decomposition

2.1 Introduction

The standard T-decomposition parses a given sequence into T-prefixes, each of which consists of the recursive concatenation of the longest matching T-prefix parsed previously and a literal symbol [14]. However it has to access the whole sequence every time it determines a T-prefix as shown in Algorithm-A (See Section 1.6). On the other hand, the LZ78 incremental parsing sequentially parses a given sequence into distinct words in such a way that each word consists of the longest matching word parsed previously and a literal symbol.

In this chapter, we propose several new T-decomposition algorithms for simple T-codes and generalized T-codes. These algorithms parse a given sequence s into $p_1^{k_1} p_2^{k_2} \cdots p_n^{k_n}$, where $k_i \in \{0, 1\}$ for simple T-codes, however, $k_i \in \mathbb{N}$ for generalized T-codes. Alike to the standard T-decomposition, each T-prefix p_i can be uniquely represented as Eq. (1.4). The proposed algorithms enable the sequential parsing of a given sequence s and can be used in on-line applications.

2.2 Forward T-decomposition Algorithm for Simple T-codes

In this section, we concentrate on simple T-code sets. The case of generalized T-code sets is treated in Section 2.3. Let $s = s_1 s_2 \cdots s_N$ be a sequence of length N and let $s_i^j = s_i s_{i+1} \cdots s_j$ be a subsequence of s. As an example, let us consider the case of a sequence $s = s_1^{26} = 00111110011111000111000100$ being parsed as $p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8$, where each p_i is given as follows.

$$
\begin{aligned}
p_1 &= 0 \\
p_2 &= p_1 1 \quad (= 01) \\
p_3 &= 1 \\
p_4 &= p_3 1 \quad (= 11) \\
p_5 &= p_3 p_1 0 \quad (= 100) \\
p_6 &= p_4 p_3 1 \quad (= 1111) \\
p_7 &= p_5 p_2 1 \quad (= 100011) \\
p_8 &= p_5 p_2 p_1 0 \quad (= 1000100)
\end{aligned}
$$

Note that these p_i's satisfy Eq. (1.4) with $k_j^{(i)} \in \{0, 1\}$, and each p_i can be obtained by using tries[*1] $t_0, t_1, t_2, \ldots, t_{i-1}$ shown in Fig. 2.1. Each trie t_i is constructed from $\{p_1, p_2, \ldots, p_i\}$ such that $p_j, 1 \leq j \leq i$, corresponds to a path from the root to node j. Nodes with index 0 do not correspond to any p_j, and the root node has no index.

Assume that $s_1^{13} = 0011111001111$ is parsed as $p_1 p_2 p_3 p_4 p_5 p_6$ and the tries t_j, $0 \leq j \leq 6$, have already been constructed from $\{p_1, p_2, \cdots, p_6\}$ as shown in Fig. 2.1. Then, the T-prefix p_7 is obtained from the previous tries as follows. First, we trace trie t_6 from the root to a leaf node following the remaining sequence of s, $s_{14}^{26} = 100011 \cdots$. Since we reach leaf node 5 in t_6, we can infer that the first part of p_7 consists of p_5, i.e., $p_7 = p_5 \cdots$. Since p_5 is parsed, the second p_j must

[*1] A trie is an ordered-multiway-tree data structure used in computer science [34].

satisfy $j \leq 4$. Hence, we next use trie t_4. Again, we trace trie t_4 from the root to a leaf node following the remaining sequence of s, $s_{17}^{26} = 011\cdots$. Because we end up at leaf node 2 in t_4, we can infer that the second part of p_7 is p_2, i.e., $p_7 = p_5 p_2 \cdots$. Since p_2 is parsed, we move on to trie t_1. In trie t_1, we cannot move from the root when we follow the remaining sequence of s, $s_{19}^{26} = 1\cdots$. In this case, the next symbol of the remaining sequence, "1", becomes the literal symbol of p_7, and p_7 is given as $p_5 p_2 1$.

Trie t_7 can be created by adding p_7 into trie t_6. Similarly, p_8 is obtained as follows. Following the remaining sequence of s, $s_{20}^{26} = 1000100$, we trace trie t_7 from the root toward a leaf node. But, in this case, the tracing ends at a node with index 0 instead of a leaf node. This means that s_{20}^{26} $(= 1000100)$ does not coincide with any p_j and hence it must be parsed with shorter p_j. So, we move backwards to the nearest node with a positive index, and we find that $p_8 = p_5 \cdots$. We next use trie t_4 since p_5 is parsed. After similar iterations, p_8 is given as $p_5 p_2 p_1 0$.

Sometimes the last T-prefix p_n does not end with a literal symbol. For instance, if s_1^{19} is the same as the previous example and s ends with $s_{20}^{22} = 100$, p_8 is given as $p_8 = p_5$. In this case, similarly to the above case, we move backwards to the nearest node with a positive index. After similar iterations, p_8 can be parsed as $p_8 = p_5 = p_3 p_1 0$.

In the above exapmle, we assumed for simplicity that all tries t_i are constructed separately. But, we note that trie t_i can simulate any t_j for $0 \leq j < i$ by considering only the nodes with index l satisfying $l \leq j$ in the trie t_i. Hence, it is sufficient that only the latest trie is memorized.

The above scheme can be described formally as the following Algorithm-B.

Algorithm-B (Forward T-decomposition Algorithm for Simple T-codes)

B1 (Initialization)

Let s be a given sequence.

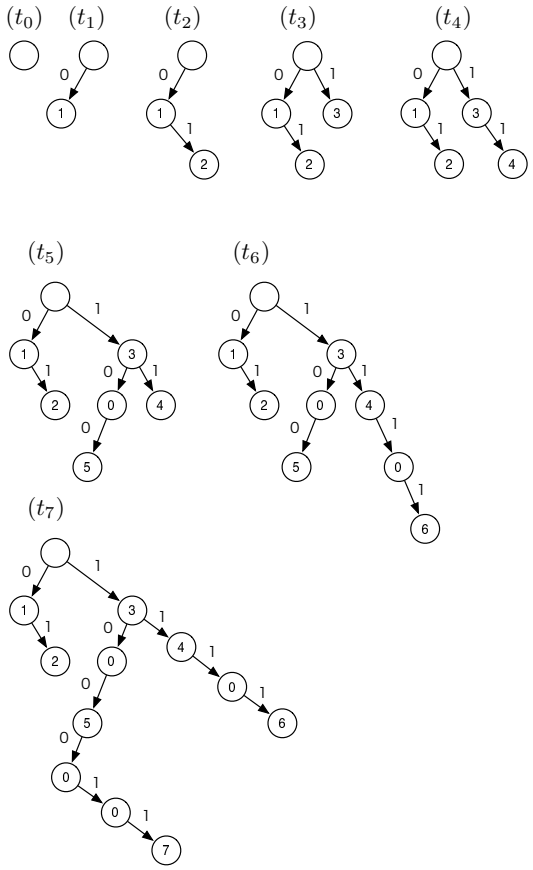


Figure 2.1. Trie growth for a sequence $s = 00111110011111000111000100$.

$i := 1$.

Create t_0.

B2 $p_i := \lambda$. $\nu := i - 1$.

B3 Following s, trace a path from the root toward a leaf node in trie ${t_\nu}^{*2}$ as far as possible. Let v represent the farthest node that we can reach.

- If v is a node with a positive index and s is not exhausted, then go to B4.
- If v is a node with index 0 or s is exhausted, then go to B5.
- If v is the root, i.e., we cannot move from the root, then go to B6.

B4 Let j be the index of node v.

[*2] Trie t_ν is simulated by considering only the nodes with index l satisfying $l \leq \nu$ in trie t_{i-1}.

$p_i := p_i p_j$.

$\nu := j - 1$.

Remove p_j from the head of s.

Go back to B3.

B5 If there exists no node with a positive index between node v and the root, then go to B6. Otherwise, move backwards from node v toward the root in trie t_ν. Let $\hat{v}$ be the first node with a positive index that we find when moving backwards. Let v represent the node $\hat{v}$, and go back to B4.

B6 $p_i := p_i \omega$, where ω is the first symbol of s.

Output p_i.

Remove ω from the head of s.

If $s = \lambda$, exit.

Otherwise, update trie t_{i-1} to t_i by adding p_i into t_{i-1}.

$i := i + 1$.

Go back to B2.

Table 2.1 shows average time of ten trials necessary to compute the LZ-complexity and the T-complexity (Algorithms A and B) for random sequences with 3 GHz CPU. Algorithm-A is implemented on the basis of [62] and its computation time is about $O(N)$ for a random N-bit sequence. On the other hand, we note from Table 2.1 and Fig. 2.2 that Algorithm-B has about $O(N^{1.2})$ computation time. However, Algorithm-B can process a given sequence on-line, and Algorithm-B is faster than Algorithm-A for $2^{15} \leq N \leq 2^{24}$. Furthermore, Algorithm-B can compute the T-complexity for a random sequence of length $10^6 \approx 2^{20}$ within about 0.07 seconds. This means that Algorithm-B can be practically used in a randomness test that we propose in Chapter 6.

Table 2.1. Comparison of average time to compute the LZ-complexity and the T-complexity of a random sequence for various lengths.

Length (bits)	LZ-complexity (seconds)	Algorithm-A (seconds)	Algorithm-B (seconds)
2^{10}	2.36×10^{-4}	1.54×10^{-4}	5.63×10^{-4}
2^{11}	2.54×10^{-4}	2.46×10^{-4}	6.78×10^{-4}
2^{12}	2.98×10^{-4}	4.39×10^{-4}	1.13×10^{-3}
2^{13}	3.96×10^{-4}	8.28×10^{-4}	1.36×10^{-3}
2^{14}	7.81×10^{-4}	1.59×10^{-3}	1.97×10^{-3}
2^{15}	1.01×10^{-3}	3.34×10^{-3}	2.88×10^{-3}
2^{16}	1.42×10^{-3}	6.85×10^{-3}	4.77×10^{-3}
2^{17}	2.30×10^{-3}	2.06×10^{-2}	8.58×10^{-3}
2^{18}	4.10×10^{-3}	5.14×10^{-2}	1.67×10^{-2}
2^{19}	7.22×10^{-3}	1.10×10^{-1}	3.29×10^{-2}
2^{20}	1.35×10^{-2}	2.37×10^{-1}	6.80×10^{-2}
2^{21}	2.60×10^{-2}	4.51×10^{-1}	1.51×10^{-1}
2^{22}	5.13×10^{-2}	9.42×10^{-1}	3.43×10^{-1}
2^{23}	1.05×10^{-1}	1.88	7.90×10^{-1}
2^{24}	2.18×10^{-1}	3.82	1.81

2.3 Forward T-decomposition Algorithm for Generalized T-codes

In this section, we introduce a forward T-decomposition algorithm for generalized T-code sets by extending Algorithm-B.

We now consider the problem of parsing a sequence s sequentially to $p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots$ with each p_i satisfing Eq. (1.4), and k_i being a positive integer. As an example, let us consider the case of $s = s_1^{25} = 1100011110111111111111111$ being parsed as $p_1^{k_1} p_2^{k_2} p_3^{k_3} p_4^{k_4} p_5^{k_5} p_6^{k_6}$ with $k_1 = 2,\ k_2 = 3,\ k_3 = 1,\ k_4 = 1,\ k_5 = 2,\ k_6 = 1$ and the

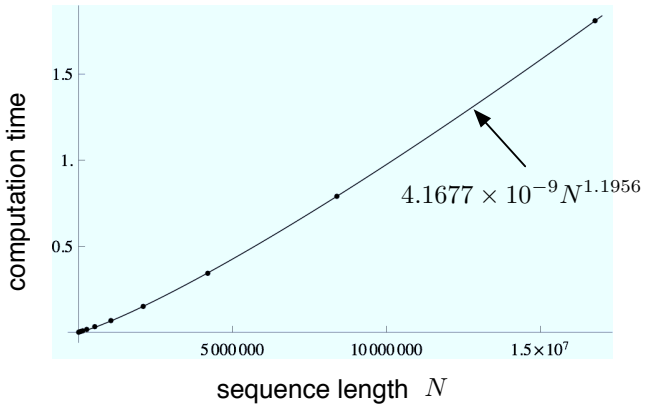


Figure 2.2. Computation time of the T-complexity using Algorithm-B.

subsequent p_i's.

$$
\begin{aligned}
p_1 &= 1 \\
p_2 &= 0 \\
p_3 &= p_1^2 1 \quad (= 111) \\
p_4 &= p_1 0 \quad (= 10) \\
p_5 &= p_3 p_1^2 1 \quad (= 111111) \\
p_6 &= p_1^2 1 \quad (= 111)
\end{aligned}
$$

Note that each p_i satisfies Eq. (1.4) and can be obtained by tries identical to those of Algorithm-B. However, in the case of forward T-decomposition for generalized T-codes, the value of k_i has to be stored in the node with index i.

Assume that $s_1^{10} = 1100011110$ is parsed as $p_1^{k_1} p_2^{k_2} p_3^{k_3} p_4^{k_4}$ and the tries $t_j,\ 0 \leq j \leq 4$, have already been constructed from $\{p_1, p_2, p_3, p_4\}$ and $\{k_1, k_2, k_3, k_4\}$ as shown in Fig. 2.3. Then p_5 is inferred from the tries as follows. First, we trace trie t_4 from the root to a leaf node following the remaining sequence of s, $s_{11}^{25} = 111111\cdots$. Since we reach leaf node 3 in t_4, we may conclude that the first part of p_5 consists of p_3, i.e., $p_5 = p_3 \cdots$. Because p_3 is parsed and $k_3 = 1$, the second p_j must satisfy $j \leq 2$. Hence, we move on to trie t_2. Again, we trace t_2 from the root to a leaf node following the remaining sequence of s, $s_{14}^{25} = 111\cdots$. Then, since we have reached

leaf node 1 in t_2, we know that the second part of p_5 is p_1, i.e., $p_5 = p_3 p_1 \cdots$. Since node 1 features $k_1 = 2$, p_1 may become the third p_j. Hence, we proceed with trie t_1 rather than trie t_0. Again, we trace t_1 from the root to a leaf node following the remaining sequence of s, $s_{15}^{25} = 11\cdots$. For we reached leaf node 1 in t_1, the third part of p_5 is p_1, i.e., $p_5 = p_3 p_1^2 \cdots$. Since p_1 is now included in p_5 k_1 times, p_1 cannot be used anymore. Hence next is trie t_0. In trie t_0, we cannot move from the root. In this case, the next symbol "1" becomes the literal symbol of p_5, and p_5 is given as $p_3 p_1^2 1$.

Trie t_5 can be created by adding p_5 into trie t_4 as shown in Fig. 2.3. At this point, we set $k_5 = 1$, which is stored at node with index 5. Next, we trace trie t_5 from the root to a leaf node following the remaining sequence of s, $s_{17}^{25} = 111111\cdots$. Since the tracing ends at the just created leaf node 5 in t_5, we find that p_5 occurs twice successively in s. Thus, we increment k_5 by 1 as shown in Fig. 2.3.

p_6 is obtained in a similar fashion. Following the remaining sequence of s, $s_{23}^{25} = 111$, we traverse trie t_5 from the root toward a leaf node, finally reaching node 3 in t_5. When $p_6 = p_3$, p_6 does not end with a literal symbol. Therefore, we move backwards until we reach the nearest node with a positive index, which is 1 in this example, and hence we obtain that $p_6 = p_1 \cdots$. Next we repeat the traversal of trie t_1 since $k_1 = 2$. After similar iterations, p_6 turns out $p_1^2 1$.

Alike to the case of the formal description of Algorithm-B, t_i contains any t_j for $0 \leq j < i$, and hence it is sufficient that only the latest trie is memorized.

The above scheme follows as Algorithm-C.

Algorithm-C

C1 (Initialization)

Let s be a given sequence.

$i := 1$.

Create t_0.

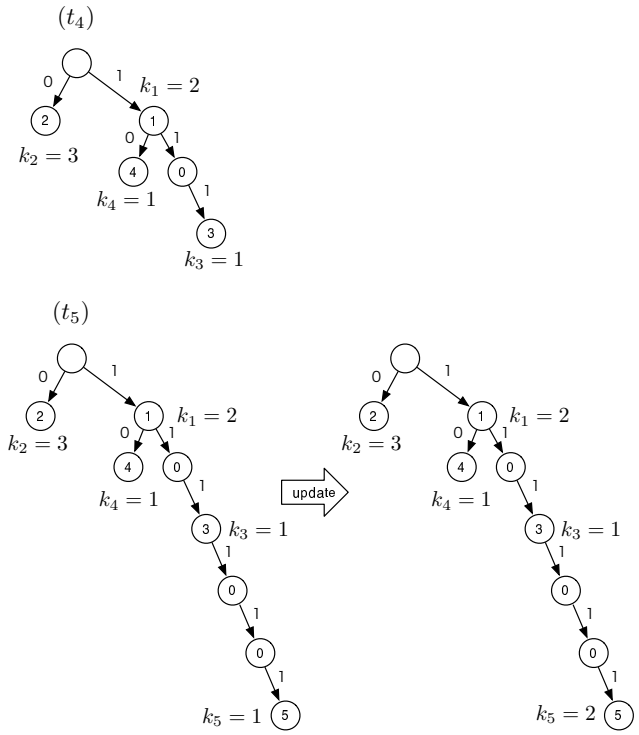


Figure 2.3. Trie growth for a sequence $s = 11000111101111111111111111$.

C2 $p_i := \lambda$. $\nu := i - 1$.

C3 Following s, trace a path from the root toward a leaf node in trie t_ν[*3] as far as possible. Let v represent the farthest node that we can reach.

- If v is a node with a positive index, then go to C4.
- If v is a node with index 0, then go to C5.
- If v is the root, i.e., we cannot move from the root, then go to C6.

C4 Let j be the index of node v.

If $j = i - 1$, then go to C7.

If p_i ends with $p_j^{k_j}$ or s is exhausted, then go to C5. Otherwise, $p_i := p_i p_j$.

Remove p_j from the head of s.

If $k_j = 1$, $\nu := j - 1$. Otherwise, $\nu := j$.

Go back to C3.

[*3] Trie t_ν is simulated by considering only the nodes with index l satisfying $l \leq \nu$ in trie t_{i-1}.

C5 If there exists no node with a positive index between v and the root, then go to C6. Otherwise, move back from node v toward the root in trie t_ν. Let $\hat{v}$ be the first node with a positive index that we find in the backward movement. Newly let v represent the node $\hat{v}$, and go back to C4.

C6 $p_i := p_i\omega$, where ω is the first symbol of s.

Output p_i.

If $i > 1$, output k_{i-1}.

Remove ω from the head of s.

$k_i := 1$.

If $s = \lambda$, output k_i, exit.

Otherwise, update trie t_{i-1} to t_i by adding p_i into t_{i-1}.

$i := i + 1$.

Go back to C2.

C7 $k_{i-1} := k_{i-1} + 1$.

Remove p_{i-1} from the head of s.

If $s = \lambda$, output k_{i-1}, exit.

Otherwise, go back to C3.

2.4 Conclusions

We devised the forward T-decomposition algorithm for simple T-codes, Algorithm-B, in Section 2.2, and then devised that for generalized T-codes, Algorithm-C, in Section 2.3. Both algorithms are efficient owing to the use of a trie structure. Algorithm-B can compute the T-complexity of a random sequence of length 10^6 within about 0.07 seconds with 3 GHz CPU. The experiment showed that its computation time for a random sequence of length N is about $O(N^{1.2})$, but there may be some room for improvement in the implementation of Algorithm-B.

In Chapter 6, we use Algorithm-B rather than Algorithm-C to compute the T-complexity of a sequence for a randomness test based on the T-complexity because in the case of random numbers, the same long pattern seldom occurs sequentially even if it occurs several times, and a defect of pseudo-random numbers such that some long pattern tends to occur sequentially can also be detected by Algorithm-B.

Chapter 3

Differential Equation Method for Derivation of the Formulas of the T-complexity and the LZ-complexity

3.1 Introduction

Titchener stated that the maximum T-complexity of a sequence of length N is very accurately described by $\mathrm{li}((\ln \#\mathcal{A})N)$ [57]. Although his finding was based solely on experimental evidence, it has been proved correct recently [59]. However, the proof of [59] is the so-called top-down approach and requires the knowledge that the maximum T-complexity profile is probably expressed as $\mathrm{li}((\ln \#\mathcal{A})N)$.

In this chapter, we show that the expression of the T-complexity profile can be derived using a differential equation technique. The proposed method is the so-called bottom-up approach and shows how the logarithmic integral function necessarily appears in the expression of the T-complexity profile. Moreover, in order to strengthen the reliability of our technique, we show that the proposed method is

applicable to not only the T-complexity profile but also the LZ-complexity profile. The distinctive feature of our approach is to focus on incremental quantities of average codeword length.

In Section 3.2, we briefly review the previous derivation method of the maximum T-complexity profile. Our differential equation method to derive the T-complexity profile and the LZ-complexity profile is shown in Section 3.3 and 3.4, respectively.

