SUMMARY We propose a randomness test based on the T-complexity of a sequence, which can be calculated using a parsing algorithm called T-decomposition. Recently, the Lempel-Ziv (LZ) randomness test based on LZ-complexity using the LZ78 incremental parsing was officially excluded from the NIST test suite in NIST SP 800-22. This is caused from the problem that the distribution of P-values for random sequences of length $10^6$ is strictly discrete for the LZ-complexity. Our proposed test can overcome this problem because T-complexity has almost ideal continuous distribution of P-values for random sequences of length $10^6$. We also devise a new sequential T-decomposition algorithm using forward parsing, while the original T-decomposition is an off-line algorithm using backward parsing. Our proposed test can become a supplement to NIST SP 800-22 because it can detect several undesirable pseudo-random numbers that the NIST test suite almost fails to detect.

key words: T-code, T-complexity, Lempel-Ziv complexity, NIST SP 800-22, NIST statistical test suite, multiplicative congruential generator

1. Introduction

A pseudo-random sequence is often used as a secret key stream or secret information in many cryptographic systems because the generation of a true random sequence requires high cost. But such pseudo-random sequences must be indistinguishable from true random sequences not to provide any clue to attackers. Hence, randomness tests to evaluate the randomness of sequences are very important.

In 2001, the National Institute of Standards and Technology (NIST) of the U.S. government released NIST SP 800-22 [25], which describes the NIST test suite, a widely used battery of statistical tests in the field of cryptography. But some problems have been found in the NIST test suite. It was reported in [6], [7], [18], [21] that the DFT test and the Lempel-Ziv complexity test (LZ test) in the NIST test suite have some problems. Furthermore, it was found in [20] that the recommended input size of the approximate entropy test should also be modified. 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 more accurate value of a parameter for the DFT test was derived in [11]. Okutomi et al. [27] evaluated the randomness of sequences generated by DES and SHA-1 on the basis of the NIST test suite, and showed that the overlapping template matching test and the Mauer's universal statistical test (Universal test) in the NIST test suite did not follow the theoretical binomial distribution if DES or SHA-1 can be assumed to be an ideal pseudorandom number generator. The problem of the overlapping template matching test was caused from inaccurate probability estimation for templates in the NIST test suite [9]. The accurate probabilities given in [9] are described in NIST SP 800-22 Revision 1 [26] although they have not yet been implemented in the NIST test suite (the Statistical Test Suite Version 2.0b). Moreover, the revised Universal test based on the model proposed by Coron [1] resolved the problems of the original Universal test [17]. It is also reported in [8], [28] that the probabilities used in the longest-run-of-ones test in the NIST test suite need to be corrected. Furthermore, we showed in [11] 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. We also showed in [13] 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 from NIST SP 800-22 [26]. The LZ test is a randomness test based on LZ-complexity, which comes from the LZ78 data compression algorithm [32]. After the LZ test is excluded, NIST SP 800-22 includes no randomness test based on a 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.

According to [19], the main defect of the LZ test is the distribution of P-value derived from the LZ-complexity of a sequence. P-values of random sequences for random sequences of length $10^6$ is strictly discrete even if the number of samples is very
large.

Do˘ganaksoy and Golo˘glu [2] proposed a randomness test based on the theoretical distribution of LZ-complexity such that a sequence of length $N$ is divided into $N = \left\lfloor \frac{N}{2} \right\rfloor$ non-overlapping blocks of length $M$ and the $\chi^2$-statistic is calculated from $N'$ LZ-complexities obtained for every block. The theoretical distribution for the number of sequences with a given LZ-complexity is calculated for length $M$ by two recurrence equations. But the value of $M$ is restricted to be relatively small (e.g., $M \leq 1024$) because the time complexity to compute the theoretical distribution is very high, i.e., $O\left(\frac{N^2}{\log N}\right)$. By contrast, the NIST LZ test uses the LZ-complexity for a whole sequence of length $N = 10^6$. This restriction is a defect because this test does not evaluate the complexity of a long sequence directly and hence this test cannot become a replacement of the NIST LZ test.

In this paper, we rely on T-complexity instead of LZ-complexity. The T-complexity is defined based on T-codes, which are variable-length self-synchronizing codes introduced by Titchener [30]. T-code codewords are constructed by a recursive hierarchical-pattern copying algorithm called T-augmentation. A sequence can be encoded to a longest T-prefix $p_i$ of which is the T-complexity of the sequence $s$. Decomposition parses a sequence into T-prefixes, the number of which is the $T$-complexity of the sequence $s$. In this paper, we propose a randomness test based on T-complexity, which can overcome the main defect of the NIST LZ test.

This paper is organized as follows. After briefly introducing T-codes in Sect. 2, we give a new on-line T-decomposition algorithm using tries\(^\dagger\) to compute the T-complexity of a sequence in Sect. 3. In Sect. 4, we propose a new randomness test based on T-complexity, and we show some examples of undesirable pseudo-random numbers which the proposed test can detect considerably better than the NIST test suite in Sect. 5.

2. T-Codes

We first introduce T-codes [4] and define some notions related to T-codes. Let $A$ be a finite alphabet set. Let $uv$ and $u^*$ represent the concatenation of two strings $u$ and $v$ and the concatenation of $k$ copies of $u$, respectively. A series of T-code sets $S_i$, $i = 1, 2, \ldots$, are constructed using the following recursive formula called T-augmentation.

$$S_i = \bigcup_{j=0}^{k_i} \{p_i^j u \mid u \in S_{i-1}\setminus\{p_i\}\} \cup \{p_i^{k_i+1}\},$$

where $S_0 = A$ and a string $p_i$ is selected from $S_{i-1}$ and $k_i \in \mathbb{N} \equiv \{1, 2, 3, \ldots\}$. String $p_i$ and integer $k_i$ are called a T-prefix and a T-expansion parameter, respectively. $S_i$ is called a T-code set at T-augmentation level $i$. $S_i$ is also represented as $S_i^{(k_1, k_2, \ldots, k_i)}$. A simple T-code set is a T-code set such that T-expansion parameters are restricted to 1, i.e., $k_1 = k_2 = \cdots = 1$. Figure 1 shows code trees corresponding to T-code sets $S_i$ for the case of $S_5 = S_i^{(1, 1, 2, 1, 1)}$ with $A = \{0, 1\}$. Each left and right branches are labeled 0 and 1, respectively. As described in the next section, using a parsing algorithm called T-decomposition [5], we can parse any sequence $s$ into the following form

$$s = p_n^0 p_{n-1}^{k_n} \cdots p_1^{k_1} p_0^0,$$

where $k_n \in A$ 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 $S_n = S_n^{(k_1, k_2, \ldots, k_n)}$. Furthermore, each $p_i$ can be uniquely represented as

$$p_i = p_{i-1}^{k^{(i-1)}} p_{i-2}^{k^{(i-2)}} \cdots p_0^{k^{(0)}} 0,$$

where $k^{(i)} \in A$ and $0 \leq k^{(i)} \leq k_i$ for $j > 0$. The T-complexity of $s$ is defined as

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

when $s$ is given by Eq. (2). It is worth noting that $S_n$ has $2^{t} = \prod_{i=1}^{n} (k_i + 1)$ internal nodes. Hence, the T-complexity coincides with the number of bits necessary to represent all internal nodes in its code tree where $s$ corresponds to one of the longest codewords. In the case of simple T-code sets, the T-complexity $t$ is equal to the number of T-prefixes, namely $n$.

3. T-Decomposition Algorithm

The original T-decomposition algorithm [5], which is described below as Algorithm-A, parses a sequence $s$ backward from the end to the head. So, it is an off-line algorithm since a whole sequence is required before starting the algorithm. Note that each T-prefix $p_i$ obtained by Algorithm-A satisfies Eq. (3).

\(^{\dagger}\)A trie is an ordered-multiway-tree data structure used in computer science [22].
Algorithm-A

A1 Let \( s \) be a given sequence, \( i := 0 \). \( S_0 := \mathcal{A} \). \( s := sa \), where \( a \) is an arbitrary symbol in \( \mathcal{A} \).
A2 Parse \( s \) into codewords in \( S_i \).
A3 If \( s \) becomes a single codeword in \( S_i \), i.e., \( s \in S_i \), then exit.
A4 Let \( p_{i+1} \) be the second last codeword of \( s \).
A5 If the same \( p_{i+1} \) occurs \( \ell \) times adjacent including the second last codeword, let \( k_{i+1} := \ell \).
A6 \( i := i + 1 \). Generate \( S_i \) by Eq. (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 \( S_0 = (\mathcal{A}) \). We obtain \( s = 0.0.1.0.1.0.0.1.0.1.a \), where boundaries are indicated by dots. \( p_1 \) is given as the second last codeword “1” and \( k_1 = 1 \). Then, \( S_1 = \{0, 10, 11\} \) is constructed from \( p_1, k_1 \), and \( S_0 \). For \( i = 1 \), \( s \) is parsed into codewords in \( S_1 \) as \( s = 0.0.10.10.0.0.10.1a \), we obtain \( p_2 = 10, k_2 = 1, S_2 = \{0, 11, 100, 1010, 1011\} \). For \( i = 2 \), \( s \) is parsed into codewords in \( S_2 \) as \( s = 0.0.1010.0.0.101a \), and we obtain \( p_3 = 0 \). Since \( p_3 \) occurs twice adjacent including the second last codeword, we have \( k_3 = 2 \), and