3.2 Previous Approach to the Derivation of the Maximum T-complexity Profile

In this section, we summarize the previous approach to derive the maximum T-complexity profile [59]. Let d_i be the number of codewords of length i in a systematic T-code set. Then, we may consider a generating function $d(z) = \sum_{j=1}^{\infty} d_j z^j$. A single T-augmentation step changes $d(z)$ to

$$\widetilde{d}(z) = (d(z) - 1)z^l + d(z) = (d(z) - 1)(z^l + 1) + 1,$$

where l is the length of the T-prefix chosen for this step. It immediately results in the following equation.

$$\frac{\tilde{d}(z) - 1}{d(z) - 1} = z^l + 1.$$

Let us consider a systematic T-code set when all codewords shorter than l have been just exhausted in the systematic T-augmentation. Furthermore, let m_l, n_l, and $d^l(z)$ be the number of codewords of length l, the length of a longest codeword, and the generating function of such a systematic T-code set, respectively. Then,

n_l and $d^l(z)$ are given as follows.

$$n_l = \sum_{i=1}^{l-1} i m_i + 1, \tag{3.1}$$

$$d^l(z) = (rz-1)\prod_{i=1}^{l-1}(z^i+1)^{m_i} + 1 = m_l z^l + d_{l+1}z^{l+1} + \cdots + d_{n_l}z^{n_l}, \tag{3.2}$$

where $r = \#\mathcal{A}$. Let $C_T(n_l)$ be the T-complexity when the length of a longest codeword is n_l. Since the T-complexity increases by one with each T-augmentation step, it holds that

$$C_T(n_l) = \sum_{i=1}^{l-1} m_i. \tag{3.3}$$

Hence, the value of $m_i,\ 1 \le i \le l-1$, is required in order to evaluate $C_T(n_l)$. From Eq. (3.2), we derive

$$l m_l = \left[\frac{\mathrm{d}d^l(z)}{\mathrm{d}z} \bmod z^l\right]_{z=1} = r^l - \sum_{i=1, i(j+1)=l, j\ge 0}^{l-1} (-1)^{\frac{l}{i}-1} i m_i. \tag{3.4}$$

Combining Eqs. (3.1) and (3.4), we get

$$\frac{r^l-1}{r-1} \le n_l < \frac{r^l}{r-1} + \frac{r^{\lfloor\frac{l+3}{2}\rfloor}}{(r-1)^2}. \tag{3.5}$$

Since $n_{l+1} - n_l = l m_l$, we obtain

$$\frac{r^l}{l} - \frac{r^{\lfloor\frac{l+3}{2}\rfloor}+r-1}{l(r-1)^2} < m_l < \frac{r^l}{l} + \frac{r^{\lfloor\frac{l+4}{2}\rfloor}+r-1}{l(r-1)^2}.$$

The combination of this inequality and Eq. (3.3) results in

$$\sum_{i=1}^{l-1}\frac{r^i}{i} - \delta_L(l) < C_T(n_l) < \sum_{i=1}^{l-1}\frac{r^i}{i} + \delta_U(l),$$

where

$$\begin{aligned}\delta_L(l) &= \frac{r}{(r-1)^2}\sum_{i=1}^{l-1}\frac{r^{\lfloor\frac{i+1}{2}\rfloor}}{i} + \frac{1}{(r-1)}\sum_{i=1}^{l-1}\frac{1}{i},\\ \delta_U(l) &= \frac{r^2}{(r-1)^2}\sum_{i=1}^{l-1}\frac{r^{\lfloor\frac{i}{2}\rfloor}}{i} + \frac{1}{(r-1)}\sum_{i=1}^{l-1}\frac{1}{i}.\end{aligned}$$

Using the knowledge that the maximum T-complexity profile is probably expressed as $\mathrm{li}(n_l \ln r)$, we can prove the following equations.

$$\begin{aligned}\lim_{l\to\infty}\frac{\delta_L(l)}{\mathrm{li}(n_l \ln r)} &= \lim_{l\to\infty}\frac{\delta_U(l)}{\mathrm{li}(n_l \ln r)} = 0,\\ \lim_{l\to\infty}\frac{\sum_{i=1}^{l-1}\frac{r^i}{i}}{\mathrm{li}(n_l \ln r)} &= 1.\end{aligned}$$

Finally we obtain

$$\lim_{l\to\infty}\frac{C_T(n_l)}{\mathrm{li}(n_l \ln r)} = 1.$$

3.3 Derivation of the T-complexity Profile

In this section, we derive the T-complexity profile, which is related to the forward T-decomposition for simple T-code sets. Although, similarly to the LZ78 incremental parsing algorithm, the forward T-decomposition may result in an incomplete parsing at the end of a sequence, the end effect becomes negligible asymptotically as the sequence length N goes to infinity.

Take a look at Fig. 3.1. Let $\mathcal{A}$ be the binary alphabet set with symbols 0 and 1, i.e., $\mathcal{A} = \{0, 1\}$ and $\#\mathcal{A} = 2$. Assume that an index number i, $i = 1, 2, \ldots,$ is assigned to each leaf in a code tree representing a T-code set $\mathcal{S}_n$. Let q_i and l_i be the probability and depth of leaf i, respectively. Then the average codeword

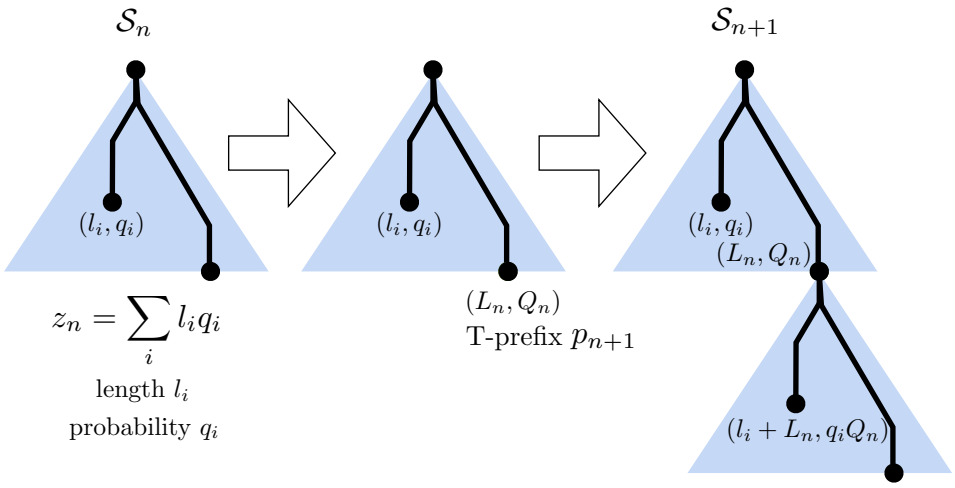


Figure 3.1. Illustration of code trees of simple T-code sets.

length of $\mathcal{S}_n$ is given by

$$z_n = \sum_i l_i q_i.$$

Let L_n be the length of the T-prefix p_{n+1} selected from $\mathcal{S}_n$, and let Q_n be the probability of p_{n+1}. Then, z_{n+1} is calculated as follows:

$$\begin{aligned} z_{n+1} &= z_n - L_n Q_n + \sum_i (l_i + L_n) Q_n q_i \\ &= z_n - L_n Q_n + Q_n z_n + L_n Q_n \\ &= (1 + Q_n) z_n. \end{aligned}$$

In the case of $\Pr(0) = \Pr(1) = 1/2$, Q_n is given by $1/2^{L_n}$. Hence, substituting $Q_n = 1/2^{L_n}$ into the above equation, we obtain

$$\frac{z_{n+1}}{z_n} = 1 + \frac{1}{2^{L_n}}. \tag{3.6}$$

From this equation and the approximation $\ln(1 + \eta) \approx \eta$ for $\eta \ll 1$, for large n,

we may deduce the following approximation.

$$\ln z_{n+1} - \ln z_n \approx \frac{1}{2^{L_n}}. \tag{3.7}$$

On the other hand, the expected length of sequences with T-complexity n is given by

$$\mathrm{E}[|p_1 p_2 \cdots p_n|] = \mathrm{E}\left[\sum_{i=1}^{n} |p_i|\right] = \sum_{i=1}^{n} \mathrm{E}[|p_i|].$$

Hence, the point $(\sum_{i=1}^{n} \mathrm{E}[|p_i|], n)$ is considered to lie on the T-complexity profile.

A sequence with maximum T-complexity for a given sequence length is called a maximum T-complexity sequence. First, let us consider a maximum T-complexity sequence s. This sequence can be obtained from a consecutive concatenation of T-prefixes of a systematic T-code set, i.e., $s = p_1 p_2 p_3 \cdots$, since systematic T-augmentation makes the longest codewords grow most slowly. In this case, $(n_{\max}^{(l)} - n_{\min}^{(l)} + 1)$ consecutive T-prefixes p_n, $n_{\min}^{(l)} \le n \le n_{\max}^{(l)}$, have the same length l for $n_{\min}^{(l)} = \min\{n \mid |p_n| = l\}$ and $n_{\max}^{(l)} = \max\{n \mid |p_n| = l\}$. Let m_l be the maximum number of codewords of length l generated during systematic T-augmentation. Then, ${m_l}^{*1}$ is asymptotically equivalent to $2^l/l$ as l goes to infinity [59]. Furthermore, let $z(l)$ be the average codeword length when the $\left((n_{\min}^{(l)} + n_{\max}^{(l)})/2\right)$-th T-prefix is determined. From Eq. (3.7), we have

$$\ln z(l+1) - \ln z(l) \approx \frac{1}{2^l}\frac{m_l}{2} + \frac{1}{2^{l+1}}\frac{m_{l+1}}{2} \approx \frac{1}{2l} + \frac{1}{2(l+1)} \approx \frac{1}{l}.$$

Substituting $q(l) \equiv \ln z(l)$, the above equation becomes

$$\frac{q(l+1) - q(l)}{(l+1) - l} = \frac{1}{l}.$$

[*1] It is known that m_l is related to the number of cyclic equivalence classes for l [12].

Table 3.1. Computation of k for the maximum T-complexity sequences.

l	m_l	$z(l)$	$l/z(l)$
1	2	1.5	0.667
2	3	3.1	0.636
3	2	4.9	0.607
4	6	6.7	0.600
5	6	8.8	0.570
6	11	10.5	0.572
7	18	12.2	0.572
8	36	14.1	0.568
9	56	16.0	0.564
10	105	17.7	0.564
11	186	19.5	0.563
12	346	21.3	0.563
13	630	23.1	0.562
14	1179	24.9	0.562
15	2182	26.7	0.562
16	4116	28.5	0.562
17	7710	30.3	0.562
18	14588	32.0	0.562
19	27594	33.8	0.562

Hence, for sufficiently large l, the differentiation approximation yields

$$\frac{\mathrm{d}q}{\mathrm{d}l} = \frac{1}{l},$$

which has a solution $q(l) = \ln z(l) = \ln l + C$, where C is an integral constant. This means that $z(l)$ is proportional to l, i.e., $l = kz(l)$ for a constant k when l is not small. Note from Table 3.1 that this relation holds accurately for $l \geq 13$, and k is given by about 0.562.

Since z_n is monotonically increasing, it holds that

$$\frac{l}{z_{n_{\max}^{(l)}}} \leq \frac{l}{z_n} \leq \frac{l}{z_{n_{\min}^{(l)}}}$$

for any $n \in [n^{(l)}_{\min}, n^{(l)}_{\max}]$ and a fixed l. Then, it holds from Eq. (3.6) that

$$\frac{l}{z_{n^{(l)}_{\min}}} - \frac{l}{z_{n^{(l)}_{\max}}} = \frac{l}{z_{n^{(l)}_{\min}}} - \frac{l}{z_{n^{(l)}_{\min}} \left(1 + \frac{1}{2^l}\right)^{m_l - 1}} \approx \frac{l}{z_{n^{(l)}_{\min}}} \left(1 - \frac{1}{\left(1 + \frac{1}{2^l}\right)^{\frac{2^l}{l} - 1}}\right).$$

Since $1/\left(1 + \frac{1}{2^l}\right)^{\frac{2^l}{l} - 1} \approx 1 - \frac{1}{l}$ for sufficiently large l,

$$\lim_{l \to \infty} \left(\frac{l}{z_{n^{(l)}_{\min}}} - \frac{l}{z_{n^{(l)}_{\max}}} \right) = 0.$$

This means that l/z_n can also be approximated by k for sufficiently large l because $l/z_{n^{(l)}_{\max}} \leq l/z(l) \leq l/z_{n^{(l)}_{\min}}$. Furthermore, since k does not depend on l for large l, we can conclude that $L_n = kz_n$ holds for all sufficiently large n.

From Eq. (3.7) and $L_n = kz_n$, we have

$$\ln z_{n+1} - \ln z_n \approx \frac{1}{2^{kz_n}}. \tag{3.8}$$

For $q_n \equiv \ln z_n$, this equation becomes

$$\frac{q_{n+1} - q_n}{(n+1) - n} = \frac{1}{2^{k \exp(q_n)}},$$

which can be approximated as

$$\frac{\mathrm{d}q}{\mathrm{d}n} = \frac{1}{2^{k \exp(q)}}$$

by the differentiation approximation. Substituting $q = \ln z$, we have

$$\frac{2^{kz}}{z} \mathrm{d}z = \mathrm{d}n.$$

Integrating this equation results in

$$\mathrm{li}(2^{kz}) = n + C_1. \tag{3.9}$$

Let N_n be the end point of p_n in a sequence s. Then,

$$\frac{N_{n+1} - N_n}{(n+1) - n} = |p_{n+1}| = L_n = kz_n.$$

This relation can also be approximated by

$$\frac{\mathrm{d}N}{\mathrm{d}n} = kz.$$

Combining $\frac{2^{kz}}{z}\mathrm{d}z = \mathrm{d}n$ and $\frac{\mathrm{d}N}{\mathrm{d}n} = kz$, we get

$$k2^{kz}\mathrm{d}z = \mathrm{d}N.$$

Integrating this equation yields $\frac{2^{kz}}{\ln 2} = N + C_2$, i.e., $2^{kz} = (N + C_2)\ln 2$, where C_2 is an integral constant. Substituting this equation into Eq. (3.9), we obtain

$$n = \mathrm{li}\left((N + C_2)\ln 2\right) - C_1, \tag{3.10}$$

which is the T-complexity of s. Equation (3.10) coincides with the result of [59]. Figure 3.2 shows how well $n = \mathrm{li}(N \ln 2)$ fits with the maximum T-complexity profile even for relatively small N.

Next, let us consider random sequences. In this case, L_n and z_n depend on a random sequence and hence they cannot be uniquely determined. Let z_n be the expectation of the average codeword length of $\mathcal{S}_n$. We assume that Eq. (3.8) also holds for random sequences. Since $\mathrm{E}[L_n] = z_n$, $\frac{N_{n+1}-N_n}{(n+1)-n} = |p_{n+1}| = L_n$ can be

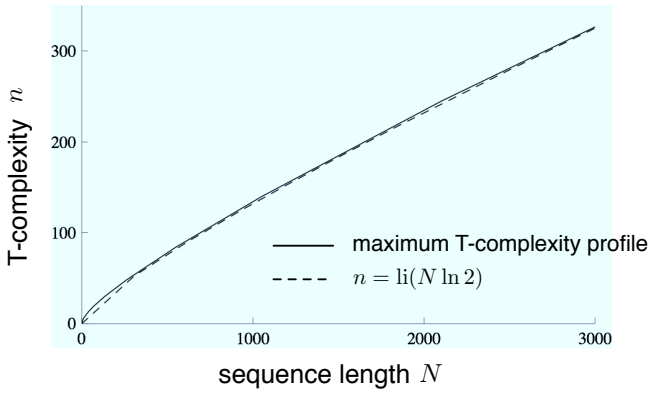


Figure 3.2. Maximum T-complexity profile and the graph of $n = \mathrm{li}(N \ln \#\mathcal{A})$ for the binary alphabet.

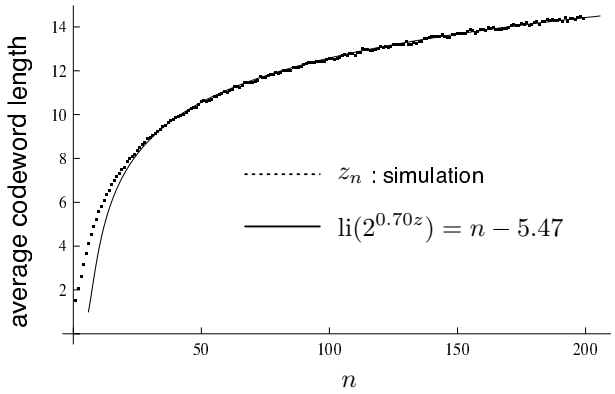


Figure 3.3. Graph of z_n for the T-complexity.

approximated by

$$\frac{\mathrm{d}N}{\mathrm{d}n} = z$$

in this case. Following a line of argument similar to the case of the maximum T-complexity sequences, we derive

$$n = \mathrm{li}\left(k(N + C_2)\ln 2\right) - C_1. \tag{3.11}$$

Figure 3.3 shows the graph of z_n computed from 10^4 pseudo-random sequences generated by Mersenne twister (MT) [39] , which coincides with the plot of Eq. (3.9) with $k = 0.70,\ C_1 = -5.47$ for $n \geq 25$.

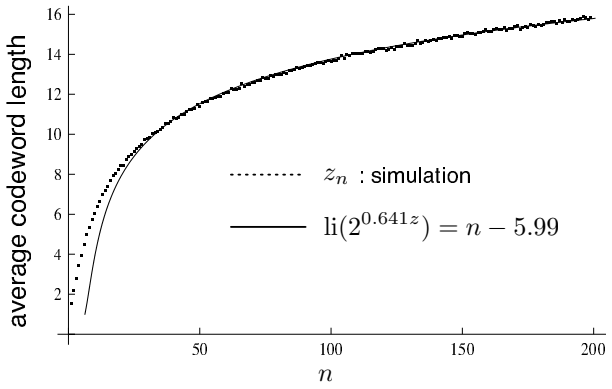


Figure 3.4. Graph of z_n for the T-complexity for the case of $p = 0.33$.

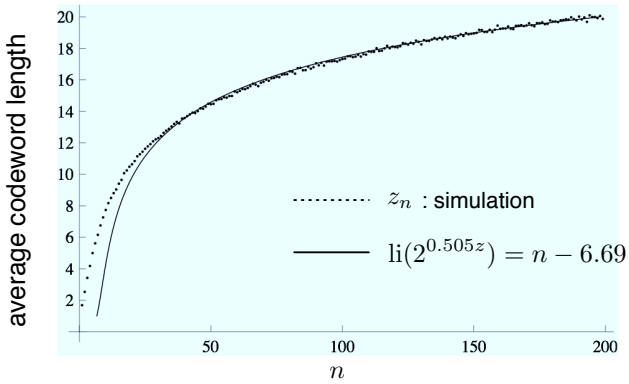


Figure 3.5. Graph of z_n for the T-complexity for the case of $p = 0.20$.

Finally, let us consider non-symmetric i.i.d. sources with $\Pr(0) = p,\ p \neq 1/2$. From the argument of typical sequences, the probability of codeword length l can be approximated by $2^{-lh(p)}$ for sufficiently large l, where $h(p)$ is the binary entropy function. The effect of $(2^{h(p)})^l$ is equivalent to the case where the alphabet size is reduced from 2 to $2^{h(p)}$. Hence, for this case, we obtain from Eq. (3.11) that

$$n = \mathrm{li}\,(kh(p)(N + C_2)\ln 2) - C_1. \tag{3.12}$$

As shown in Fig. 3.3, $k = 0.70$ matches with the simulation result of $p = 0.5$. Simulation also confirms that $kh(p) = 0.641$ and $C_1 = -5.99$ fit well with the setting of $p = 0.33$ (See Fig. 3.4), and $kh(p) = 0.505$ and $C_1 = -6.69$ fit well with the setting of $p = 0.20$ (See Fig. 3.5). Since $0.7 \times h(0.33) = 0.641$ and $0.7 \times h(0.20) = 0.505$, we can conjecture that $k = 0.70$ holds for all p.

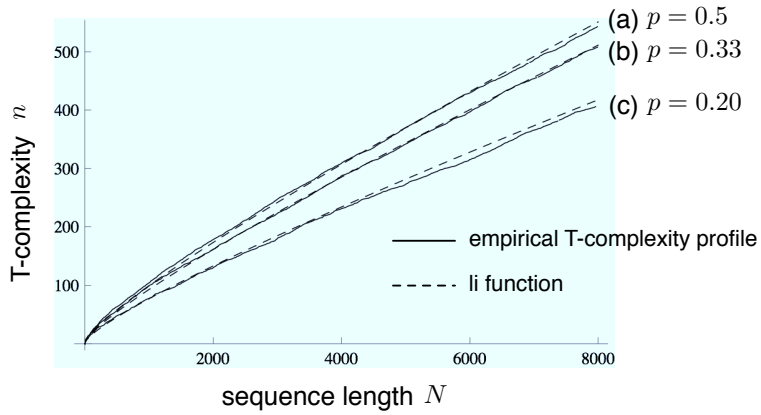


Figure 3.6. Empirical T-complexity profile and the graph of $n = \mathrm{li}((\ln \#\mathcal{A})H_T N)$ for the binary alphabet.

Titchener showed that the T-complexity profile of s with length N can be represented by $\mathrm{li}((\ln \#\mathcal{A})H_T N)$, where H_T is a real constant called T-entropy, $0 < H_T \leq 1$ [58]. Note that $kh(p)$ in Eq. (3.12) corresponds to H_T for the case of $\#\mathcal{A} = 2$. Therefore, the normalized T-entropy H_T/k coincides with the Shannon entropy.

Figure 3.6 compares the graph of the T-complexity profile of a sequence generated by MT and the graph of $n = \mathrm{li}((\ln \#\mathcal{A})H_T N)$ for three cases: (a) $p = 0.5$, $H_T = 0.7$, (b) $p = 0.33$, $H_T = 0.641$, and (c) $p = 0.20$, $H_T = 0.505$. The empirical T-complexity profiles were obtained from the forward T-decomposition algorithm for simple T-codes. In each case, the graph of $n = \mathrm{li}((\ln \#\mathcal{A})H_T N)$ fits well with the empirical T-complexity profile.