\[ S_3 = \{11, 100, 1010, 1011, 011, 0100, 01010, 01011, 000, 0011, 00100, 001010, 001011\} \]

For \( i = 3 \), \( s \) is parsed into codewords in \( S_3 \) as \( s = 001010.00101a \), and we obtain \( p_4 = 001010 \) and \( k_4 = 1 \).
For \( i = 4 \), \( s \) is parsed into codewords in \( S_4 \) as \( s = 00101000101a \). Finally \( s \) satisfies \( s \in S_4 \), and the algorithm terminates. Then, we have \( s = 00101000101a = p_4 \mid p_3 \mid p_2 \mid p_1 \mid a \), where \( a \) is one of the longest codewords in \( S_4 \) (See the code tree of \( S_4 \) in Fig. 1). The above Algorithm-A is an off-line algorithm. But we propose a new T-decomposition which parses \( s \) sequentially to \( p_1p_2p_3 \cdots \) such that each \( p_i \) satisfies Eq. (3). For simplicity, we consider a simple T-code sets in this section\(^*\). Let \( s = s_1s_2 \cdots s_N \) be a sequence of length \( N \) and let \( s_i = s_{i+1} \cdots s_j \) be a subsequence of \( s \). As an example, let us consider a case that a sequence \( s = s_9^{26} = 00111110011111001111000100 \) is parsed as \( p_1p_2p_3p_4p_5p_6p_7p_8 \), where each \( p_i \) is given as follows.

\[ \begin{align*}
  p_1 &= 0 \\
  p_2 &= p_11 \ (= 01) \\
  p_3 &= 1 \\
  p_4 &= p_11 \ (= 11) \\
  p_5 &= p_5p_10 \ (= 100) \\
  p_6 &= p_6p_11 \ (= 1111) \\
  p_7 &= p_5p_11 \ (= 100011) \\
  p_8 &= p_5p_2p_10 \ (= 1000100) \\
\end{align*} \]

Note that these \( p_i \)'s satisfy Eq. (3) with \( k_j^{(i)} \in \{0, 1\} \), and each \( p_i \) can be obtained by using tries \( t_0, t_1, t_2, \ldots, t_{i-1} \) shown in Fig. 2. 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 don’t have correspondence to any \( p_j \), and the root node has no index.

Assume that \( s_9^{13} = 001111100111110001101 \) is parsed as \( p_1p_2p_3p_4p_5p_6p_7p_8p_9 \) and tries \( t_j, 0 \leq j \leq 6, \) are already constructed from \( \{p_1, p_2, \ldots, p_6\} \) as shown in Fig. 2. Then, T-prefix \( p_7 \) is obtained from the tries as follows. First, we trace \( t_6 \) from the root to a leaf node following the remaining sequence of \( s, s_9^{14} = 100111 \cdots \). Since we reach leaf node 5 in \( t_6 \), we can know 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 satisfy \( j \leq 4 \). Hence, we next use trie \( t_4 \). Again, we trace \( t_4 \) from the root to a leaf node following the remaining sequence of \( s, s_9^{26} = 00111 \cdots \). Since we reach leaf node 2 in \( t_4 \), we can know that the second part of \( p_7 \) is \( p_2, \) i.e., \( p_7 = p_5p_2 \cdots \). Since \( p_2 \) is parsed, we next use trie \( t_1 \). In \( t_1 \), we cannot move from the root when we follow the remaining sequence of \( s, s_9^{19} = 1 \cdots \). In this case, the next symbol “1” becomes the literal symbol of \( p_7, \) and \( p_7 \) is given as \( p_5p_21 \).