3.4 Derivation of the LZ-complexity Profile

The LZ-complexity profile can be derived in the same way as the T-complexity profile. Assume that a sequence $s = w_1 w_2 \cdots w_n$ is parsed by the LZ78 incremental parsing. See Fig. 3.7. Let z_n be the average codeword length of the n-th LZ78 parse tree [64]. Furthermore, let L_n and Q_n be the length of w_{n+1} and the probability

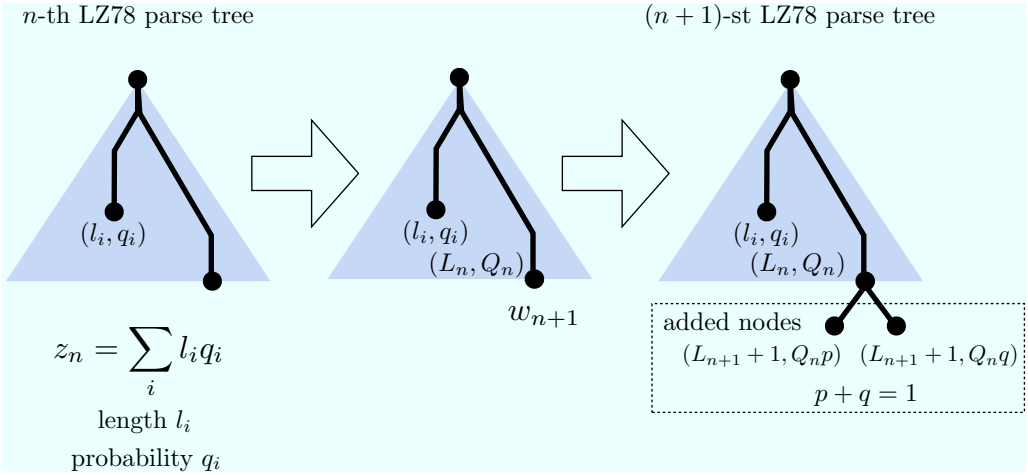


Figure 3.7. Illustration of LZ78 parse trees.

of w_{n+1}, respectively. Then, z_{n+1} is calculated as follows:

$$\begin{aligned} z_{n+1} &= z_n - L_n Q_n + (L_n+1)Q_n p + (L_n+1)Q_n q \\ &= z_n + Q_n, \end{aligned}$$

where $\Pr(0) = p$, $\Pr(1) = q$, and $p + q = 1$. Since $Q_n = 1/2^{L_n}$ for the case of $\Pr(0) = \Pr(1) = 1/2$, we obtain

$$z_{n+1} - z_n = \frac{1}{2^{L_n}}. \tag{3.13}$$

A sequence with maximum LZ-complexity for a given sequence length is called a maximum LZ-complexity sequence. First, let us consider a maximum LZ-complexity sequence, which can be obtained from a consecutive concatenation of all bit patterns of length $l = 1, 2, 3, \ldots$. In this case, $(n^{(l)}_{\max} - n^{(l)}_{\min} + 1)$ successive subsequences have the same length l for $n^{(l)}_{\min} = \min\{n \mid |w_n| = l\}$ and $n^{(l)}_{\max} = \max\{n \mid |w_n| = l\}$. The number of subsequences with length l is 2^l. Let $z(l)$ be the average codeword length when the $\Big((n^{(l)}_{\min} + n^{(l)}_{\max})/2\Big)$-th word is

determined. Then, we have

$$z(l+1) - z(l) = \frac{2^{l-1}}{2^l} + \frac{2^l}{2^{l+1}} = 1.$$

This means that $z(l) = l - k$, where k is a constant. Since z_n is increasing infinitely and $z_{n^{(l)}_{\max}} - z_{n^{(l)}_{\min}}$ is less than 1 for any large l, the relation $L_n = z_n + k$ holds for all sufficiently large n. From this relation and Eq. (3.13), we obtain

$$z_{n+1} - z_n = \frac{1}{2^{z_n+k}}, \tag{3.14}$$

which can be further approximated by

$$\frac{\mathrm{d}z}{\mathrm{d}n} = \frac{1}{2^{z+k}},$$

or

$$2^{z+k}\mathrm{d}z = \mathrm{d}n. \tag{3.15}$$

Integrating this equation, we get

$$\frac{2^{z+k}}{\ln 2} + C_1 = n,$$

or

$$z = \log_2((n - C_1)\ln 2) - k, \tag{3.16}$$

where C_1 is an integral constant. Let N_n be the end point of w_n in a sequence s. Then,

$$\frac{N_{n+1} - N_n}{(n+1) - n} = |w_{n+1}| = L_n = z_n + k.$$

This relation can be approximated by

$$\frac{\mathrm{d}N}{\mathrm{d}n} = z + k. \tag{3.17}$$

Combining Eqs. (3.15) and (3.17), we have

$$(z+k)2^{z+k}\mathrm{d}z = \mathrm{d}N.$$

Integrating this equation, we have

$$\frac{2^{z+k}((z+k)\ln 2 - 1)}{(\ln 2)^2} = N + C_2, \tag{3.18}$$

where C_2 is an integral constant. Combining Eqs. (3.16) and (3.18), we obtain

$$n - C_1 = \frac{N + C_2}{\log_2(n - C_1) + A_1}, \tag{3.19}$$

where $A_1 = \frac{-1+\ln\ln 2}{\ln 2} \approx -1.97146$. Using Eq. (3.19) recursively, we can derive

$$n = \frac{N + C_2}{(1 - A_2)\log_2(N + C_2)} + C_1,$$

where

$$\begin{aligned} A_2 &= \frac{-A_1 + \log_2(\log_2(N + C_2) + \mathrm{O}(\log_2\log_2 N))}{\log_2(N + C_2)} \\ &< \frac{-A_1 + \log_2(2\log_2(N + C_2))}{\log_2(N + C_2)} \\ &= \frac{(1 - A_1) + \log_2\log_2(N + C_2)}{\log_2(N + C_2)}. \end{aligned}$$

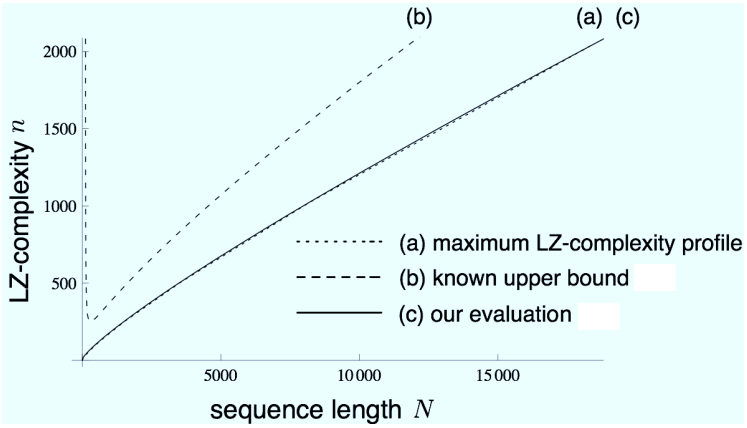


Figure 3.8. Comparison of Eq. (3.20) and our evaluation Eq. (3.19).

This formula agrees well with the following upper bound on the LZ-complexity [3]

$$n \leq \frac{N}{(1-\varepsilon_N)\log_2 N}, \tag{3.20}$$

where

$$\varepsilon_N = \min\left\{1, \frac{\log_2(\log_2 N)+4}{\log_2 N}\right\}.$$

Figure 3.8 shows the upper bound given by Eq. (3.20) and the graph computed from Eq. (3.19) with $C_1 = C_2 = 0$, which is very close to the maximum LZ-complexity profile.

Next, let us consider random sequences. Similarly to the case of the T-complexity, we assume that Eq. (3.14) also holds for random sequences. Since $\mathrm{E}[|w_{n+1}|] = z_n$, $\frac{N_{n+1}-N_n}{(n+1)-n} = |w_{n+1}|$ can be approximated by

$$\frac{\mathrm{d}N}{\mathrm{d}n} = z$$

in this case. Using an argument similar to the case of the maximum LZ-complexity

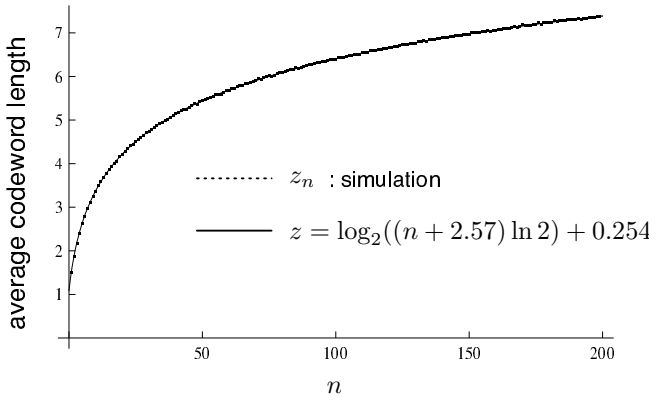


Figure 3.9. Graph of z_n for the LZ-complexity.

sequences, we can derive

$$n - C_1 = \frac{N + C_2}{\log_2(n - C_1) + (A_1 - k)}. \tag{3.21}$$

Figure 3.9 shows the graph of z_n computed from 10^4 pseudo-random sequences generated by MT, which is well fitted by the graph of Eq. (3.16) with $k = -0.254$ and $C_1 = -2.57$.

Finally, let us consider non-symmetric i.i.d. sources with $\Pr(0) = p,\ p \neq 1/2$. Following the argument of typical sequences, the probability of codeword length l can be approximated by $2^{-lh(p)}$ for sufficiently large l. The effect of $(2^{h(p)})^l$ is equivalent to the case where the alphabet size $\#\mathcal{A}$ is reduced from 2 to $2^{h(p)}$. In general, Eq. (3.15) is written as follows:

$$(\#\mathcal{A})^{z+k}\mathrm{d}z = \mathrm{d}n.$$

Integrating this equation, we get

$$\frac{(\#\mathcal{A})^{z+k}}{\ln(\#\mathcal{A})} + C_1 = n,$$

or

$$z = \frac{\log_2((n - C_1)\ln(\#\mathcal{A}))}{\log_2(\#\mathcal{A})} - k.$$

For $\#\mathcal{A} = 2^{h(p)}$, this equation becomes as follows.

$$z = \frac{\log_2((n - C_1)h(p)\ln 2)}{h(p)} - k. \tag{3.22}$$

Figures 3.10 and 3.11 show the graph of z_n for the case of $p = 0.33$ and $p = 0.20$, respectively. The graph of z_n was computed from 10^4 sequences generated by MT. $k = -0.333$ and $C_1 = -2.45$ fit well with the simulation result of $p = 0.33$, and $k = -0.599$ and $C_1 = -2.49$ fit well with the simulation result of $p = 0.20$. Since $\#\mathcal{A} = 2^{h(p)}$, the following equation is used instead of Eq. (3.21).

$$n - C_1 = \frac{N + C_2}{\frac{\log_2(n-C_1)}{h(p)} + (B_1 - k)}, \tag{3.23}$$

where $B_1 = \frac{-1+\ln(h(p)\ln 2)}{h(p)\ln 2}$. Figure 3.12 compares the graph of the LZ-complexity profile of a sequence generated by MT and the graph computed from Eq. (3.23) with $C_1 = C_2 = 0$ for three cases: (a) $p = 0.5,\ k = -0.254$, (b) $p = 0.33,\ k = -0.333$, and (c) $p = 0.20,\ k = -0.599$. In each case, the graph computed from Eq. (3.23) fits well with the empirical LZ-complexity profile. Inserting $n = 1$ and $C_1 = -2.50$ in Eq. (3.22), we obtain $k = \frac{\log_2(3.50h(p)\ln 2)}{h(p)} - z_1$, where $z_1 = 2(p^2 - p + 1)$. z_1 can be easily derived from the LZ78 parse tree. Note from Fig. 3.13 that this k represents well the values of k obtained by simulation.

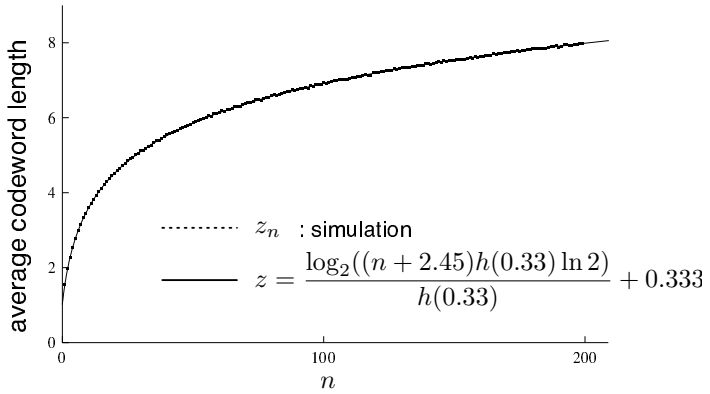


Figure 3.10. Graph of z_n for the LZ-complexity for the case of $p = 0.33$.

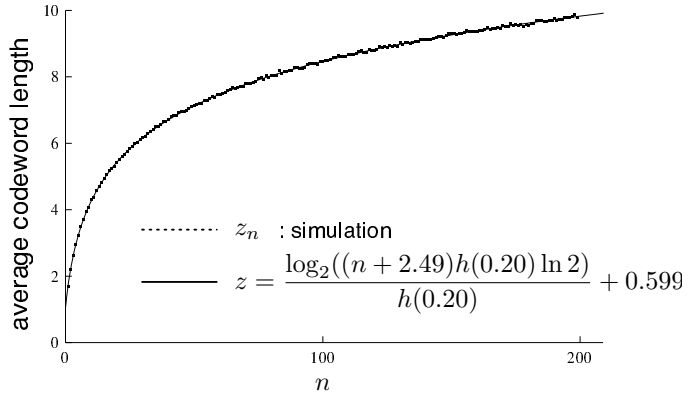


Figure 3.11. Graph of z_n for the LZ-complexity for the case of $p = 0.20$.

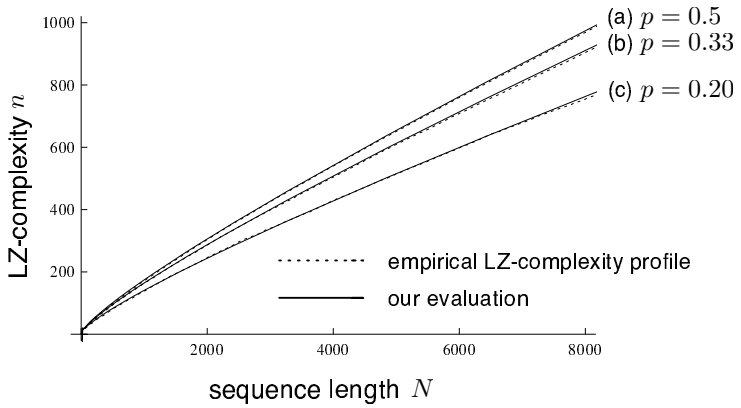


Figure 3.12. Empirical LZ-complexity profile and the graph computed from Eq. (3.21).

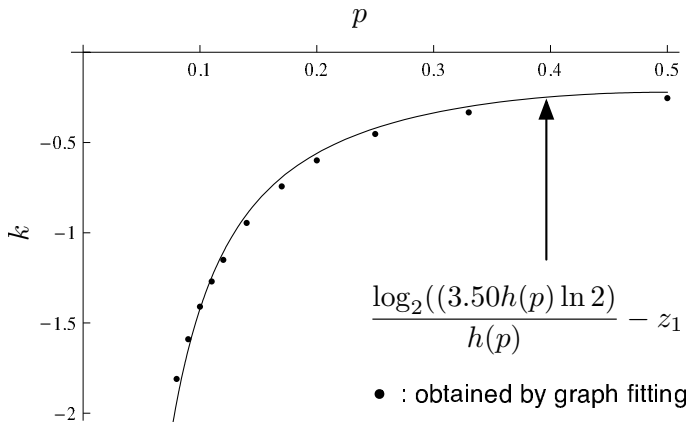


Figure 3.13. Graph of k for various p's.

3.5 Conclusions

In this chapter, the expressions of the T-complexity profile and the LZ-complexity profile were derived in a unified way using the proposed differential equation technique.

We summarized the previous approach [59] to give the proof of the Titchener's conjecture regarding the maximum T-complexity profile in Section 3.2. In Section 3.3, first, we showed our new differential equation technique to derive the maximum T-complexity profile. Then, we derived the T-complexity profile for random sequences, following a line of argument similar to the case of the maximum T-complexity sequences. In this derivation, we assumed that Eq. (3.8) holds for random sequences. Alike to the case of the T-complexity profile, we derived the expressions of the maximum LZ-complexity profile and the LZ-complexity profile for random sequences in Section 3.4. To derive the LZ-complexity profile for random sequences, we assumed that Eq. (3.14) holds for random sequences. The obtained formulas regarding the T-complexity profile and the LZ-complexity profile agree well with the ones in previous studies. We also clarified that the T-entropy H_T corresponds to $kh(p)$ in our derivation.

The proof in [59] requires the knowledge that the maximum T-complexity profile is probably expressed as $\mathrm{li}((\ln \#\mathcal{A})N)$. By contrast, our derivation can explain

the reason why the maximum T-complexity profile is necessarily expressed using the logarithmic integral function. This is the distinct advantage of our derivation over the proof in [59].

Chapter 4

Properties of the Maximum T-complexity Sequences

4.1 Introduction

The magnitude of the T-complexity of a given sequence s in general indicates the degree of randomness. As shown in Chapter 3, the T-complexity profile of a sequence of length N generated from a stationary ergodic source with entropy h is described as $\mathrm{li}(khN \ln \#\mathcal{A})$, where k is a constant. The empirical mean and standard deviation of the T-complexity of binary random sequences of length 10^6 are 38720.6 and 58.2937, respectively. However, there exist interesting sequences that have larger T-complexities than any random sequences (See Fig. 4.1). A maximum T-complexity sequence is a sequence with maximum T-complexity for a given sequence length, and is obtained from a consecutive concatenation of T-prefixes of a systematic T-augmentation. The T-complexity of a binary maximum T-complexity sequence of length 10^6 is 56170.

In this chapter, we investigate several properties of the maximum T-complexity sequences using various techniques including the NIST test suite[*1] [21]. The prop-

[*1] The NIST test suite is the package of statistical randomness tests released by the NIST of

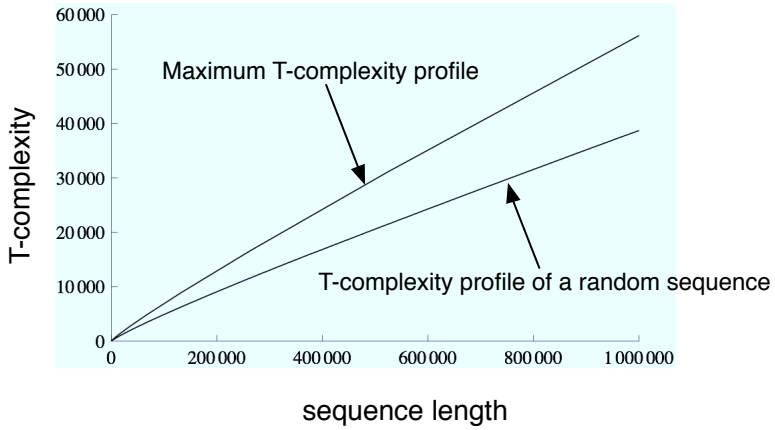


Figure 4.1. Maximum T-complexity profile for the binary alphabet.

erties of the maximum T-complexity sequences are compared with those of the maximum LZ-complexity sequences, each of which is obtained from a consecutive concatenation of all bit patterns of length $l = 1, 2, 3, \cdots$.

In Section 4.2, we show an algorithm to generate the binary maximum T-complexity sequences, Algorithm-D. Since we compare the properties of the maximum T-complexity sequences with those of the maximum LZ-complexity sequences, we also show an algorithm to generate the binary maximum LZ-complexity sequences, Algorithm-E. Various maximum T-complexity and LZ-complexity sequences are generated by using Algorithm-D and Algorithm-E, respectively. Analysis results on those sequences are shown in Section 4.3.

4.2 Generation Algorithm of the Maximum T-complexity Sequences

First, we introduce an algorithm that enables the generation of a binary maximum T-complexity sequence of length 10^6 with T-complexity 56170. Since systematic T-augmentation makes longest codewords grow slowest, a maximum T-complexity sequence s can be generated by a consecutive concatenation of T-prefixes of a

the U.S. government. Details of the NIST test suite are described in Subsection 6.1.2.

systematic T-code set, i.e., $s = p_1p_2p_3\cdots$. Let n_l be the length of a longest codeword in a systematic T-code set when all the codewords shorter than l have been just exhausted in the systematic T-augmentation. From Eq. (3.5), we find that $n_l > 10^6$ for $r = 2$ and $l = 20$. Hence, we may ignore any codeword whose length is not less than 20 in the systematic T-augmentation for the generation of a binary maximum T-complexity sequence of length 10^6.

The algorithm uses an array $X[i],\ i = 0, 1, 2, \ldots$, such that each $X[i]$ takes a value from the set $\{0, 1, 2\}$ and each node of the array is related to a specific string denoted by $x[i]$. For all i, $X[i]$ is initialized to zero. Let $l(i)$ be the integer l that satisfies the inequality $2^l - 2 \leq i < 2^{l+1} - 2$. The string $x[i]$ is defined as a binary $l(i)$-bit representation of an integer $i - (2^{l(i)} - 2)$. For example, $x[0] = 0, x[1] = 1, x[2] = 00, x[3] = 01, x[4] = 10, x[5] = 11, x[6] = 000, x[7] = 001, x[8] = 010, x[9] = 011, x[10] = 100, x[11] = 101, x[12] = 110, x[13] = 111, \cdots$. Since we have only to consider codewords shorter than 20, the array size is set to 1048574 $(= 2^{19+1} - 2)$. '$X[i] = 0$' indicates that $x[i]$ does not belong to a systematic T-code set. When $x[i]$ belongs to a systematic T-code set, $X[i]$ is updated to one. The value '2' is only used for ease of the algorithm. Let $\mathcal{Y}_l$ be a set $\{i \mid 2^l - 2 \leq i < 2^{l+1} - 2,\ X[i] = 1\}$. '$i \in \mathcal{Y}_l$' means that $x[i]$ is a codeword of length l in the systematic T-code set. Let m_l be the number of codewords of length l when all the codewords shorter than l have been just exhausted in the systematic T-augmentation. When a codeword $x[y]$ of length $l(y)$ is chosen as the T-prefix in a systematic T-augmentation step, for each i such that $X[i] = 1$, a codeword $x[y]x[i]$ of length $(l(y)+l(i))$ is added to the systematic T-code set. The algorithm creates a maximum T-complexity sequence of length $\sum_{l=1}^{19} lm_l = 1049522$, and then outputs the first 10^6 bits of it. The above generation scheme of a binary maximum T-complexity sequence is formally described as the following Algorithm-D. Note that Algorithm-D can also generate a longer maximum T-complexity sequence by

changing the array size and the termination condition.

Algorithm-D

D1 (Initialization)

For each $i \in [2, 1048573]$, $X[i] := 0$. $X[0] := 1$. $X[1] := 1$. $l := 1$. $s := \lambda$. $\mathcal{Y}_l := \{0, 1\}$.

D2 Randomly choose an element y from $\mathcal{Y}_l$.