Trie \( t_4 \) can be created by adding \( p_7 \) into trie \( t_6 \). Similarly, \( p_9 \) is obtained as follows. Following the remaining sequence of \( s, s_9^{20} = 1000100, \) we trace trie \( t_1 \) 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 \( s_9^{20} = 1000100 \) does not coincide with any \( p_j \) and hence it must be parsed with shorter \( p_j \). So, we move back to the nearest node with a positive index, and we find that \( p_8 = p_5 \cdots \). We next use trie \( t_4 \) since \( p_8 \) is parsed. After similar iterations, \( p_8 \) is given as \( p_5p_2p_10 \).

Sometimes the last T-prefix \( p_8 \) does not end with a literal symbol. For instance, in the case that \( s_9^{19} \) is the same as

\( \footnote{The case of generalized T-code sets is treated in the appendix.} \)
the previous example and $s$ ends by $s_{20}^{22} = 100$, $p_8$ is given as $p_8 = p_3$. In this case, similar to the above case, we move back to the nearest node with a positive index. After similar iterations, $p_3$ can be parsed as $p_8 = p_5 = p_3p_10$.

In the above, 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 parsing algorithm can be described formally as the following Algorithm-B, where $\lambda$ stands for a null string, and “$u := w\lambda$” means that string $u$ is concatenated by string $\lambda$.

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

B1 (Initialization)
- Let $s$ be a given sequence.
  - $i := 1$.
  - Create $t_0$.
B2 $p_i := \lambda$, $v := i - 1$.
B3 Following $s$, trace a path from the root toward a leaf node in trie $t_v$ 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 $v$.
  - $p_j := p_{j-1}$.
  - $v := j - 1$.
  - 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 back from node $v$ toward the root in trie $t_v$. Let $\hat{v}$ be the first node with a positive index that we find in the moving back. Newly let $v$ represent the node $\hat{v}$, and go back to B4.
B6 $p_i := p_{i-1}w$, where $\omega$ is the first symbol of $s$.
  - Output $p_i$.
  - Remove $w$ from the head of $s$.
  - If $s = \lambda$, exit.
  - Update trie $t_{i-1}$ to $t_i$ by adding $p_i$ into $t_{i-1}$.
  - $i := i + 1$.
  - Go back to B2.

Table 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 [31] and its computation time is about $O(N)$ for a random $N$-bit sequence. On the other hand, we note from Table 1 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 $10^8 \approx 2^{20}$ within about 0.07 seconds. This means that Algorithm-B can be practically used in a randomness test that we propose in the next section.

4. Randomness Test Based on T-Complexity

We first compare two distributions of LZ-complexity and T-complexity. We sampled $10^3$ sequences of length $10^6$ from Marsaglia’s random numbers [23], which are claimed to be virtually unassailable sources of random bits. Figure 3 shows the empirical distribution of LZ-complexity and the normal distribution in the form of ogive. 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 LZ-complexity is strictly discrete although these two distributions are close to each other. This means that the distribution of $P$-value also becomes discrete. In contrast, as shown in Fig. 4, the empirical distribution of 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 LZ-complexity can be treated as a continuous distribution which can be approximated well by the normal distribution. But it needs much larger time and many memory resources to compute LZ-complexity. Additionally, it is uncommon to evaluate random sequences of length $10^8$ with the NIST test suite because the same sequences must be evaluated by all randomness tests included in the NIST test suite, some of which are time-consuming.

Next we investigate T-complexity profile by a moving average model $U(i) = \varepsilon(i) - \psi\varepsilon(i-1)$, where $\varepsilon(i)$ is a random variable following the standard normal distribution $N(0, 1)$ and $\psi$ is a constant parameter. We used Mersenne twister (MT) [24] to generate $\varepsilon(i)$. MT is widely regarded as a high quality pseudo-random number generator. $U(i)$ outputs zero

$^a$Trie $t_i$ is simulated by considering only the nodes with index $l$ satisfying $l \leq v$ in trie $t_{i-1}$.
or one depending on its sign. The dependence between adjacent bits becomes stronger as $\psi$ becomes larger. Figure 5 shows T-complexity profiles for sequences of length $10^6$ and Fig. 6 is a magnified view of Fig. 5. Five values of $\psi$ are considered. We note from Figs. 5 and 6 that T-complexity profiles can distinguish the level of dependence.