D3 Append $x[y]$ to s. Remove y from $\mathcal{Y}_l$.

D4 For each $i \in [0, 1048573]$, if $X[i] = 1$ and $l(i) + l \leq 19$, then

$$X[(2^{l(i)+l} - 2) + (y - (2^l - 2))2^{l(i)} + (i - (2^{l(i)} - 2))] := 2.$$

D5 For each $i \in [0, 1048573]$, if $X[i] = 2$, then $X[i] := 1$.

D6 If $\#\mathcal{Y}_l = 0$ and $l = 19$, then output the first 10^6 bits of s, and exit.

D7 Otherwise, if $\#\mathcal{Y}_l = 0$, then $l := l + 1$, $\mathcal{Y}_l := \{i \mid 2^l - 2 \leq i < 2^{l+1} - 2,\ X[i] = 1\}$.

D8 Go back to D2.

Next, we present a scheme for generating a binary maximum LZ-complexity sequence of length 10^6 with LZ-complexity 70690. A maximum LZ-complexity sequence s can be obtained from a sequential and consecutive concatenation of all bit patterns of length $l = 1, 2, 3, \ldots$. Since $\sum_{l=1}^{16} l2^l = 1966082 > 10^6$, we have only to consider binary patterns whose lengths are at most 16. The string $x[i]$ is defined in the same way as Algorithm-D. The algorithm creates a maximum LZ-complexity sequence of length 1966082, and then outputs the first 10^6 bits of it. Above approach of binary maximum LZ-complexity sequence generation is formally described as the following Algorithm-E.

Algorithm-E

E1 (Initialization)

$l := 1$. $s := \lambda$. $\mathcal{Y}_l := \{0, 1\}$.

E2 Randomly choose an element y from $\mathcal{Y}_l$.

E3 Append $x[y]$ to s. Remove y from $\mathcal{Y}_l$.

E4 If $\#\mathcal{Y}_l = 0$ and $l = 16$, then output the first 10^6 bits of s, and exit.

E5 Otherwise, if $\#\mathcal{Y}_l = 0$, then $l := l + 1$, $\mathcal{Y}_l := \{i \mid 2^l - 2 \leq i < 2^{l+1} - 2\}$.

E6 Go back to E2.

4.3 Experiments

We evaluated the maximum T-complexity and LZ-complexity sequences (10^3 sequences of length 10^6) with the NIST test suite in order to compare the two types of sequences. The significance level α was set to 0.01, and the default parameter values were used. The original NIST test suite was modified according to the suggestions made in [2, 15–18, 20].

Figures 4.2 and 4.3 show the respective pass ratios. We observe that the maximum T-complexity sequences are less random than the maximum LZ-complexity sequences. Evaluation results of the DFT test and the Universal test included in the NIST test suite are shown in Tables 4.1 and 4.2. For the maximum T-complexity sequences, the empirical distributions of P-values obtained from the DFT test and the Universal test are biased to small values. By contrast, they are almost uniform for the maximum LZ-complexity sequences.

We analyzed a maximum T-complexity sequence in depth using the discrete Fourier transform (DFT) and the autocorrelation function. Let $s = s_0 s_1 \cdots s_{N-1} \in \{0, 1\}^N$ be a binary sequence of length N. The discrete Fourier transform of s is

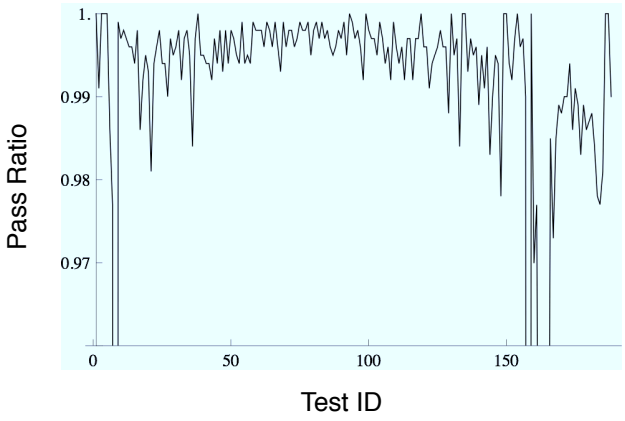


Figure 4.2. Pass ratios for the maximum T-complexity sequences.

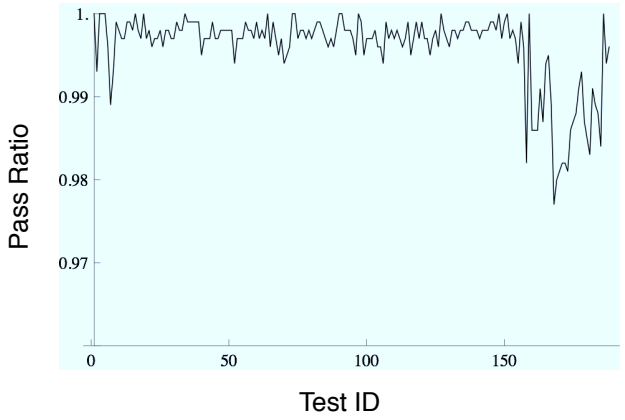


Figure 4.3. Pass ratios for the maximum LZ-complexity sequences.

Table 4.1. Test Results of the DFT test and the Universal test for the maximum T-complexity sequences.

	DFT test	Universal test
f_1	1000	878
f_2	0	53
f_3	0	25
f_4	0	15
f_5	0	8
f_6	0	5
f_7	0	6
f_8	0	4
f_9	0	3
f_{10}	0	3
Uniformity U	0.000000	0.000000
Pass ratio P	0.000000	0.352

Table 4.2. Test Results of the DFT test and the Universal test for the maximum LZ-complexity sequences.

	DFT test	Universal test
f_1	99	122
f_2	89	88
f_3	99	97
f_4	103	85
f_5	97	97
f_6	110	114
f_7	94	110
f_8	119	83
f_9	92	104
f_{10}	98	100
Uniformity U	0.630872	0.098920
Pass ratio P	0.993	0.982

given by

$$F_j = \sum_{k=0}^{N-1} (2s_k - 1) \exp\left(\mathrm{i}\frac{2\pi jk}{N}\right),\ 0 \le j \le N-1,$$

where i is the imaginary unit. The spectrum of s is defined as

$$\left\{ |F_j| : 0 \le j \le \frac{N}{2} - 1 \right\}.$$

Note that there is a well-known relation $F_j = \bar{F}_{N-j}$, where $\bar{F}_{N-j}$ is the complex conjugate of F_{N-j}. Let τ be a lag. When s is a periodic sequence with period N, the autocorrelation function of s [40] is defined as

$$\hat{C}(\tau) = \frac{1}{N} \sum_{k=0}^{N-1} (2s_k - 1)(2s_{k+\tau} - 1),\ 0 \le \tau \le N-1.$$

Figures 4.4 and 4.5 show the spectrum and the autocorrelation function[*2] of a maximum T-complexity sequence of length 10^6, respectively. The spectrum

[*2] $\hat{C}(0)$ was excluded from the figure because its value is always N.

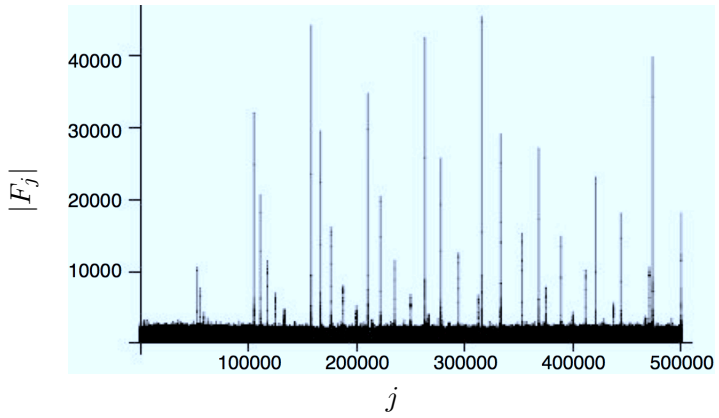


Figure 4.4. Spectrum of a maximum T-complexity sequence.

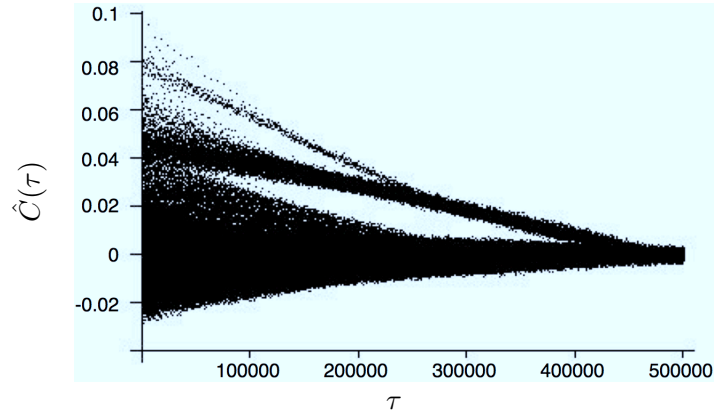


Figure 4.5. Autocorrelation function of a maximum T-complexity sequence.

has strong spikes, and the graph of the autocorrelation function is considerably different from the graph that is expected for random sequences. However, as shown in Figs. 4.6 and 4.7, the spectrum of a maximum LZ-complexity sequence did not have strong spikes, and its graph of autocorrelation function was not characteristic.

The non-randomness in the maximum T-complexity sequences is caused by a particularity of the generation procedure. A maximum T-complexity sequence is generated by a consecutive concatenation of T-prefixes of a systematic T-code set. In the sequence, T-prefixes of the same length are positioned next to each other. Each T-prefix is constructed by the recursive concatenation of previous T-prefixes. Since one of the shortest available codewords is chosen as the T-prefix for each T-augmentation step, the possible patterns of T-prefixes are restricted compared to

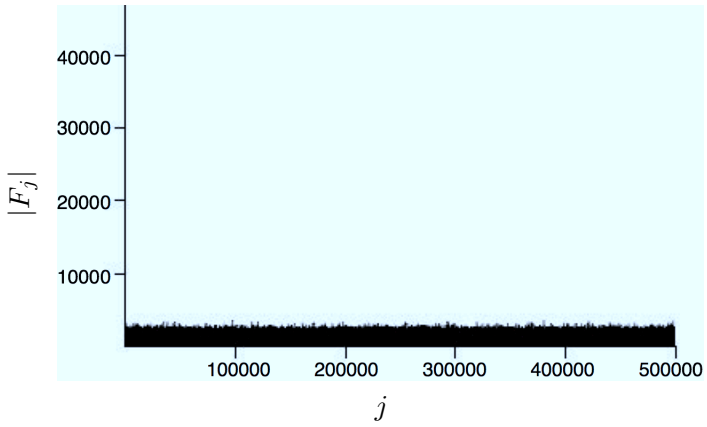


Figure 4.6. Spectrum of a maximum LZ-complexity sequence.

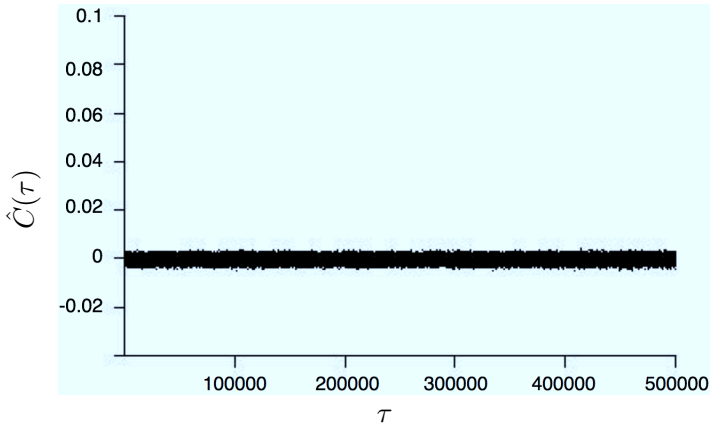


Figure 4.7. Autocorrelation function of a maximum LZ-complexity sequence.

the case of random sequences.

The non-randomness in the maximum T-complexity sequences is quantitatively evaluated as follows. When Algorithm-D creates a maximum T-complexity sequence of length 1049522, it requires a random sequence of

$$\sum_{l=1}^{19} \sum_{i=1}^{m_l} \log_2 i \approx 729910 \text{ bits.}$$

This length is about 69% of the length of a maximum T-complexity sequence (1049522 bits). By contrast, when Algorithm-E creates a maximum LZ-complexity

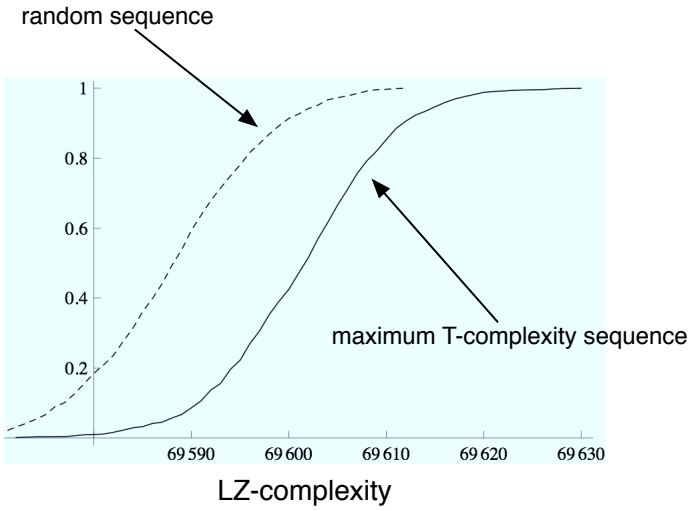


Figure 4.8. Empirical distributions of LZ-complexities of the maximum T-complexity sequences (solid line) and Marsaglia's random sequences (broken line).

sequence of length 1966082, it requires a random sequence of

$$\sum_{l=1}^{16}\sum_{i=1}^{2^l}\log_2 i \approx 1777077 \text{ bits.}$$

This length is about 90% of the length of a maximum LZ-complexity sequence (1966082 bits). Since $0.69 < 0.90$, Algorithm-E consumes more random bits than Algorithm-D. This is why the maximum T-complexity sequences are less random than the maximum LZ-complexity sequences.

However, we must note that, the magnitude of the T-complexity of a given sequence s in general indicates the degree of randomness (See Figs. 6.3 and 6.4). We must also note the following results. Figure 4.8 shows an empirical distribution of LZ-complexities of 10^3 maximum T-complexity sequences of length 10^6 generated by Algorithm-D and that of 10^3 Marsaglia's random sequences of length 10^6 extracted from a file named 'bits.01' [37]. Surprisingly, we found that the maximum T-complexity sequences on average show larger LZ-complexities than random sequences. So, the maximum T-complexity sequences are also 'complex' in terms of the LZ-complexity.

4.4 Conclusions

In Section 4.2, we showed an algorithm to generate the binary maximum T-complexity sequences, Algorithm-D, and an algorithm to generate the binary maximum LZ-complexity sequences, Algorithm-E. The properties of the maximum T-complexity sequences were investigated and compared with those of the maximum LZ-complexity sequences in Section 4.3.

In our analysis, the NIST test suite, the discrete Fourier transform (DFT), and the autocorrelation function were used. We applied 10^3 maximum T-complexity sequences of length 10^6 generated by Algorithm-D and 10^3 maximum LZ-complexity sequences of length 10^6 generated by Algorithm-E to the NIST test suite.

The analysis showed that the maximum T-complexity sequences are less random than the maximum LZ-complexity sequences. The DFT test and the Universal test included in the NIST test suite clearly detected the non-randomness in the maximum T-complexity sequences. Several strong spikes were seen in the spectrum of a maximum T-complexity sequence and its autocorrelation function was considerably different from the graph that is expected for random sequences. However, such characteristics were not found in the case of the maximum LZ-complexity sequences. Hence, we cannot consider that the maximum T-complexity sequences are random although they have larger T-complexities than random sequences. On the other hand, the maximum T-complexity sequences on average show larger LZ-complexities than random sequences. The cause of the non-randomness in the maximum T-complexity sequences was quantitatively and qualitatively explained.

On the basis of the test results, a randomness test based on the T-complexity (T-complexity test) proposed in Chapter 6 uses the two-tailed test.

Chapter 5

Application of the Forward T-decomposition to Data Compression

5.1 Introduction

The T-complexity of a sequence in general tends to be smaller than the respective LZ-complexity. Thus, data compression based on T-codes is expected to achieve a better compression performance than the so-called LZ78 family. However, no efficient data compression scheme has been devised from the spirit of T-decomposition so far.

In a student research project [41], C. Müller and R. Schimpfky considered several compression schemes based on T-codes, in which the T-prefix matrix is compressed. But, even the best scheme presented in [41] could not compete with the performance of the UNIX *compress*, a variant of LZ78, on the Calgary Corpus [52].

In this chapter, we propose a new data compression scheme based on a dictionary method such that all phrases added to a dictionary have a recursive structure

similar to T-codes [25]. Our scheme can compress the Calgary Corpus more efficiently than both the known schemes presented in [41] and the UNIX *compress*. We examine three dictionary updating rules called Methods A, B, and C.

In Section 5.2, we review LZ78, LZW, and LZMW, all of which belong to the LZ78 family. In Section 5.3, we briefly describe the best scheme presented by C. Müller and R. Schimpfky in [41], called the Müller-Schimpfky scheme in this thesis, and show its defects. Then, we propose a new data compression scheme based on T-codes in Section 5.4. Our scheme can overcome the defects of the Müller-Schimpfky scheme. In Section 5.5, the performance of our scheme is compared with that of other schemes on the Calgary Corpus. The difference between Methods A, B, and C is closely examined using the dictionary size and the number of parsed subsequences in Section 5.6. It is also shown that the proposed scheme can be implemented as a single-pass scheme and our scheme with Method C is universal for stationary ergodic sources in Section 5.6.

Throughout this chapter, an input file is treated as a sequence of 8-bit symbols, and hence its alphabet $\mathcal{A}$ has 2^8 different symbols, which corresponds to the ASCII code in the range of 0 to 255.

5.2 LZ78 Family

In this section, we briefly review LZ78, LZW, and LZMW, all of which belong to the so-called LZ78 family using a dictionary method.

LZ78 [66] processes a given sequence s as follows. The dictionary $\mathcal{D}$ starts empty. The null string λ is assumed to be in position 0 in $\mathcal{D}$. Suppose that the LZ78 incremental parsing (See Subsection 1.3.2) has parsed s up to the j-th word as $s(n_0+1, n_1)s(n_1+1, n_2)s(n_2+1, n_3)\cdots s(n_{j-1}+1, n_j)$, and the i-th word $s(n_i+1, n_{i+1})$ is added to $\mathcal{D}$ at position i. Let $d(w)$ be the word obtained by deleting the last symbol of w. Furthermore, let $\pi(j)$ be the non-negative integer i, $0 \leq i < j$,

Table 5.1. Encoding steps in LZ78 for the sequence "*abracadabraabracadabra*".

Entry	Phrase Added to $\mathcal{D}$	Token
0	λ	–
1	a	$(0, a)$
2	b	$(0, b)$
3	r	$(0, r)$
4	ac	$(1, c)$
5	ad	$(1, d)$
6	ab	$(1, b)$
7	ra	$(3, a)$
8	abr	$(6, r)$
9	aca	$(4, a)$
10	d	$(0, d)$
11	$abra$	$(8, a)$

such that $s(n_{i-1}+1, n_i) = d(s(n_{j-1}+1, n_j))$. Recall that $s(n_{-1}+1, n_0) = \lambda$. The output of the encoder for the j-th word is a token with two fields. The first field encodes $\pi(j)$ and the second field encodes the code of a symbol s_{n_j} (e.g., the ASCII code). For example, consider the sequence "*abracadabraabracadabra*". The LZ78 incremental parsing parses it into $a \cdot b \cdot r \cdot ac \cdot ad \cdot ab \cdot ra \cdot abr \cdot aca \cdot d \cdot abra \cdot$. Table 5.1 shows how this sequence is encoded in LZ78.

The encoder of LZW [61] is designed to eliminate the second field of a token because it worsens compression performance. Initially, the first 256 positions in the dictionary $\mathcal{D}$ are occupied by the respective symbols in $\mathcal{A}$. A string I is initialized to the null string λ. Then, the encoder reads the current input symbol and appends it to I. If the encoder finds I in $\mathcal{D}$, the next symbol is concatenated to I to form a two-symbol string. Then, if the encoder finds I in $\mathcal{D}$, the next symbol is concatenated to I to form a three-symbol string, and so on. When I cannot be found in $\mathcal{D}$, the encoder adds I to the next available position in $\mathcal{D}$, encodes the position of the word $d(I)$ in $\mathcal{D}$, and then initialize I to λ. This process continues until the end of s is reached. Since words added to $\mathcal{D}$ get only one symbol longer each step, the encoder of LZW slowly adapts itself to the sequence s. Table 5.2

Table 5.2. Example of LZW for the sequence "*abracadabraabracadabra*".

I	Output	Entry	Phrase Added to $\mathcal{D}$
a	97	256	ab
b	98	257	br
r	114	258	ra
a	97	259	ac
c	99	260	ca
a	97	261	ad
d	100	262	da
ab	256	263	abr
ra	258	264	raa
abr	263	265	$abra$
ac	259	266	aca
ad	261	267	ada
$abra$	265	–	–

shows the LZW parsing of s and the words added to the dictionary $\mathcal{D}$ when it is applied to $s = abracadabraabracadabra$.

The encoder of LZMW [47] processes a given sequence s as follows. The encoder reads the current input symbol, concatenates it to I, searches the dictionary $\mathcal{D}$ for I in the same way as LZW. The concatenation is repeated until I cannot be found in $\mathcal{D}$. Then, $I := d(I)$. Let $\widetilde{I}$ be the previous I. Then, the encoder adds the concatenation of $\widetilde{I}$ and I, i.e., $\widetilde{I}I$, to the next available position in $\mathcal{D}$, outputs the position of the word I in $\mathcal{D}$, $\widetilde{I} := I$, and initializes I to λ. This process continues until the end of the sequence s is reached. Since words added to $\mathcal{D}$ can grow by more than one symbol each step, the encoder of LZMW adapts itself to the sequence s faster than that of LZW. Table 5.3 shows the LZMW parsing of s and the words added to the dictionary $\mathcal{D}$ when it is applied to $s = alfeatsalfalfaalfeatsalfalfa$.

Table 5.3. Example of LZMW for the sequence "$alfeatsalfalfaalfeatsalfalfa$".

I	Output	Entry	Phrase Added to $\mathcal{D}$
a	97	–	–
l	108	256	al
f	102	257	lf
e	101	258	fe
a	97	259	ea
t	116	260	at
s	115	261	ts
al	256	262	sal
f	102	263	alf
alf	263	264	$falf$
a	97	265	$alfa$
alf	263	266	$aalf$
ea	259	267	$alfea$
ts	261	268	$eats$
$alfa$	265	269	$tsalfa$
lf	257	270	$alfalf$
a	97	271	lfa

5.3 Müller-Schimpfky Scheme

In this section, we briefly describe the best scheme among those presented in [41], called the Müller-Schimpfky scheme.

Let $s = p_n^{k_n} p_{n-1}^{k_{n-1}} \cdots p_1^{k_1}$ be the output of T-decomposition. Then, s can be represented by a T-prefix matrix, a literal vector, and a set of T-expansion parameters (See Fig. 5.1). The concatenation of all columns of the T-prefix matrix yields the following sequence (See Fig. 5.2),

$$k_1^{(2)}, \ldots, k_1^{(n)}, k_2^{(3)}, \ldots, k_2^{(n)}, \ldots, k_{n-1}^{(n)}. \tag{5.1}$$

In general, most T-expansion parameters take "1", and hence only the elements unequal to "1" are encoded by a fixed-length code. On the other hand, since the T-prefix matrix is sparse, an arithmetic coder encodes run lengths of 0's in the

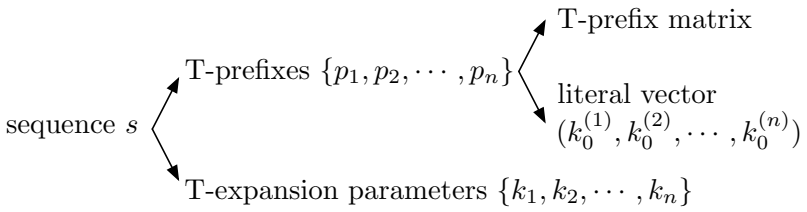


Figure 5.1. Illustration of the Müller-Schimpfky scheme (I).

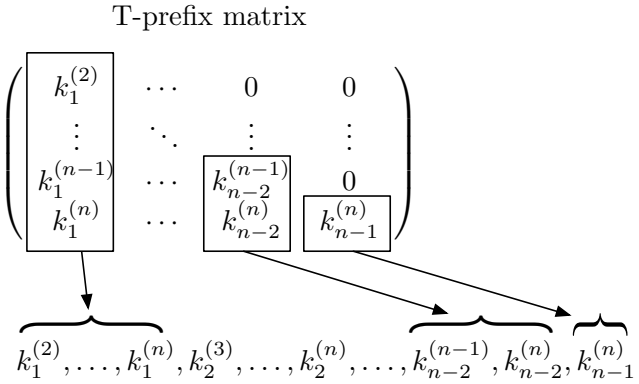


Figure 5.2. Illustration of the Müller-Schimpfky scheme (II).

sequence given by (5.1). The literal vector is also encoded by another arithmetic encoder.

The scheme has the following two defects.

- $\mathrm{O}(n^2)$ elements of the T-prefix matrix have to be encoded.
- The literal vector with n bytes is encoded separately from the T-prefix matrix.

Those defects worsen compression ratio. As shown in Table 5.4, this scheme is inferior to the UNIX *compress* in 12 files out of 18.

Instead, let us consider the following method using a code tree of a simple T-code set. Each leaf node in the code tree can be indexed systematically as shown in Fig. 5.3 using the contiguous range index conversion [14]. The number of leaf nodes of $\mathcal{S}_i$ is $2^i(\#\mathcal{A}-1)+1$. So, the T-prefix p_i can be represented as an integer of $(i+\log_2 \#\mathcal{A})$ bits. The T-complexity of s with length N is $t = \mathrm{li}(\bar{H}N)$, where $\bar{H}$ is in the range of 1.65 to 1.95 for typical English texts [55]. The sequence s can be

Table 5.4. Compression ratios for the Calgary Corpus achieved by the UNIX *gzip*, *bzip2*, *compress*, the Müller-Schimpfky (M-S) scheme, and our scheme (A, B, and C) described in Section 5.4.

File	gzip	bzip2	comp.	M-S	A	B	C
bib	0.315	0.247	0.418	0.500	0.348	0.376	0.392
book1	0.408	0.303	0.413	0.550	0.410	0.393	0.400
book2	0.338	0.258	0.411	0.510	0.381	0.385	0.375
geo	0.669	0.556	0.760	0.691	0.609	0.635	0.640
news	0.384	0.315	0.487	0.545	0.428	0.431	0.436
obj1	0.480	0.501	0.653	0.635	0.543	0.552	0.567
obj2	0.331	0.310	0.521	0.495	0.412	0.443	0.455
paper1	0.350	0.311	0.472	0.555	0.437	0.448	0.440
paper2	0.362	0.305	0.440	0.543	0.428	0.426	0.417
paper3	0.389	0.340	0.476	0.571	0.462	0.462	0.453
paper4	0.417	0.390	0.524	0.589	0.479	0.497	0.494
paper5	0.418	0.405	0.550	0.604	0.512	0.505	0.513
paper6	0.347	0.323	0.491	0.558	0.440	0.445	0.452
pic	0.110	0.097	0.121	0.120	0.107	0.107	0.113
progc	0.335	0.317	0.483	0.557	0.431	0.435	0.445
progl	0.227	0.217	0.379	0.340	0.310	0.327	0.342
progp	0.228	0.217	0.389	0.401	0.318	0.327	0.347
trans	0.203	0.191	0.408	0.376	0.292	0.313	0.361

parsed as $s = p_1 p_2 \cdots p_t$ by the forward T-decomposition for simple T-codes. Using the contiguous range index conversion, s can be encoded to a series of integers, whose total size is at least

$$\sum_{i=1}^{t} (i + \log_2 \#\mathcal{A}) = \frac{t(t+1)}{2} + t \log_2 \#\mathcal{A} \text{ [bits]}. \tag{5.2}$$

When $\#\mathcal{A} = 256, N = 10^5$, and $\bar{H} = 1.7$, Eq. (5.2) is about $1.5 \times 10^7 \gg N$. However, if a small share of the nodes in the code tree is indexed in some way, we can compress s. This idea is realized in the next section.

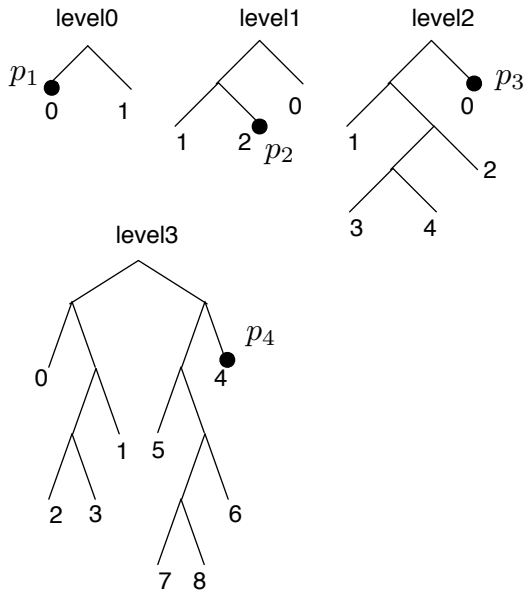


Figure 5.3. Indexing leaf nodes in the code tree.

5.4 New Compression Scheme Based on T-codes

In this section, we propose a new data compression scheme based on T-codes. Our encoder creates phrases, which have a recursive structure similar to T-codes, and they are added to a dictionary $\mathcal{D}$. Initially, the dictionary consists of all the symbols in $\mathcal{A}$. Let I be the longest prefix of the unparsed part of s that coincides with a phrase included in $\mathcal{D}$. Then, s is parsed by I, and $\mathcal{D}$ is updated using I. Repeating this operation, s is represented as a sequence of I's. If the current I differs from all preceding I's, then a new T-prefix, p_j, is assigned to the current I. Hence, the parsing of s can also be represented as a sequence of p_j's.

We considered three dictionary updating rules, called Methods A, B, and C. In Method A, every suffix w of the parsed part of s satisfying (5.3) is added to the dictionary $\mathcal{D}$.

$$p_1^{k'_1} p_2^{k'_2} \cdots p_{j-1}^{k'_{j-1}} p_j, \text{ where } k'_l \geq 0,\ 1 \leq l \leq j-1. \tag{5.3}$$

Note that T-prefixes $p_l^{k'_l}$ in (5.3) have the opposite order compared with (1.4).

Hence, in Methods B and C, we consider every suffix w of the parsed part of s satisfying (5.4), where m is the current largest index of p_j.

$$p_m^{k'_m} \cdots p_{j+2}^{k'_{j+2}} p_{j+1}^{k'_{j+1}} p_j, \text{ where } k'_l \geq 0,\ l = j+1, \cdots, m. \tag{5.4}$$

Every w is added to the dictionary $\mathcal{D}$ in Method B. But, in Method C, wx instead of w is added to $\mathcal{D}$, where x is the next symbol of w in s. The idea of appending x to w was adopted from LZW [61]. Note that our encoding scheme with Method C is decodable in the same way as LZW. In all three methods, if w ends with $p_j^h,\ h \neq 1$, then p_j^h is added to $\mathcal{D}$. This additional rule makes it possible to encode repetitions of strings efficiently.

Our scheme consists of the following three phases.

- In Phase 1, the encoding scheme sequentially parses a given sequence s and outputs ξ's, each of which represents the position of I in the dictionary $\mathcal{D}$.
- In Phase 2, each ξ is converted to an integer η so that the distribution of η's becomes almost monotonically decreasing.
- In Phase 3, η is encoded to a binary sequence by using an arithmetic coder.

Phase 1 is described formally as follows, where strings r and s represent the parsed part and the unparsed part of a given sequence, respectively, and λ is the null string.

Phase 1

S1 (Initialization)

Let s be a given sequence.

A dictionary $\mathcal{D}$ is initialized to all the symbols of $\mathcal{A}$.

$I := \lambda$, $r := \lambda$, $\alpha := 1$.

S2 Let x be the first symbol of s.

S3 (Parsing)

If $Ix \in \mathcal{D}$, then $I := Ix$, remove x from the head of s, go back to S2.

Otherwise, output the integer ξ which represents the position of I in $\mathcal{D}$.

If I is not equal to any T-prefix $p_l,\ 1 \leq l \leq \alpha - 1$, then $p_\alpha := I$, $\alpha := \alpha + 1$.

S4 (Dictionary Updating)

$r := rI$, and represent r as a sequence of T-prefixes.

If r ends with $p_\alpha^h,\ h \neq 1$, then add p_α^h to $\mathcal{D}$, go to S5.

Otherwise,

[Method A] consider every suffix w of r that satisfies (5.3).

If $w \notin \mathcal{D}$, then add w to $\mathcal{D}$.

[Method B] consider every suffix w of r that satisfies (5.4).

If $w \notin \mathcal{D}$, then add w to $\mathcal{D}$.

[Method C] consider every suffix w of r that satisfies (5.4).

If $wx \notin \mathcal{D}$, then add wx to $\mathcal{D}$.

S5 If $s = \lambda$, then exit.

Otherwise, $I := \lambda$. Go back to S2.

Example

We show an example how Phase 1 with Method A processes $s = s_1^{13} = alfeatsalfalf$. Assume that all one-byte characters are assigned to integers 0–255 in the dictionary $\mathcal{D}$ (e.g., a is 97, e is 101, f is 102, and so on), and $s_1^7 = alfeats$ has already been processed as shown in Table 5.5. Then, $s_8^{13} = alfalf$ is processed as follows.

1. $I := alf$. $j := 7$. Output $\xi = 258$.

2. Since I is not equal to any T-prefix $p_i,\ 1 \leq i \leq 6$, I is assigned to p_7.

Table 5.5. Example of Phase 1 with Method A for $s_1^7 = alfeats$.

I	ξ	p_j	Entry	Phrase Added to $\mathcal{D}$
a	97	p_1	–	–
l	108	p_2	256	$p_1p_2 = al$
f	102	p_3	257	$p_2p_3 = lf$
			258	$p_1p_2p_3 = alf$
e	101	p_4	259	$p_3p_4 = fe$
			260	$p_2p_3p_4 = lfe$
			261	$p_1p_2p_3p_4 = alfe$
a	97	p_1	–	–
t	116	p_5	262	$p_1p_5 = at$
s	115	p_6	263	$p_5p_6 = ts$
			264	$p_1p_5p_6 = ats$

3. The parsed part of s is represented as $r := s_1^{10} = p_1p_2p_3p_4p_1p_5p_6p_7$. Then, three phrases satisfying (5.3), i.e., $p_6p_7, p_5p_6p_7, p_1p_5p_6p_7$, are added, as entries 265–267, respectively, to the dictionary $\mathcal{D}$.

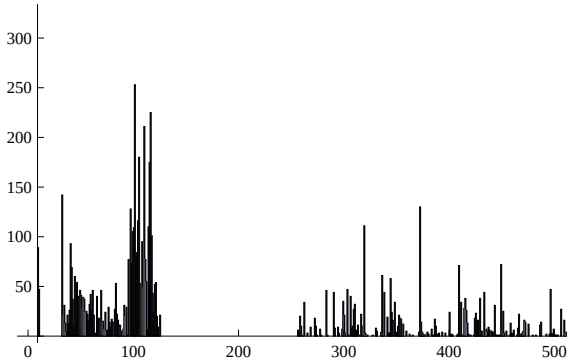
4. Again, the next I is alf. Output $\xi = 258$.

5. $r := s_1^{13} = p_1p_2p_3p_4p_1p_5p_6p_7p_7$.

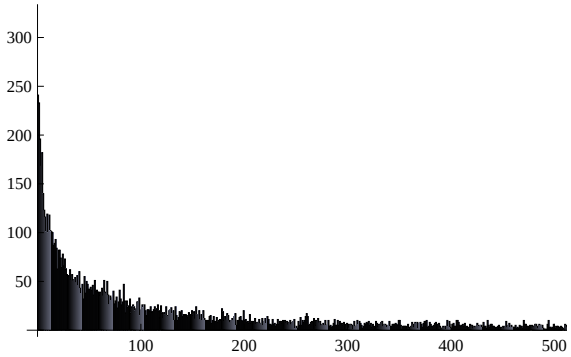
6. $h := 2$. $p_7^2 = alfalf$ is added as entry 268 to $\mathcal{D}$.

In Phase 2, a sequence $(\xi_1, \xi_2, \xi_3, \ldots)$ obtained in Phase 1 is converted to a sequence $(\eta_1, \eta_2, \eta_3, \ldots)$ as follows. For $l = 1, 2, \ldots$, we sort all numbers appeared in $\xi^{l-1} = (\xi_1, \xi_2, \ldots, \xi_{l-1})$ in the descending order of frequencies, and other numbers not appeared in ξ^{l-1} are attached to the sorted list in the increasing order of numbers. Then, η_l is the position of ξ_l in the above list. Note that the maximum of η_l, say M_{η_l}, is bounded by the number of phrases stored in the dictionary $\mathcal{D}$ at the moment ξ_l is outputted from Phase 1. As a result, the distribution of η's becomes almost monotonically decreasing as shown in Figs. 5.4 and 5.5.

In Phase 3, first, each η_l is encoded to a codeword of a Start-Step-Stop code



(a) Distribution of ξ's.

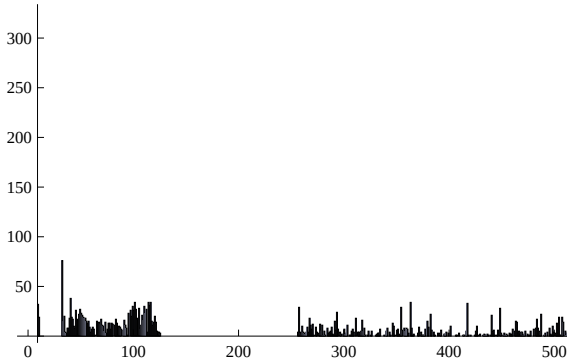


(b) Distribution of η's.

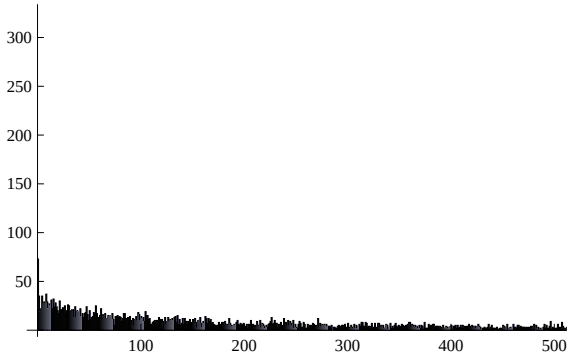
Figure 5.4. Distributions of ξ's and η's for 'progc' in the Calgary Corpus for Method A.

Table 5.6. Start-Step-Stop code with $start = 1, step = 2, stop = 7$.

Number	First	Second
0	1	0
1	1	1
2	01	000
3	01	001
⋮	⋮	⋮
9	01	111
10	001	00000
11	001	00001
⋮	⋮	⋮
41	001	11111
42	000	0000000
43	000	0000001
⋮	⋮	⋮
169	000	1111111



(a) Distribution of ξ's.



(b) Distribution of η's.

Figure 5.5. Distributions of ξ's and η's for 'progc' in the Calgary Corpus for Method C.

[47] with three parameters, i.e., $start, step$, and $stop$[*1]. Each codeword consists of two parts. The first part represents an integer, say m, as a unary number. That is, m is encoded to m "0's" terminated by a single "1". However, when $m = (stop - start)/step$, m is exceptionally encoded to m "0's". The length of the second part is given by $l_m = start + m \times step$. The value of $stop$ can be determined by $start, step$, and M_{η_l}.

Next, an arithmetic coder encodes the first part with $(stop - start)/step$ frequency tables, where the i-th frequency table is used to encode the i-th bit of the first part. For each m represented by the first part, we separately prepare l_m frequency tables in order to encode the second part by another arithmetic coder. The i-th frequency table is used to encode the i-th bit of the second part. Each time after η_l is encoded, frequency counters are updated. For the purpose of decoding,

[*1] We used $start = 1$ and $step = 1$.

file size (or the number of η's) is written to the header of the output file.

5.5 Experiments

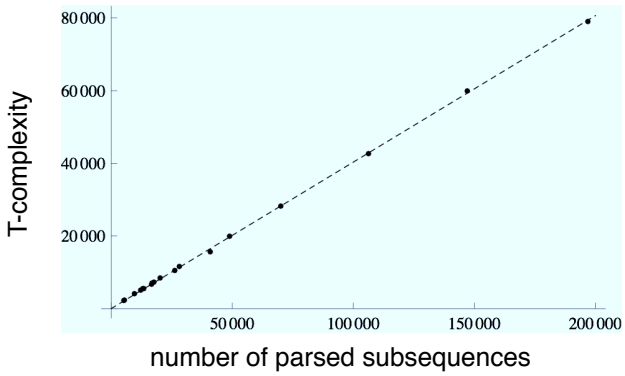
First, we compared the proposed scheme with the Müller-Schimpfky scheme and the UNIX *compress* on the Calgary Corpus. As shown in Table 5.4, the proposed scheme can compress all the files in the Calgary Corpus more efficiently than the Müller-Schimpfky scheme and the UNIX *compress*. On the whole, Method A is superior to Methods B and C. However, as shown in Table 5.4, the proposed scheme is inferior to the UNIX *bzip2*, which is based on the block sorting scheme [1]. In the case of 'book1', 'geo', and 'pic', the proposed scheme can compete with the UNIX *gzip*, which is based on the LZ77 scheme.

Since the parsing procedures of our scheme are not exactly the same as T-decomposition, we examined how each of our parsing procedures is correlated with T-decomposition. Figure 5.6 shows the relation between the T-complexity and the number of parsed subsequences ($\#\xi$) for the files in the Calgary Corpus. We can see a linear relation between $\#\xi$ and the T-complexity. Figure 5.7 shows the relation between the dictionary size and $\#\xi$. We can also see a linear relation between $\#\xi$ and the dictionary size.

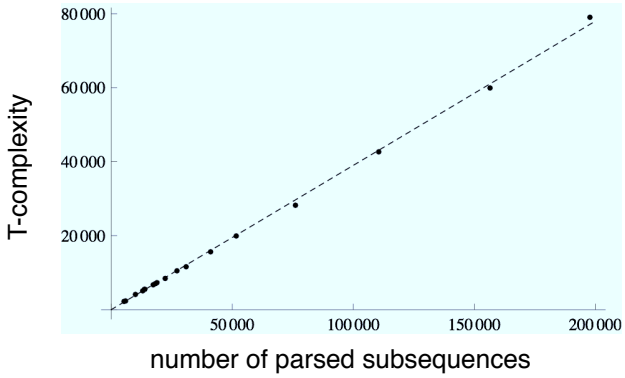
Next, we carried out an experiment on the file of 10^5 bytes, called 'alphabet.txt'[*2], which consists of repetitions of the alphabet, i.e., $abc\cdots xyzabc\cdots xyz\cdots$. Our scheme outperformed than the UNIX *compress* as shown in Table 5.7. In particular, our schemes with Methods A and B compressed this file more efficiently than the UNIX *gzip*.

Finally, we compared our schemes with LZW and LZMW [47], which are used in Phase 1 but the same coding is used in Phases 2 and 3. As shown in Table 5.8, LZMW defeated Methods A, B, and C, while the proposed scheme was better than

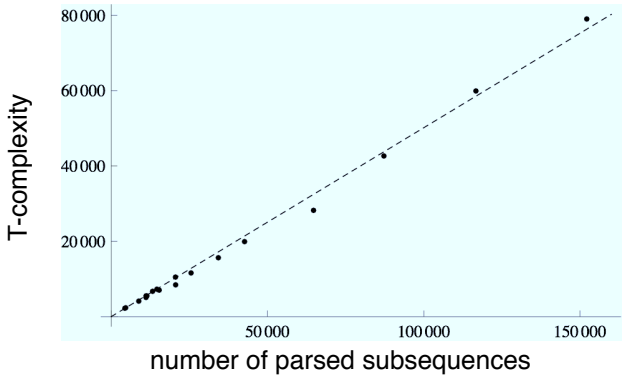
[*2]This file is available at http://corpus.canterbury.ac.nz/descriptions/artificl/alphabet.html.