Let $t$ be the T-complexity of a sequence. Under the null hypothesis that the sequence is random, $Z = \frac{t - \mu}{\sigma}$ approximately follows N(0, 1), where $\mu = 38720.6$ and $\sigma = 58.2937$. The values of $\mu$ and $\sigma$ were obtained experimentally from 4800 Marsaglia’s random sequences of length $10^6$. If $Z \sim$ N(0, 1), P-value = erf($|Z|/\sqrt{2}$) follows the uniform distribution U(0, 1), where erf($x$) = $\frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$. So, the distribution of P-value derived by T-complexity can satisfy the assumption of the NIST test suite. We sampled $10^3$ sequences of length $10^6$ generated by the DES with the output feedback mode [3], which is considered to be a reliable random number generator. The Kolmogorov-Smirnov test was applied to the empirical distribution of P-value and U(0, 1). The test result was that $K^+ = 1.21525$ and $K^- = 0.139923$. Since the critical point of significance level $\alpha = 5\%$ is 1.2188, the KS test concluded that the P-value follows U(0, 1).

Next, we consider pass ratio $r_\alpha$ which is defined by

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

where $\#A$ stands for the number of occurrences of event $A$, $\alpha$ is a given significance level, $I$ is a given number of trials, and the P-value is calculated from T-complexity $t$. The pass ratio $r_\alpha$ can be normalized by $\xi = \frac{r_\alpha - (1-\alpha)}{\sqrt{\frac{1}{2} \cdot \text{var}(r_\alpha)}}$. When $X_i$, $1 \leq i \leq I$, are Bernoulli random variables taking 1 with probability $1 - \alpha$ and 0 with probability $\alpha$, $Y = \sum_{i=1}^I X_i$ follows the binomial distribution with parameters $I$ and $\alpha$. So, under the null hypothesis, $\xi$ is expected to follow the distribution of $\frac{Y - (1-\alpha)}{\sqrt{\frac{1}{2} \cdot \text{var}(Y)}}$. In Fig. 7, this theoretical distribution is described together with an empirical distribution of $\xi$ based on $10^3$ samples of $r_\alpha$ in the case that $I = 10^3$, $\alpha = 0.01$ and sequences of length $10^6$ are generated by the DES with the output feedback mode. The $\chi^2$ test was applied to these two distributions using data shown in Table 2. The calculated $\chi^2$ test statistic is 4.64. Since the upper 5\% critical point of the $\chi^2$ distribution with 6-degree of freedom is 12.592, the $\chi^2$ test concluded that the empirical distribution of $\xi$ follows the theoretical distribution. Therefore, a randomness
test can be well-constructed by using the T-complexity of a sequence.

From the above results, the procedure of a new randomness test based on T-complexity can be constructed as follows.

**Test Procedure**

1. Set \( \alpha \) and \( I \) to a given significance level and a given trial number, respectively, e.g., \( \alpha = 0.01 \) and \( I = 10^3 \).
2. Generate a sequence of length \( N \). Compute the T-complexity \( t \) of the sequence.
3. Compute \( \xi = \frac{r}{\sigma} \).
4. Compute P-value as \( \text{erf} \left( \frac{t}{\sqrt{2}} \right) \).
5. If the number of trials is less than \( I \), go to C2.
6. Compute \( r_a = \frac{\#(\text{P-value} : (\text{P-value}) \geq \alpha)}{I} \).
7. Compute \( \xi = \frac{r_a - (1 - \alpha)}{\sqrt{\frac{r_a(1 - r_a)}{I}}} \).
8. Test the null hypothesis \( H_0 : \xi \sim N(0,1) \).

Instead of steps C6–C9, you may follow the decision rule of NIST SP 800-22 after collecting \( I \) P-values.

**5. Experiments**

Sequences of good random numbers, e.g., Marsaglia’s random numbers and pseudo-random numbers generated by the DES with the output feedback mode, can pass the proposed 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 consider pseudo-random numbers obtained by a multiplicative congruential generator (MCG):

\[
\begin{align*}
X_{n+1} & = 65539X_n \mod 2^{31} \\
X_0 & = 1
\end{align*}
\]  

(4)

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. 8. We generated \( 10^3 \) sequences of length \( 10^6 \) by sequential concatenations of \( \left\lfloor \frac{X}{2^{32}} \right\rfloor \) (eight bits). In the case of the Marsaglia’s random numbers, the mean and standard deviation of LZ-complexity were \( \mu_{LZ} = 69588.2 \) and \( \sigma_{LZ} = 8.55788 \), respectively, and the mean and standard deviation of 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 LZ-complexity was 69584.2 \( \approx \mu_{LZ} - 0.46\sigma_{LZ} \), and the mean of T-complexity was 37768.4 \( \approx \mu_T - 16.3\sigma_T \). Thus, T-complexity can detect the non-randomness of the MCG sequences more easily than LZ-complexity. The \( 10^3 \) MCG sequences were evaluated using both the NIST test suite and the proposed test. The default parameters were used in the NIST test suite. The pass ratio of the proposed test was \( r_a = 0 (\xi = -314.6) \). On the other hand, the reject ratios for the NIST test suite were very low as shown in Table 3. Hence, the proposed test is considerably superior to the NIST test suite to reject undesirable MCG sequences. Additionally, we also tried the modified LZ test proposed in [2], where significance level was set to 0.01. But, the reject ratio was 0.006.