(a) Method A

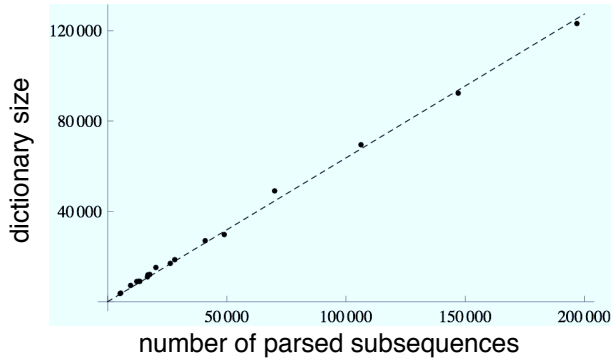


(b) Method B

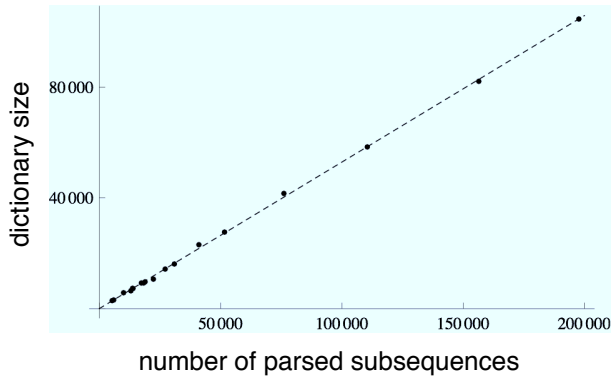


(c) Method C

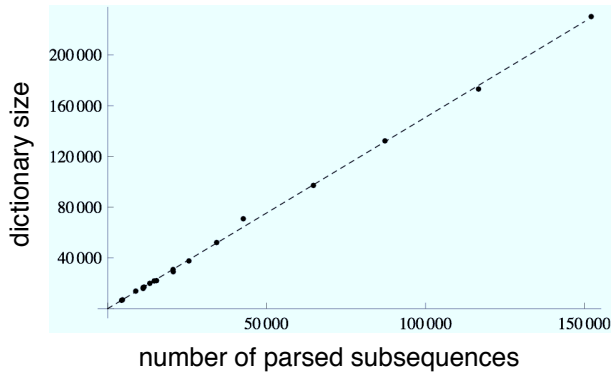
Figure 5.6. T-complexity (vertical axis) plotted against the number of parsed subsequences (horizontal axis). The broken line shows the result of linear regression of the data measurement values.



(a) Method A



(b) Method B



(c) Method C

Figure 5.7. Dictionary size (vertical axis) plotted against the number of parsed subsequences (horizontal axis). The broken line shows the result of linear regression of the data measurement values.

Table 5.7. Compressed file size [bytes] of 'alphabet.txt' in the case of the UNIX *gzip*, *bzip2*, *compress*, and our scheme (A, B, and C).

gzip	bzip2	compress	A	B	C
315	131	3053	68	305	2459

Table 5.8. Compression ratios for the Calgary Corpus when LZW and LZMW are used in Phase 1 but the same coding is used in Phases 2 and 3. The compression ratios achieved by our scheme (A, B, and C) are shown again.

File	LZW	LZMW	A	B	C
bib	0.401	0.330	0.348	0.376	0.392
book1	0.395	0.353	0.410	0.393	0.400
book2	0.378	0.314	0.381	0.385	0.375
geo	0.632	0.588	0.609	0.635	0.640
news	0.444	0.377	0.428	0.431	0.436
obj1	0.573	0.521	0.543	0.552	0.567
obj2	0.469	0.372	0.412	0.443	0.455
paper1	0.450	0.382	0.437	0.448	0.440
paper2	0.421	0.367	0.428	0.426	0.417
paper3	0.456	0.410	0.462	0.462	0.453
paper4	0.496	0.452	0.479	0.497	0.494
paper5	0.521	0.465	0.512	0.505	0.513
paper6	0.463	0.396	0.440	0.445	0.452
pic	0.113	0.107	0.107	0.107	0.113
progc	0.454	0.383	0.431	0.435	0.445
progl	0.360	0.278	0.310	0.327	0.342
progp	0.363	0.274	0.318	0.327	0.347
trans	0.382	0.268	0.292	0.313	0.361

LZW on the whole.

5.6 Discussions

5.6.1 Comparison of Methods A, B, and C

In general, when the dictionary size increases, I becomes longer, $\#\xi$ decreases, and compression ratio improves.

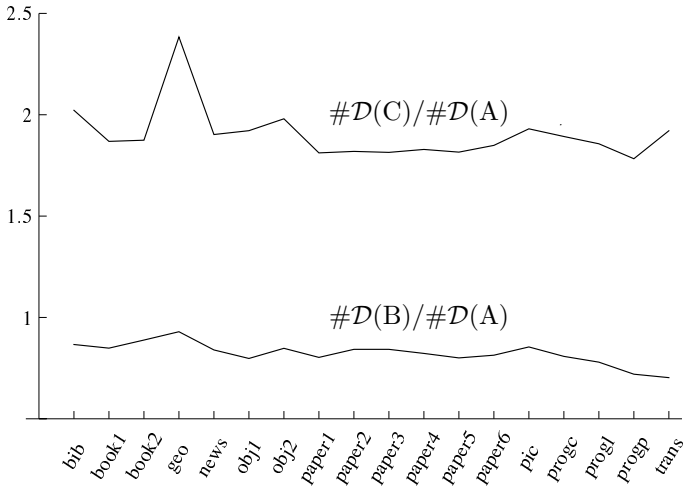


Figure 5.8. Comparison of the dictionary size.

Since the form satisfying (5.3) is more likely to fit the suffix of the parsed part of a sequence than the form satisfying (5.4), Method A adds more phrases to its dictionary than does Method B. Figure 5.8 shows $\#\mathcal{D}(\mathrm{C})/\#\mathcal{D}(\mathrm{A})$ and $\#\mathcal{D}(\mathrm{B})/\#\mathcal{D}(\mathrm{A})$, where $\#\mathcal{D}(\mathrm{A})$, $\#\mathcal{D}(\mathrm{B})$, and $\#\mathcal{D}(\mathrm{C})$ represent the dictionary size for Methods A, B, and C, respectively. We observe that $\#\mathcal{D}(\mathrm{B}) < \#\mathcal{D}(\mathrm{A})$. Figure 5.9 shows $\#\xi(\mathrm{C})/\#\xi(\mathrm{A})$ and $\#\xi(\mathrm{B})/\#\xi(\mathrm{A})$, where $\#\xi(\mathrm{A}), \#\xi(\mathrm{B})$, and $\#\xi(\mathrm{C})$ are the number of parsed subsequences for Methods A, B, and C, respectively. We observe that $\#\xi(\mathrm{B}) > \#\xi(\mathrm{A})$. So, Method A can attain better compression than Method B.

Furthermore, Method A outperforms Method C although $\#\mathcal{D}(\mathrm{A}) < \#\mathcal{D}(\mathrm{C})$ and $\#\xi(\mathrm{A}) > \#\xi(\mathrm{C})$, as shown in Figs. 5.8 and 5.9. This is because the distribution of η's for Method A is more biased than that for Method C. In Method C, the number of appearances of each phrase as I is limited by the number of alphabet symbols. Thus, the distribution of η's becomes flat and worsens the compression of η's.

5.6.2 Single-Pass Realization

In Section 5.4, we explained the proposed scheme by dividing it into three phases. But, we note that encoding in each phase is sequential. So we can easily realize

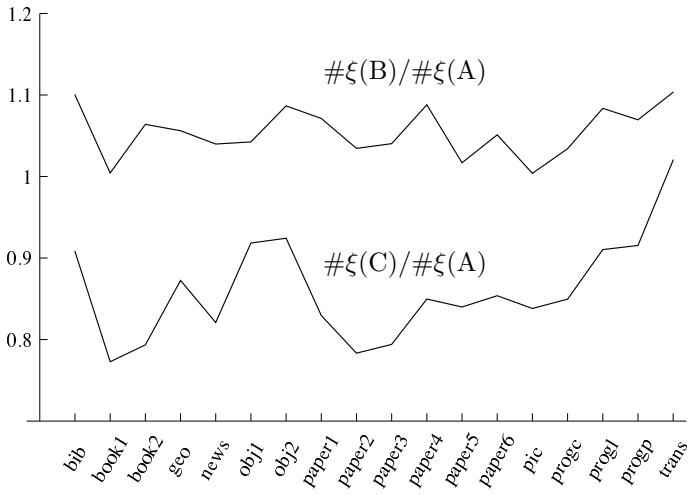


Figure 5.9. Comparison of $\#\xi$.

the proposed scheme as a single-pass scheme.

Furthermore, encoding in Phase 1 can be implemented independently of Phases 2 and 3. So, Phase 1 can easily be combined with any other coding scheme of integers instead of Phases 2 and 3.

5.6.3 Universality of the Proposed Scheme with Method C

The universality of the proposed scheme with Method C for stationary ergodic sources can be proved following a line of argument alike to the universality proof of LZ78 [3].

Let c and d be the number of parsed subsequences ($\#\xi$) and the dictionary size, respectively. On the basis of Figs. 5.6(c) and 5.7(c), we assume that there exist constants α, β such that $d \leq \alpha c$ and $c \leq \beta t$, where t is the T-complexity. Let N be the length of a sequence $s^N = s_1 s_2 \cdots s_N \in \mathcal{A}^N$. Then compression ratio $(c \log_2 d)/N$ is evaluated as follows.

$$\begin{aligned} \frac{c \log_2 d}{N} &\leq \frac{c \log_2 (\alpha c)}{N} \\ &= \frac{c \log_2 c}{N} + (\log_2 \alpha)\frac{c}{N} \\ &\leq \frac{c \log_2 c}{N} + \beta(\log_2 \alpha)\frac{t}{N}. \end{aligned}$$

The second term converges to 0 as $N \to \infty$ because the T-complexity is an increasing concave function of N [24,59]. For a fixed integer k, define the probability distribution Q_k on S^N by

$$Q_k(s^N) = \prod_{j=1}^{N} \Pr(s_j | s_{j-k}^{j-1}),$$

where $s_i^j \equiv (s_i, s_{i+1}, \ldots, s_j)$, $i \le j$, and $s_{-i}^j = s_1^j$ for any $i \ge 0$. Suppose that s^N is parsed into c subsequences, $y_1, y_2, \ldots, y_c$. Let ν_i be the index of the start of y_i, i.e., $y_i = s_{\nu_i}^{\nu_{i+1}-1}$. For each $i = 1, 2, \ldots, c$, define $r_i = s_{\nu_i-k}^{\nu_i-1}$. Let c_{lr} be the number of subsequences y_i with length l and preceding state $r_i = r$ for $l \ge 1$ and $r \in \mathcal{A}^k$. Then, $\sum_{l,r} c_{lr} = c$ and $\sum_{l,r} l c_{lr} = N$ hold.

Following the proof of Ziv's inequality in [3], we have

$$\begin{aligned}
\log_2 Q_k(s^N) &= \sum_{i=1}^{c} \log_2 Q_k(y_i|r_i) \\
&= \sum_{l,r} \sum_{i:\ |y_i|=l, r_i=r} \log_2 Q_k(y_i|r_i) \\
&= \sum_{l,r} c_{lr} \sum_{i:\ |y_i|=l, r_i=r} \frac{1}{c_{lr}} \log_2 Q_k(y_i|r_i) \\
&\le \sum_{l,r} c_{lr} \log_2 \left(\frac{\sum_{i:\ |y_i|=l, r_i=r} Q_k(y_i|r_i)}{c_{lr}} \right).
\end{aligned}$$

Since the number of duplications of y_i is limited by $\#\mathcal{A}$ in Method C, $\sum_{i:\ |y_i|=l, r_i=r} Q_k(y_i|r_i) \le \#\mathcal{A}$. Hence,

$$\begin{aligned}
\log_2 Q_k(s^N) &\le c \log_2 \#\mathcal{A} - \sum_{l,r} c_{lr} \log_2 c_{lr} \\
&= c \log_2 \#\mathcal{A} - c \log_2 c - c \sum_{l,r} \frac{c_{lr}}{c} \log_2 \frac{c_{lr}}{c}.
\end{aligned}$$

Denoting $\pi_{lr} = \frac{c_{lr}}{c}$, we have

$$\sum_{l,r} \pi_{lr} = 1, \quad \sum_{l,r} l\pi_{lr} = \frac{N}{c}.$$

Let us define random variables U, V such that $\Pr(U = l,\ V = r) = \pi_{lr}$, $\mathrm{E}[U] = \frac{N}{c}$. Then,

$$\begin{aligned} -\frac{\log_2 Q_k(s^N)}{N} &\geq -\frac{c\log_2 \#\mathcal{A}}{N} + \frac{c\log_2 c}{N} - \frac{c}{N}H(U,V) \\ &\geq -\frac{c\log_2 \#\mathcal{A}}{N} + \frac{c\log_2 c}{N} - \frac{c}{N}(H(U) + H(V)) \\ &\geq \frac{c\log_2 c}{N} - \frac{c}{N}(H(U) + (k+1)\log_2 \#\mathcal{A}). \end{aligned}$$

Since $H(U) \leq (\mathrm{E}[U]+1)\log_2(\mathrm{E}[U]+1) - \mathrm{E}[U]\log_2 \mathrm{E}[U]$ holds [3],

$$H(U) \leq \log_2 \frac{N}{c} + \left(\frac{N}{c} + 1\right)\log_2\left(\frac{c}{N} + 1\right).$$

Recalling the assumption $c \leq \beta t$, we have

$$-\frac{\log_2 Q_k(s^N)}{N} \geq \frac{c\log_2 c}{N} - \varepsilon_k(N),$$

where $\varepsilon_k(N) \to 0$ as $N \to \infty$. Thus, for a satisfactory ergodic process,

$$\begin{aligned} \limsup_{N\to\infty} \frac{c\log_2 c}{N} &\leq \lim_{N\to\infty}\left[-\frac{\log_2 Q_k(S^N)}{N}\right] \\ &= \lim_{N\to\infty}\left[-\frac{1}{N}\sum_{j=1}^{N}\log_2 \Pr(S_j|S_{j-k}^{j-1})\right] \\ &\to -\mathrm{E}[\log_2 \Pr(S_j|S_{j-k}^{j-1})] \\ &= H(S_j|S_{j-k}^{j-1}) \end{aligned}$$

Since $H(S_j|S_{j-k}^{j-1})$ converges to the entropy rate of the process as k becomes large for $j = k+1$, the proposed scheme with Method C is universal.

5.7 Conclusions

We reviewed several schemes in the LZ78 family, i.e., LZ78, LZW, and LZMW, in Section 5.2. In Section 5.3, we briefly described the Müller-Schimpfky scheme. In the Müller-Schimpfky scheme, a given sequence is represented by a T-prefix matrix, a literal vector, and a set of T-expansion parameters, and then the sequence obtained from the concatenation of all columns of the T-prefix matrix is encoded by an arithmetic coder. The Müller-Schimpfky scheme has the following defects. The first defect is that $\mathrm{O}(n^2)$ elements of the T-prefix matrix have to be encoded, where n is the number of T-prefixes. The second defect is that the literal vector with n bytes is encoded separately from the T-prefix matrix. To overcome the defects, we proposed the data compression scheme based on a dictionary method such that all phrases added to a dictionary have a recursive structure similar to T-codes in Section 5.4. We considered the three dictionary updating rules, Methods A, B, and C.

The compression performance of our scheme was compared with other schemes in Section 5.5. Although our scheme is inferior to the UNIX *bzip2* and *gzip* on the whole, it succeeded in compressing the Calgary Corpus more efficiently than the Müller-Schimpfky scheme and the UNIX *compress*. The experiment showed that the number of parsed subsequences of our scheme has a linear relation with the T-complexity. On the whole, Method A outperformed Methods B and C. Methods A and B were superior to Method C with respect to compressing simple periodic sequences. However, LZMW outperformed Methods A and B because the dictionary size for LZMW is larger than that for Methods A and B.

In Section 5.6, we first explained the reason why Method A outperformed Method B and C. Next, we showed how to realize a single-pass scheme. Finally, we theoretically proved that Method C is universal for stationary ergodic sources under the assumptions that the number of ξ's and the dictionary size are propor-

tional to the T-complexity, which is experimentally validated.

Chapter 6

Application of the T-complexity to Randomness Testing for Cryptography

6.1 Introduction

6.1.1 Importance of Randomness Testing in Cryptography

A symmetric-key cryptosystem is ideal when an exhaustive key search is the best way to break it. If keys are not sampled equiprobably from a specified key space, such a symmetric-key cryptosystem is regarded as insecure. Hence, keys used in symmetric cryptosystems are generated by random number generators. Random number generators are also used to generate public-key parameters such as RSA-moduli of the RSA public-key cryptosystem [40]. So, how can we obtain random numbers or random bit sequences? A series of coin tosses may provide us with a random bit sequence. But, can we guarantee the fairness of the coin? Since coin-tossing obeys Newton's laws of motion, the final outcome can be predicted from initial conditions under certain circumstances [4]. Even if the coin is assumed to

be fair, such a generation is too slow to be used in cryptosystems. Instead of coin-tossing, we may use physical chaotic processes such as radioactive decay, thermal noise, metastability, atmospheric noise, and so on. But, can we be sure that the output is not biased and independent of the previous outputs[*1]? The only way of telling is to carry out statistical randomness tests on the obtained bit sequences.

Random number generators are divided into two categories: the truly (physical) random number generator and the pseudo-random number generator. In general, a physical random number generator is expensive, slow compared to computation or communication speed, and does not have reproducibility. These demerits limit applications in cryptosystems. If a physical random number generator is used as a keystream generator of a stream cipher, the whole keystream has to be shared between two parties using a secure channel. But, such a keystream distribution via a secure channel is affordable for only specialized purposes in the military or the diplomatic services. Therefore, pseudo-random number generators are usually used in cryptosystems.

A pseudo-random number generator used in cryptosystems is a polynomial time algorithm that takes a short seed with length k bits as input and expands it into a long sequence with length l ($\gg k$) bits deterministically that is indistinguishable from a truly random sequence to anyone who does not know the seed, and hence it has reproducibility. In a strict sense, the output of a pseudo-random number generator is not random because the output is sampled from only 2^k l-bit sequences, but sharing the same seed between two parties is easy.

In order to check the security of a cryptosystem, first of all, we have to apply a package of statistical randomness tests to the pseudo-random number generator used in it. Random numbers used in cryptosystems are required to have polynomial-time unpredictability as well as the properties listed in Section 1.5.

[*1] The bias can be removed by de-skewing techniques like von Neumann's method [44] or hash functions. This removal of bias is called a post processing.

Let us consider Mersenne twister (MT) [39]. MT is a high-quality pseudo-random number generator, which generates M-sequences (maximum length linear feedback shift register sequence), and is widely used in computer simulations. MT provides a period of $2^{19937} - 1$ and 623-dimensional equidistribution up to 32-bit accuracy, and it passes numerous randomness tests including the Diehard tests. However, we can predict future outputs completely by observing a relatively small number of outputs and using the Berlekamp-Massey algorithm [38]. Therefore, the evaluation of random numbers for cryptosystems should be stricter than that for other applications. Unpredictability of a pseudo-random number generator can be measured by the complexity of the output sequence. The NIST test suite includes some randomness tests based on computable complexity measures. This is the reason why the NIST test suite is most widely used in cryptographic fields. Each randomness test included in the NIST test suite was designed to detect a specific defect shown in Table 6.1 [49]. The NIST test suite enables us to examine the output of a pseudo-random number generator from many different perspectives.

6.1.2 NIST Test Suite

History

In 2001, the NIST released NIST SP 800-22 [42] describing the NIST test suite. When the NIST test suite was released, it included sixteen core randomness tests, but currently consists of fifteen core randomness tests, which can be viewed as 188 statistical tests as shown in Table 6.2. The default input parameters used in the NIST test suite are also shown in Table 6.2.

The NIST test suite is a widely used package of statistical tests in the field of cryptography. For example, it was used in the following projects.

- AES project: a project undertaken by the NIST to select a block cipher suitable for replacing the Data Encrption Standard (DES) [9].

Table 6.1. Characteristics of randomness tests included in the NIST test suite.

Test Name	Defect Detected
Frequency (Monobits)	Too many zeros or ones in the entire sequence.
Test for Frequency Within A Block	Too many zeros or ones within M-bit blocks.
Cumulative Sum (Cusum) Test	Too large or too small maximal excursion of a random walk.
Runs Test	Large (small) total number of runs indicates that the oscillation in the sequence is too fast (too slow).
Test For The Longest Run Of Ones In A Block	Deviation of the distribution of long runs of ones.
Random Binary Matrix Rank Test	Deviation of the rank distribution from a corresponding random sequence, due to periodicity.
Discrete Fourier Transform (Spectral) Test	Periodic features in the sequence.
Non-Overlapping (Aperiodic) Template Matching Test	Too many occurrences of non-periodic templates.
Overlapping (Periodic) Template Matching Test	Too many occurrences of m-bit runs of ones.
Maurer's Universal Statistical Test	Compressibility
Approximate Entropy Test	Non-uniform distribution of m-length words.
Random Excursions Test	Deviation from the distribution of the number of cycles having exactly K visits in a cumulative sum random walk to a certain state.
Random Excursions Variant Test	Deviation from the distribution of the total number of visits to a certain state in a cumulative sum random walk.
Serial Test	Non-uniform distribution of m-length words.
Linear Complexity Test	Deviation from the distribution of the linear complexity for finite length (sub)strings.
Lempel-Ziv Complexity Test	More compressed than a truly random sequence.

Table 6.2. Breakdown of the 188 Statistical Tests

Test Name	#P-value	Test ID	Parameter
Frequency (Monobits)	1	1	-
Test For Frequency Within A Block	1	2	128
Cumulative Sum (Cusum) Test	2	3-4	-
Runs Test	1	5	-
Test For The Longest Run Of Ones In A Block	1	6	-
Random Binary Matrix Rank Test	1	7	-
Discrete Fourier Transform (Spectral) Test	1	8	-
Non-Overlapping (Aperiodic) Template Matching Test	148	9-156	9
Overlapping (Periodic) Template Matching Test	1	157	9
Maurer's Universal Statistical Test	1	158	7, 1280
Approximate Entropy Test	1	159	10
Random Excursions Test	8	160-167	-
Random Excursions Variant Test	18	168-185	-
Serial Test	2	186-187	16
Linear Complexity Test	1	188	500

- eSTREAM [8]: a project sponsored by the ECRYPT Network of Excellence to identify promising new stream ciphers.

Despite its reputation, several shortcomings of the NIST test suite have been noted. It was reported in [15, 16, 28, 32] that the DFT test and the Lempel-Ziv complexity test (LZ test) in the NIST test suite have some problems. Furthermore, in [30], the modification of the input size of the approximate entropy test was recommended. The NIST updated some values of parameters for the DFT test and removed the LZ test from the software of the NIST test suite in 2004. But no official explanation was given about the reason why the LZ test was removed. The DFT test with modified parameter values is still not ideal because a more suitable value of a DFT test parameter was derived in [20]. Okutomi *et al.* [45] evaluated the randomness of sequences generated by DES and SHA-1 [11] on the basis of the NIST test suite, and showed that both the overlapping template matching test and Maurer's universal statistical test (Universal test) did not follow the theoretical binomial distribution if DES or SHA-1 can be assumed to be an ideal random number generator. The problem of the overlapping template matching test was caused from inaccurate probability estimation for templates in

the NIST test suite [18]. The accurate probabilities presented in [18] were later incorporated into NIST SP 800-22 Revision 1 [43]. Moreover, the revised Universal test based on the model proposed by Coron [2] resolved the problems of the original Universal test [27]. It was also reported in [17, 46] that the probabilities used in the longest-run-of-ones test in the NIST test suite need to be corrected. Furthermore, Hamano and Yamamoto showed in [20] that the NIST test suite fails to detect non-random sequences with periodic small biases but a randomness test based on all autocorrelation values can detect. Hamano, Sato, and Yamamoto also showed in [23] that the NIST test suite including the linear complexity test (LC test) fails to detect non-random sequences generated by concatenating two different M-sequences with low linear complexity, but a modified LC test can detect it. The defect of the original LC test comes from the fact that the deviation from the ideal value is evaluated only for the last part of the whole linear complexity profile.

In 2008, the NIST released NIST SP 800-22 Revision 1, and officially excluded the LZ test, a randomness test based on the LZ-complexity, from NIST SP 800-22 [43]. After the LZ test is excluded, NIST SP 800-22 includes no randomness test based on a concrete data compression algorithm. In February 2009, the NIST announced on the web that the NIST had discovered a problem with the DFT test and advised disregarding the results of the DFT test without detailed explanation.

Test Procedure

The NIST test suite requires the generation of m (e.g., $m = 10^3$) random binary sequences of length N (e.g., $N = 10^6$). Note that some statistical tests included in the NIST test suite require that each sequence has a length at least 10^6. Each generated sequence is applied to all 188 statistical tests in the NIST test suite (See Table 6.2). Thus, each statistical test produces m P-values.

A statistical test considers that a sequence passes the test if P-value $\geq \alpha$, where

α is the significance level (e.g., $\alpha = 0.01$). For each statistical test, the proportion of sequences that pass the test is computed. Let us denote the proportion by P. P is expected to be $\hat{p} = 1 - \alpha$ if H_0 is true. If the proportion P falls outside of the range

$$\left[\hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}},\ \hat{p} + 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}\right], \tag{6.1}$$

the sequences are considered as non-random. This range was adopted from the three sigma method.

Additionally, the distribution of P-values is examined to check for uniformity of P-values via applying a chi-square goodness-of-fit test on the m P-values. The interval between 0 and 1 is divided into ten sub-intervals $C_i = [0.1(i-1), 0.1i),\ i = 1, 2, \ldots, 10$. Let f_i be the number of P-values falling into the sub-interval C_i. The χ^2-statistic given by

$$\chi^2 = \frac{\sum_{i=1}^{10}\left(f_i - \frac{m}{10}\right)^2}{\frac{m}{10}}$$

is computed. Let us denote the P-value of the χ^2 value by U. If $U \geq 0.0001$, it is treated in the NIST test suite that the P-values distribute uniformly.

However, for $\alpha = 0.01$ and $m = 10^3$, which are the most commonly used values, the probability of type I error is relatively large because

$$\Pr\left\{P \leq \hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}\right\} \approx 0.00328836,$$

and hence, under the assumption that all statistical tests are independent, the probability that all of the 188 P's exceed the threshold $\hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$ is $(1 - 0.00328836)^{188} = 0.538359$. Hence, the probability that the test on the 188 P's is passed is only about 50% even if H_0 is true.

In the NIST report [50] on the evaluation of AES finalists as random number

generators, the P-value of P is used rather than the range given by Eq. (6.1), and if the P-value of P is 0.0001 or more, the test on P is considered to be passed. If we adopt this criterion, the minimum acceptable proportion is revised to 0.976 from 0.9805608 for $\alpha = 0.01$ and $m = 10^3$.

6.1.3 Problems of the NIST LZ Test

The NIST LZ test procedure may be described as follows. Note that the one tailed test is used in the LZ test. This test rejects H_0 only for large positive z values, i.e., small W_{obs} values compared to μ.

The NIST LZ Test

S1 Let s be a given binary sequence of length N.

S2 Let W_{obs} be the LZ-complexity of s.

S3 Compute $z = (\mu - W_{obs})/\sigma$, where $\mu = 69588.2019$ and $\sigma = \sqrt{73.23726011}$ when $N = 10^6$. The values of μ and σ were computed using SHA-1.

S4 Compute P-value$= \frac{1}{2}\mathrm{erf}\left(\frac{z}{\sqrt{2}}\right) = \int_z^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right) \mathrm{d}u$.

The NIST gave no official explanation of the reason why the LZ test was excluded from the NIST test suite. However, the CRYPTREC[*2] technical report [29] listed the problems of the LZ test as follows.

1. The mean μ and variance σ^2 of the LZ-complexity are not derived by theory but are given experimentally.

2. The differences between the theoretically evaluated values of the mean and variance of the LZ-complexity and those experimentally obtained values are not negligible for sequences of practical length (e.g., 10^6).

[*2] CRYPTREC is an abbreviation of Cryptography Research and Evaluation Committees. It refers to a Japanese project to evaluate and monitor the security of e-Government recommended ciphers, as well as to examine the establishment of evaluation criteria for cryptographic modules.

3. It is suspicious that the P-value derived from the LZ-complexity of a random sequence takes *continuous* uniform distribution.

Doganaksoy and Göloglu [5] proposed a randomness test based on the theoretical distribution of the LZ-complexity. Their test is described as follows.

The Modified LZ Test

S1 Let s be a given binary sequence of length N.

S2 Divide s into $N' = \lfloor \frac{N}{M} \rfloor$ non-overlapping blocks of length M. Hereafter, we assume that $M = 1024$.

S3 Let $\pi_i,\ 1 \le i \le N'$, be the LZ-complexity of the i-th block.

S4 Count $r_1, \ldots, r_5$ given as follows.

$$\begin{aligned} r_1 &= |\{i : \pi_i \le 174,\ 1 \le i \le N'\}|, \\ r_2 &= |\{i : \pi_i = 175,\ 1 \le i \le N'\}|, \\ r_3 &= |\{i : \pi_i = 176,\ 1 \le i \le N'\}|, \\ r_4 &= |\{i : \pi_i = 177,\ 1 \le i \le N'\}|, \\ r_5 &= |\{i : \pi_i \ge 178,\ 1 \le i \le N'\}|. \end{aligned}$$

S5 Compute the test statistic: $\chi^2 = \sum_{i=1}^{5} \frac{(r_i - N'q_i)^2}{N'q_i}$, where $q_i = 0.05262, q_2 = 0.19987, q_3 = 0.39720, q_4 = 0.29107, q_5 = 0.05924$.
Make sure that $N' \min\{q_i : 1 \le i \le 5\} \ge 5$.

S6 Compute P-value= igamc $\left(\frac{4}{2}, \frac{\chi^2}{2}\right)$.

The theoretical distribution for the number of sequences with a given LZ-complexity is calculated for length M using two recurrence equations. But M is restricted to be relatively small (e.g., $M \le 1024$) because the time complexity to

compute the theoretical distribution is very high, i.e., $\mathrm{O}\left(M^5/\log M\right)$. By contrast, the NIST LZ test uses the LZ-complexity for a whole sequence of length $N = 10^6$. This restriction is a serious defect because the complexity of a long sequence is not evaluated directly. Hence, this test cannot become a replacement of the NIST LZ test.

The NIST test suite is superior to the statistical test packages of [33,37] because only the NIST test suite includes randomness tests based on complexity measures. So, the three problems of the LZ test listed before should be solved. But, it is impossible to solve the third problem because the empirical distribution of P-values for random sequences of length 10^6 is strictly discrete even if the number of samples is very large. So, the third problem is the main defect of the LZ test. In order to overcome the main defect of the LZ test, we construct a new randomness test based on the T-complexity instead of the LZ-complexity as described in the next section [19, 22].

6.2 Randomness Test Based on the T-complexity

When the forward T-decomposition algorithm for generalized T-codes (Algorithm-C) was used, the empirical mean and standard deviation of the T-complexity for 4800 Marsaglia's random numbers [37] of length 10^6 were 38718.6 and 58.6585, respectively. These values are almost the same as the case of the forward T-decomposition algorithm for simple T-codes (Algorithm-B). This result is due to the fact that in the case of random numbers, the same long sequence seldom occurs sequentially even if it occurs several times. If pseudo-random numbers have a defect such that some long subsequences tend to occur sequentially, the defect can also be detected by Algorithm-B. Hence, for the purpose of randomness testing, we use Algorithm-B rather than Algorithm-C.

We first compare distributions of the LZ-complexity and the T-complexity. We

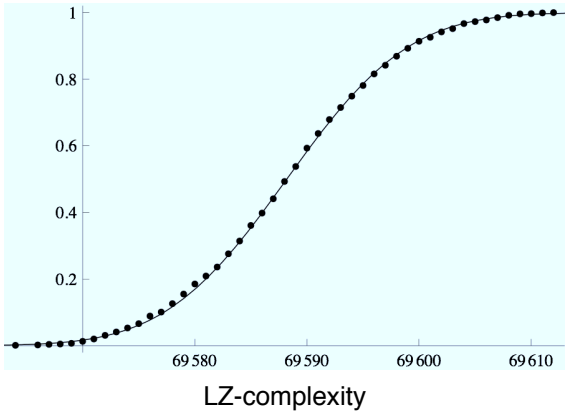


Figure 6.1. Empirical distribution of the LZ-complexity of sequences of length 10^6 (dots) and the normal distribution $\mathrm{N}(69588.2, 8.55788^2)$ (solid line).

sampled 10^3 sequences of length 10^6 from Marsaglia's random numbers, which are claimed to be virtually unassailable sources of random bits. Figure 6.1 shows the empirical distribution of the LZ-complexity and the normal distribution in the form of cumulative distribution. It was assumed in the NIST LZ test that the former distribution can be approximated by the latter distribution. But we note that the empirical distribution of the LZ-complexity is strictly discrete although these two distributions are close to each other. This means that the distribution of P-values also becomes discrete. By contrast, as shown in Fig. 6.2, the empirical distribution of the T-complexity can be approximated well by the normal distribution. When the length of a sequence is increased to 10^8, the empirical distribution of the LZ-complexity can be treated as a continuous distribution, which can be approximated well by the normal distribution. But the computation of the LZ-complexity becomes very time and memory consuming. Moreover, it is uncommon to evaluate random sequences of length $\geq 10^8$ with the NIST test suite because a single sequence has to be evaluated by all randomness tests included in the NIST test suite, some of which are time-consuming.

Next we investigate the T-complexity profile using a moving average model $U(i) = e(i) - \psi e(i-1)$, where $e(i)$ is a random variable following the standard

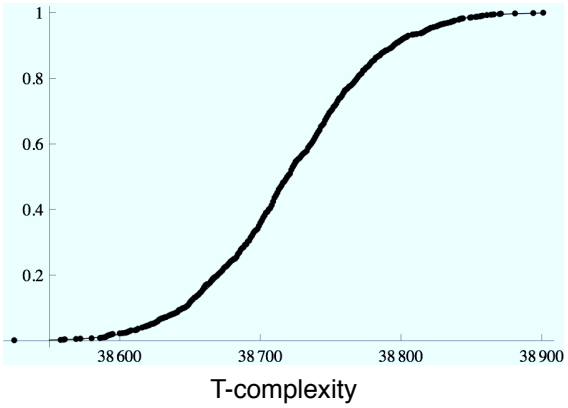


Figure 6.2. Empirical distribution of the T-complexity of sequences of length 10^6 (dots) and the normal distribution $\mathrm{N}(38720.6, 58.2937^2)$ (solid line).

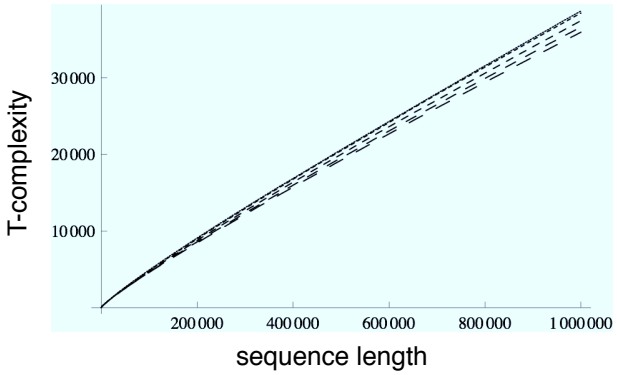


Figure 6.3. T-complexity profiles for $\psi = 0.0, 0.2, 0.4, 0.6, 0.8$. Wider broken lines correspond to larger ψ.

normal distribution $\mathrm{N}(0, 1)$ and ψ is a constant parameter. We used MT to generate $e(i)$. $U(i)$ outputs zero or one depending on its sign. The dependence between adjacent bits becomes stronger as ψ becomes larger. Figure 6.3 shows T-complexity profiles for sequences of length 10^6 and Fig. 6.4 presents a magnified detail of Fig. 6.3. Five values of ψ were considered. We note from Figs. 6.3 and 6.4 that T-complexity profiles can distinguish the level of dependence.

Let T be a random variable defined as the T-complexity of a sequence. If H_0 is true (the sequence is random), $Z = \frac{T-\mu}{\sigma}$ approximately follows $\mathrm{N}(0, 1)$, where $\mu = 38720.6, \sigma = 58.2937$. The values of μ and σ were obtained experimentally from 4800 Marsaglia's random sequences of length 10^6. If $Z \sim \mathrm{N}(0, 1)$, P-value =

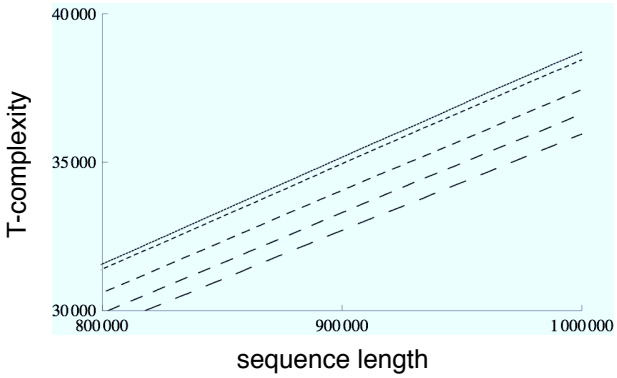


Figure 6.4. Magnified detail of Fig. 6.3.

erf $\left(|Z|/\sqrt{2}\right)$ follows the uniform distribution U$(0,1)$. Thus, the distribution of P-values derived from T-complexities satisfies the assumption of the NIST test suite. We sampled 10^3 sequences of length 10^6 generated by the DES in the output feedback mode [10], which is considered a reliable random number generator. The Kolmogorov-Smirnov test [33] was applied to the empirical distribution of P-values and U$(0,1)$. The test result was that $K_n^+ = 1.21525$ and $K_n^- = 0.139923$. Since the critical point of the significance level $\alpha = 0.05$ is 1.2188, the KS test concluded that the P-value follows U$(0,1)$.

Next, we consider a pass ratio P defined by

$$P = \frac{\#\{\text{P-value} \ : \ (\text{P-value}) \geq \alpha\}}{m},$$

where $\#\{A\}$ stands for the number of occurrences of event A, α is a given significance level, m is a given number of trials, and the P-value is calculated from the T-complexity t. The pass ratio P is normalized by

$$\eta = \frac{P - \hat{p}}{\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}},$$

where $\hat{p} = 1 - \alpha$. When $X_i, \ 1 \leq i \leq m$, are Bernoulli random variables taking 1 with probability $1-\alpha$ and 0 with probability α, $B = \sum_{i=1}^{m} X_i$ follows the binomial

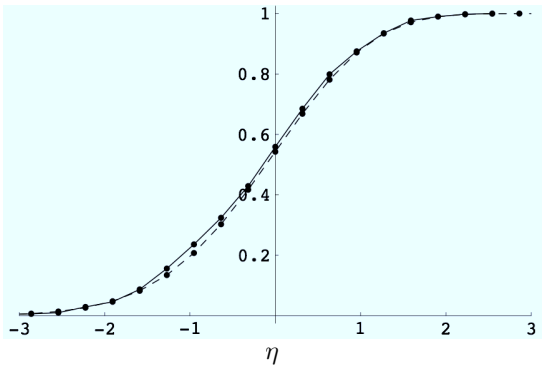


Figure 6.5. Empirical distribution of η (solid line) and the theoretical distribution (broken line).

Table 6.3. Probabilities of η.

Range of η	Expected	Observed
$(-\infty, -2)$	0.0264	0.0290
$(-2, -1)$	0.108	0.127
$(-1, 0)$	0.283	0.273
0	0.126	0.130
$(0, 1)$	0.328	0.316
$(1, 2)$	0.119	0.115
$(2, \infty)$	0.0101	0.0100
Sum	1	1

distribution with parameters m and α. Under the null hypothesis, η is expected to follow the distribution of $\left(\frac{B}{m} - \hat{p}\right) / \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$. In Fig. 6.5, this theoretical distribution is displayed together with an empirical distribution of η based on $m = 10^3$ samples of P in the case where $\alpha = 0.01$ and sequences of length 10^6 are generated by the DES with the output feedback mode. The chi-square goodness-of-fit test was applied to these two distributions using data shown in Table 6.3. The calculated χ^2-statistic is 4.64. Since the upper 5% critical point of the χ^2 distribution with 6-degree of freedom is 12.592, the chi-square goodness-of-fit test concluded that the empirical distribution of η follows the theoretical distribution. Therefore, a randomness test based on the T-complexity may be expected to perform well.

From the above results, the procedure of a new randomness test based on the

T-complexity, called the T-complexity test hereafter, can be constructed as follows.

T-complexity Test

S1 Set α and m to a given significance level and a given trial number, respectively (e.g. $\alpha = 0.01$ and $m = 10^3$).

S2 Let s and t be a given binary sequence of length N and the T-complexity of s, respectively.

S3 Compute P-value$= \mathrm{erf}\left(\frac{|z|}{\sqrt{2}}\right)$, where $z = \frac{t-\mu}{\sigma}$.

S4 If the number of trials is less than m, go back to S2.

S5 Compute $P = \dfrac{\#\{\text{P-value} : (\text{P-value}) \geq \alpha\}}{m}$.

S6 Compute $\eta = \frac{P-\hat{p}}{\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}}$, where $\hat{p} = 1 - \alpha$.

S7 Test the null hypothesis $H_0 : \ \eta \sim \mathrm{N}(0, 1)$.

S8 If H_0 is rejected, conclude that the given sequences are non-random.

Note that the decision rule of the NIST test suite can be used instead of steps S5–S8 after collecting m P-values.

6.3 Experiments

Sequences of good random numbers, e.g. Marsaglia's random numbers and pseudo-random numbers generated by the DES in the output feedback mode, can pass the T-complexity test. But, it can detect some sequences of undesirable pseudo-random numbers that cannot be detected well by the NIST test suite as shown in the following examples.

Example 1

First, we considered pseudo-random numbers obtained by a multiplicative congruential generator (MCG):

$$\begin{cases} X_{n+1} &= 65539X_n \mod 2^{31} \\ X_0 &= 1 \end{cases} \tag{6.2}$$

It is well-known that these pseudo-random numbers are undesirable because three adjacent numbers have a three-dimensional lattice structure as shown in Fig. 6.6. We generated 10^3 sequences of length 10^6 by a sequential concatenation of $\left\lfloor \frac{X_i}{2^{23}} \right\rfloor$ (eight bits). In the case of the Marsaglia's random numbers, the mean and standard deviation of the LZ-complexity were $\mu_{LZ} = 69588.2$ and $\sigma_{LZ} = 8.55788$, respectively, and the mean and standard deviation of the T-complexity were $\mu_T = 38720.6$ and $\sigma_T = 58.2937$, respectively. On the other hand, in the case of the MCG sequences, the mean of the LZ-complexity was $69584.2 \approx \mu_{LZ} - 0.46\sigma_{LZ}$, and the mean of the T-complexity was $37768.4 \approx \mu_T - 16.3\sigma_T$. Thus, the T-complexity can detect the non-randomness of the MCG sequences more easily than the LZ-complexity. The 10^3 MCG sequences were evaluated using both the NIST test suite and the T-complexity test. The default parameters were used in the NIST test suite. We used the NIST test suite after correcting it on the basis of [2, 15–18, 20]. The pass ratio of the T-complexity test was $P = 0$ ($\eta = -314.6$). On the other hand, the reject ratios for the NIST test suite were very low as shown in Table 6.4. Hence, the T-complexity test is considerably superior to the NIST test suite in terms of the rejection of the undesirable MCG sequences. Additionally, we also tried the modified LZ test proposed in [5], with α set to 0.01. But, the reject ratio turned out to be 0.006.

Example 2

Next, we considered a non-random sequence $Y = Y_0, Y_1, Y_2, \cdots$ such that the size of each Y_i is one byte (eight bits). For $i \geq 0$, Y_{3i} and Y_{3i+1} are generated by MT, but Y_{3i+2} is the lower eight bits of $Y_{3i} + Y_{3i+1}$. We generated 10^3 sequences of length 10^6. The mean of the LZ-complexity was $69586.5 \approx \mu_{LZ} - 0.20\sigma_{LZ}$, and the mean of the T-complexity was $38284.7 \approx \mu_T - 7.48\sigma_T$. Thus, the T-complexity can detect the non-randomness of the sequences Y more easily than the LZ-complexity. The 10^3 sequences Y were evaluated using the NIST test suite, the T-complexity test, and the modified LZ test. The pass ratio of the T-complexity test was $P = 0$ ($\eta = -314.6$). On the other hand, the reject ratios for the NIST test suite were very low as shown in Table 6.5, and the reject ratio of the modified LZ test was 0.007. Hence, the T-complexity test is also considerably superior to both the NIST test suite and the modified LZ test in terms of the rejection of the non-random sequences Y.

It is worth noting that if (Y_{3i}, Y_{3i+1}) is perfectly random, each of (Y_{3i-1}, Y_{3i}) and (Y_{3i+1}, Y_{3i+2}) is also perfectly random, but $(Y_{3i}, Y_{3i+1}, Y_{3i+2})$ is not random. We can easily construct many kinds of non-random numbers with such characteristics, but the NIST test suite is weak in the detection of such non-random numbers. Therefore, our proposed randomness test is suitable as a supplement to the NIST test suite in order to make up for its weakness.

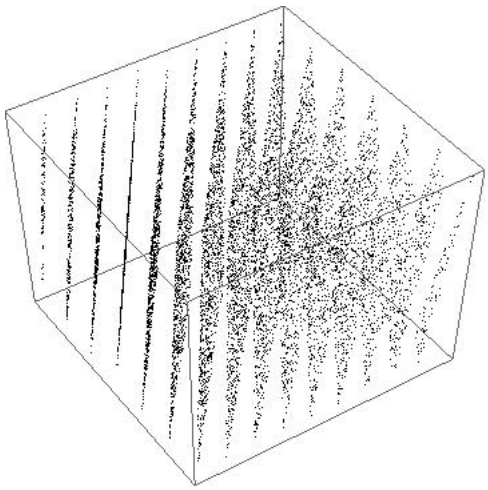


Figure 6.6. View of 10^4 triples generated from the MCG given by Eq. (6.2).

Table 6.4. Reject Ratios of the NIST test suite for MCG sequences when $\alpha = 0.01$.

Test Name	Reject Ratio
Monobit	0.010
Block Frequency	0.009
Cusum	0.011
Runs	0.007
Long Runs of Ones	0.014
Rank	0.009
Spectral DFT	0.000
Aperiodic Templates	0.021
Periodic Templates	0.010
Universal Statistical	0.015
Approximate Entropy	0.010
Random Excursions	0.021
Random Excursions Variant	0.013
Serial	0.037
Linear Complexity	0.009

Table 6.5. Reject Ratios of the NIST test suite for sequences Y when $\alpha = 0.01$.

Test Name	Reject Ratio
Monobit	0.012
Block Frequency	0.008
Cusum	0.014
Runs	0.012
Long Runs of Ones	0.023
Rank	0.004
Spectral DFT	0.000
Aperiodic Templates	0.019
Periodic Templates	0.011
Universal Statistical	0.010
Approximate Entropy	0.015
Random Excursions	0.015
Random Excursions Variant	0.015
Serial	0.021
Linear Complexity	0.010

6.4 Conclusions

In Section 6.2, we first showed that the empirical distribution of the T-complexity for random sequences of length 10^6, the most common length used, can be approximated well by the normal distribution, while the distribution of the LZ-complexity for random sequences of length 10^6 is strictly discrete. So, we found that the main defect of the NIST LZ test can be solved by use of the T-complexity instead of the LZ-complexity in randomness testing. Next, the experiment using a moving average model confirmed that the magnitude of the T-complexity of a given sequence s indicates the degree of randomness, and T-complexity profiles can distinguish the level of dependence between adjacent bits. Then, we checked that the P-value of the T-complexity of a random sequence distributes uniformly in the range of 0 to 1 via applying the Kolmogorov-Smirnov test on 10^3 sampled P-values. Additionally, we checked that, for random sequences, the empirical distribution of the normalized pass ratio η follows the theoretical distribution. On the basis of the above results, we derived the T-complexity test.

In Section 6.3, the power of the T-complexity test was evaluated. The T-complexity test detected the undesirable pseudo-random numbers generated by the MCG and the non-random byte sequences $Y = Y_0, Y_1, Y_2, \cdots$, where Y_{3i} and Y_{3i+1} are random, but Y_{3i+2} is given by $Y_{3i} + Y_{3i+1} \bmod 2^8$, more easily than the NIST LZ test. Moreover, it outperformed not only all other randomness tests included in the NIST test suite but also the modified LZ test [5] in terms of the detection of the non-random sequences.

Since the output form of the T-complexity test is the same as that of the NIST test suite, it can easily be used in combination with the NIST test suite as a supplement to the NIST test suite.

Chapter 7

Conclusions

In Chapter 2, we devised the forward T-decomposition algorithms, Algorithm-B for simple T-codes and Algorithm-C for generalized T-codes, in order to parse a given sequence sequentially. Both algorithms are efficient owing to the use of a trie structure. For Algorithm-B, the experiment showed that its computation time for a random sequence of length N is about $\mathrm{O}(N^{1.2})$. The forward T-decomposition enables the use of the T-complexity in on-line applications.

In Chapter 3, we derived the expressions of the T-complexity profile and the LZ-complexity profile in a unified way using the proposed differential equation technique. The obtained formulas regarding the T-complexity profile and the LZ-complexity profile agree well with the ones in previous studies. We also clarified that the T-entropy H_T corresponds to $kh(p)$, where k is a constant and $h(p)$ is the binary entropy function, in our derivation. The crucial point of our derivation is that it can explain the reason why the maximum T-complexity profile is necessarily expressed using the logarithmic integral function.

In Chapter 4, the properties of the maximum T-complexity sequences were investigated and compared with those of the maximum LZ-complexity sequences using the NIST test suite, the discrete Fourier transform (DFT), and the autocorrelation function. The analysis showed that the maximum T-complexity sequences

are less random than the maximum LZ-complexity sequences. The NIST DFT test and the NIST Universal test clearly detected the non-randomness in the maximum T-complexity sequences. However, the maximum T-complexity sequences on average show larger LZ-complexities than random sequences. The cause of the non-randomness in the maximum T-complexity sequences was quantitatively and qualitatively explained.

In Chapter 5, we proposed a new data compression scheme based on T-codes using a dictionary method such that all phrases added to a dictionary have a recursive structure similar to T-codes. The proposed scheme can overcome the defects of the Müller-Schimpfky scheme. We considered three dictionary updating rules, Methods A, B, and C. Regardless of the methods used, our scheme succeeded in compressing the Calgary Corpus more efficiently than the Müller-Schimpfky scheme and the UNIX *compress*. However, the proposed scheme turned out to be inferior to LZMW, a variant of LZ78, the UNIX *gzip*, and the UNIX *bzip2*. The experiment showed that the number of parsed subsequences of our scheme, $\#\xi$, has a linear relation with the T-complexity. Our scheme can be implemented as a single-pass scheme, and when it is used with Method C, it is proved to be universal for stationary ergodic sources under the assumptions that the number of ξ's and the dictionary size are proportional to the T-complexity.

In Chapter 6, we proposed the T-complexity test. We checked that the distribution of P-values and the normalized pass ratio η follow the respective theoretical distribution in the case of random sequences in order to show the validity of the T-complexity test. The proposed test outperformed the NIST LZ test, the modified LZ test proposed in [5], and all other randomness tests included in the NIST test suite in terms of the detection of the undesirable pseudo-random numbers generated by the MCG and the non-random byte sequences $Y = Y_0, Y_1, Y_2, \cdots$, where Y_{3i} and Y_{3i+1} are random, but Y_{3i+2} is given by $Y_{3i} + Y_{3i+1} \bmod 2^8$. Since the output form of the T-complexity test is the same as that of the NIST test suite,

it can easily be used in combination with the NIST test suite as a supplement to the NIST test suite.

We remark that since both the standard T-decomposition (Algorithm-A) and the forward T-decomposition (Algorithm-B) are based on the same recursive structure of T-codes, the standard T-decomposition can also be used for randomness testing in the same way as the forward T-decomposition. But, Algorithm-B is faster than Algorithm-A as shown in Table 2.1 and can process a given sequence on-line. Furthermore, the forward T-decomposition has a better correspondence to the LZ78 incremental parsing than the standard T-decomposition because, on the basis of the forward T-decomposition, we can derive the expressions of the T-complexity profile and the LZ-complexity profile in a unified way and can design the sequential data compression scheme based on T-codes.

In conclusion, the T-complexity has superior properties to the well-known LZ-complexity, and hence it may be used in numerous applications as a replacement of the LZ-complexity in the future.

Future aims are listed as follows.

- To improve the implementation of the forward T-decomposition algorithm so that its computation time is reduced to about $\mathrm{O}(N)$, where N is the length of a random sequence.

- To provide the mathematical proof that the asymptotic distribution of the T-complexity of a random sequence is the normal distribution.

- To popularize the T-complexity test in the field of cryptography.

Bibliography

[1] M. Burrows and D. J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm," *SRC Research Report 124*, Digital Systems Research Center, May 1994.

[2] J. S. Coron, "On the Security of Random Sources," *Proceedings PKC'99*, LNCS 1560, Springer-Verlag, pp. 29–42, 1999.

[3] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, 2006.

[4] P. Diaconis, S. Holmes, and R. Montgomery, "Dynamical Bias in the Coin Toss," *SIAM Review*, vol. 49, no. 2, pp. 211–235, Apr. 2007.

[5] Ali Doganaksoy and Faruk Gölogluu, "On Lempel-Ziv Complexity of Sequences," *Proceedings SETA2006*, LNCS 4086, Springer-Verlag, pp. 180–189, 2006.

[6] W. Ebeling, R. Steuer, and M. R. Titchener, "Partition-Based Entropies of Deterministic and Stochastic Maps," *Stochastics and Dynamics*, vol. 1, no. 1, Mar. 2001.

[7] R. Eimann, U. Speidel, N. Brownlee, and J. Yang, "Network Event Detection with T-Entropy," *CDMTCS Research Reports*, CDMTCS-266, May 2005.

[8] The eSTREAM Project, http://www.ecrypt.eu.org/stream.

[9] FIPS Pub 46, "Data Encryption Standard (DES)," 1977.

[10] FIPS Pub 81, "DES Modes of Operation," Dec. 1980.

[11] FIPS Pub 180-1, "Secure Hash Standard," Apr. 1995.

[12] T. A. Gulliver, I. Makwakwa, and U. Speidel, "On the Generation of Aperiodic and Periodic Necklaces via T-augmentation," *Fundamenta Informaticae*, vol. 83 (1–2), pp. 91–107, Aug. 2008.

[13] U. Günther, "Data Compression and Serial Communication with Generalized T-codes," *Journal of Universal Computer Science*, vol. 2, no. 11, pp. 769–795, 1996.

[14] U. Günther, *Robust Source Coding with Generalized T-codes*, PhD thesis, The University of Auckland, 1998.

[15] K. Hamano, F. Satoh, and M. Ishikawa, "Randomness test using Discrete Fourier Transform," *Technical Report* 6841, Technical Research and Development Institute, Japan Defense Agency, Sept. 2003. (in Japanese)

[16] K. Hamano, "The Distribution of the Spectrum for the Discrete Fourier Transform Test Included in SP800-22," *IEICE Trans. Fundamentals*, vol. E88-A, no. 1, pp. 67–73, Jan. 2005.

[17] K. Hamano, "Correction of "Test for the Longest Run of Ones in a Block" Included in NIST Randomness Test Suite," *IEICE Technical Report*, ISEC2007-3, pp. 17–21, May 2007. (in Japanese)

[18] K. Hamano and T. Kaneko, "Correction of Overlapping Template Matching Test Included in NIST Randomness Test Suite," *IEICE Trans. Fundamentals*, vol. E90-A, no. 9, pp. 1788–1792, Sept. 2007.

[19] K. Hamano and H. Yamamoto, "Construction of Randomness Testing Based on T-Codes" *IEICE Technical Report*, IT2007-55, pp. 43–50, Feb. 2008. (in Japanese)

[20] K. Hamano and H. Yamamoto, "A New Randomness Test Based on All the Autocorrelation Values," *IEICE Technical Report*, ISEC2008-4, pp. 23–30, May 2008. (in Japanese)

[21] K. Hamano and H. Yamamoto, "On Properties of Sequences with Maximum T-complexity," *Proceedings SITA2008*, pp. 1011–1016, Oct. 2008. (in Japanese)

[22] K. Hamano and H. Yamamoto, "A Randomness Test Based on T-codes," *Proceedings ISITA2008*, pp. 1095–1100, Dec. 2008.

[23] K. Hamano, F. Sato, and H. Yamamoto, "A New Randomness Test Based on Linear Complexity Profile," *IEICE Trans. Fundamentals*, vol. E92-A, no. 1, pp. 166–172, Jan. 2009.

[24] K. Hamano and H. Yamamoto, "A Differential Equation Method to Derive the Formulas of the T-complexity and the LZ-complexity," *Proceedings IEEE ISIT2009*, pp. 625–629, June 2009.

[25] K. Hamano and H. Yamamoto, "A New Data Compression Algorithm Based on a Dictionary Method Using Recursive Construction of T-codes," *Technical Report of IEICE*, IT2009-20, pp. 85–90, July 2009. (in Japanese)

[26] C. J. A. Jansen, "Investigations on Nonlinear Stream Cipher Systems: Construction and Evaluation Methods," Ph.D. Thesis, Technical University of Delft, 1989.

[27] M. Kaneda, H. Okutomi, and K. Nakamura, "A Study on Maurer's "Universal Statistical" Test Included in NIST Randomness Test Suite," *Proceedings SCIS2007*, 3E1-3, Jan. 2007. (in Japanese)

[28] M. Kaneda, H. Okutomi, and K. Nakamura, "A Study on Discrete Fourier Transform Test Included in NIST Randomness Test Suite," *IEICE Technical Report*, ISEC2006-124, pp. 53–58, Mar. 2007. (in Japanese)

[29] T. Kaneko, "Investigation Report on Test Methods of Pseudo random Number Generator System - Lempel-Ziv Compression Test -," *CRYPTREC Technical Report*, no. 0206. (in Japanese)

[30] T. Kaneko, "Investigation Report on Test Methods of the Pseudo Random Number Generating System," *CRYPTREC Technical Report*, no. 0211. (in Japanese)

[31] J. C. Kieffer and En-Hui Yang, "Grammar-Based Codes: A New Class of Universal Lossless Source Codes," *IEEE Trans. Inf. Theory*, vol. IT-46, no. 3, pp. 737–754, May 2000.

[32] S. Kim, K. Umeno, and A. Hasegawa, "On the NIST statistical test suite for randomness," *IEICE Technical Report*, ISEC2003-87, pp. 21–27, Dec. 2003. (in Japanese)

[33] D. E. Knuth, *The Art of Computer Programming, vol. 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1997.

[34] D. E. Knuth, *The Art of Computer Programming, vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, pp. 492–512, 1997.

[35] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 1, pp. 75–81, Jan. 1976.

[36] M. Li and P. Vitányi, *An introduction to Kolmogorov Complexity and its Applications*, 3rd ed., Springer-Verlag, 2008.

[37] G. Marsaglia, DIEHARD: a battery of tests of randomness, *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*, Florida State University, Florida, 1995.

[38] J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inf. Theory*, vol. IT-15, pp. 122–127, Jan. 1969.

[39] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Trans. on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[40] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC-Press, 1996.

[41] C. Müller and R. Schimpfky, "Data Compression with T-codes," Student Research Project (University of Auckland, Humboldt-University Berlin), 2004.

[42] National Institute of Standards and Technology, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *NIST special publication 800-22*, 2001.

[43] National Institute of Standards and Technology, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *NIST special publication 800-22 Revision 1*, 2008.

[44] J. von Neumann, "Various Techniques Used in Connection with Random Digits," *J. Research National Bureau of Standards, Applied Math. Series*, no. 12, pp. 36–38, 1951.

[45] H. Okutomi, M. Kaneda, K. Yamaguchi, and K. Nakamura, "A Study on the Randomness Evaluation method using NIST randomness test," *Proceedings SCIS2006*, 1E2-3, Jan. 2006. (in Japanese)

[46] H. Okutomi, M. Kaneda, and K. Nakamura, "A Study on the NIST Randomness Test ～Especially Evaluating "The Test for the Longest Run of Ones in a Block"～," *Proceedings SCIS2008*, 4A1-6, Jan. 2008. (in Japanese)

[47] D. Salomon, *Data Compression : The Complete Reference*, 3rd ed. Springer-Verlag, 2004.

[48] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and Oct. 1948.

[49] J. Soto, "Statistical Testing of Random Number Generators," *Proceedings 22nd NISSC*, Oct. 1999.

[50] J. Soto, "Randomness Testing of the AES Candidate Algorithms," *NIST AES report*, available at http://citeseer.ist.psu.edu/250886.html.

[51] P. R. Stubley, "Adaptive data compression using tree codes," Univ. of Waterloo, Dep. of Elec. Eng., PhD thesis, 1992.

[52] Text Compression Corpus, University of Calgary, available at ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus.

[53] M. R. Titchener, "Technical Note: Digital Encoding by Way of New T-codes," *IEE Proceedings Pt.E (Computers and Digital Techniques)*, vol. 131, no. 4, pp. 151–153, July, 1984.

[54] M. R. Titchener, "Generalised T-codes: extended construction algorithm for self-synchronising codes," *IEE Proceedings Communications*, vol. 143, no. 3, pp. 122–128, June 1996.

[55] M. R. Titchener, "A Deterministic Theory of Complexity, Information and Entropy," *Proceedings ITW'98*, p. 80, Feb. 1998.

[56] M. R. Titchener, "A Novel Deterministic Approach to Evaluating the Entropy of Language Texts," *Proceedings ITALLC'98*, June 1998.

[57] M. R. Titchener, "Deterministic Computation of Complexity, Information and Entropy," *Proceedings IEEE ISIT'98*, p. 326, Aug. 1998.

[58] M. R. Titchener, "A measure of information," Data Compression Conference 2000, *Proceedings DCC 2000*, pp. 353–362, 2000.

[59] M. R. Titchener, R. Nicolescu, L. Staiger, A. Gulliver, and U. Speidel, "Deterministic Complexity and Entropy," *Fundamenta Informaticae*, vol. 64, no. 1-4, pp. 443–461, Jan. 2005.

[60] B. P. Tunstall, "Synthesis of Noiseless Compression Codes," Ph.D. Thesis, Georgia Institute of Technology, 1968.

[61] T. A. Welch, "A Technique for High-Performance Data Compression," *IEEE Computer*, vol. 17, no. 6, pp. 8–19, June 1984.

[62] J. Yang and U. Speidel, "A T-decomposition algorithm with $O(n \log n)$ time and space complexity," *Proceedings IEEE ISIT2005*, pp. 23–27, 2005.

[63] J. Yang and U. Speidel, "String parsing-based similarity detection," *Proceedings ITW2005*, pp. 263–267, 2005.

[64] H. Yokoo, "An improved Ziv-Lempel coding scheme for universal source coding," *Trans. IEICE Japan*, vol. J68-A, no. 7, pp. 664–671, July 1985, (English translation: *Electron. and Commun. Jap.*, Part I: COM, vol. 69, no. 12, pp. 12–20, 1986, *Scripta Technica*).

[65] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[66] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 5, pp. 530–536, Sept. 1978.

List of Publications

Journal and International Symposium

1. K. Hamano and T. Kaneko, "Correction of Overlapping Template Matching Test Included in NIST Randomness Test Suite," *IEICE Trans. Fundamentals*, vol. E90-A, no. 9, pp. 1788–1792, Sept. 2007. ([18])
2. K. Hamano, F. Sato, and H. Yamamoto, "A New Randomness Test Based on Linear Complexity Profile," *IEICE Trans. Fundamentals*, vol. E92-A, no. 1, pp. 166–172, Jan. 2009. ([23])
3. K. Hamano and H. Yamamoto, "A Randomness Test Based on T-codes," *Proceedings ISITA2008*, pp. 1095–1100, Dec. 2008. ([22])
4. K. Hamano and H. Yamamoto, "A Differential Equation Method to Derive the Formulas of the T-complexity and the LZ-complexity," *Proceedings IEEE ISIT2009*, pp. 625–629, June 2009. ([24])
5. K. Hamano and H. Yamamoto, "A Randomness Test Based on T-complexity." (submitted to the IEICE Transactions on Fundamentals)
6. K. Hamano and H. Yamamoto, "Data Compression Based on a Dictionary Method Using Recursive Construction of T-codes," *Proceedings DCC2010*, Mar. 2010. (to appear)

Domestic Symposium and Technical Meeting

1. K. Hamano, "Correction of "Test for the Longest Run of Ones in a

Block" Included in NIST Randomness Test Suite," *Technical Report of IEICE*, ISEC2007-3, pp. 17–21, May 2007. (in Japanese, [17])

2. K. Hamano and H. Yamamoto, "Construction of Randomness Testing Based on T-Codes" *Technical Report of IEICE*, IT2007-55, pp. 43–50, Feb. 2008. (in Japanese, [19])

3. K. Hamano and H. Yamamoto, "A New Randomness Test Based on All the Autocorrelation Values," *Technical Report of IEICE*, ISEC2008-4, pp. 23–30, May 2008. (in Japanese, [20])

4. K. Hamano and H. Yamamoto, "On Properties of Sequences with Maximum T-complexity," *Proceedings SITA2008*, pp. 1011–1016, Oct. 2008. (in Japanese, [21])

5. K. Hamano and H. Yamamoto, "A New Data Compression Algorithm Based on a Dictionary Method Using Recursive Construction of T-codes," *Technical Report of IEICE*, IT2009-20, pp. 85–90, July 2009. (in Japanese, [25])