**Example 2**

Next, we considered non-random sequence \( Y = Y_0, Y_1, Y_2, \cdots \) such that the size of each \( Y \) 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 LZ-complexity was 69586.5 \( \approx \mu_{LZ} - 0.20\sigma_{LZ} \), and the mean of T-complexity was 38284.7 \( \approx \mu_T - 7.48\sigma_T \). Thus, T-complexity can detect the
The modified LZ test to reject the non-random sequences is also considerably superior to both the NIST test suite and the modified LZ test was 0.007. Hence, the proposed test were very low as shown in Table 4, and the reject ratio of On the other hand, the reject ratios for the NIST test suite as a supplement to NIST SP 800-22. Since the output form of our proposed test is the same \( Y_0 \), \( Y_1 \), \( Y_2 \), \( Y_3 \), \( Y_{i+1} \), \( Y_{i+2} \), and can design a sequential data compression scheme based on T-codes [15], [16] that can compress the Calgary Corpus [29] more efficiently than the UNIX compress.

### Table 4

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

Finally we remark that since both the original T-decomposition (Algorithm-A) and the forward T-decomposition (Algorithm-B) are based on the same recursive structure of T-codes, the original 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 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 original T-decomposition. 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 using a differential equation technique [14], and can design a sequential data compression scheme based on T-codes [15], [16] that can compress the Calgary Corpus [29] more efficiently than the UNIX compress.

### Acknowledgement

The work of H. Yamamoto is supported in part by JSPS Grand-in-Aid for Scientific Research, No. 22656085.

### References

HANANO and YAMAMOTO: A RANDOMNESS TEST BASED ON T-COMPLEXITY


Appendix: Forward T-Decomposition Algorithm for Generalized T-Codes

In Sect. 3, we proposed Algorithm-B to realize forward T-decomposition for simple T-codes. In this appendix, we give a forward T-decomposition algorithm for generalized T-codes by extending Algorithm-B.
Let $s$ be a given sequence.
\[ i := 1. \]
Create $t_0$.
\[ p_1 := \lambda, \nu := i - 1. \]
Following $s$, trace a path from the root toward a leaf node in trie $t_i$ as far as possible. Let $\nu$ represent the farthest node that we can reach.
\begin{itemize}
  \item If $\nu$ is a node with a positive index, then go to D4.
  \item If $\nu$ is a node with index 0, then go to D5.
  \item If $\nu$ is the root, i.e., we cannot move from the root, then go to D6.
\end{itemize}

Let $j$ be the index of node $\nu$.
\[ j := i - 1. \]
If $p_i$ ends with $p_j$, or $s$ is exhausted, then go to D5.
\[ p_i := p_i p_j. \]
Remove $p_j$ from the head of $s$.
\[ \text{If } k_j = 1, \nu := j - 1. \text{ Otherwise, } \nu := j. \]
Go back to D3.
\[ \text{If there exists no node with a positive index between } \nu \text{ and the root, then go to D6. Otherwise, move back from node } \nu \text{ toward the root in trie } t_i. \]
\[ \text{Let } \hat{\nu} \text{ be the first node with a positive index that we find in the moving back.} \]
\[ \text{Newly let } \nu \text{ represent the node } \hat{\nu}, \text{ and go back to D4.} \]
\[ p_i := p_i \omega, \text{ where } \omega \text{ is the first symbol of } s. \]
Output $p_i$.
\[ \text{If } i > 1, \text{ output } k_{i-1}. \]

$p_1$, i.e., $p_5 = p_3 p_1^2 \cdots$. Since $p_1$ is now included $k_1$ times in $p_5$, $p_1$ cannot be used anymore. Hence we next use 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 $p_5 p_1^1$.

Trie $t_5$ can be created by adding $p_5$ into trie $t_4$ as shown in Fig.A.1. 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 $s$, $s_{23}^{25} = 1111111 \cdots$. Then, since we reach the just created leaf node 5 in $t_5$, we can know that $p_5$ occurs twice successively in $s$.

In this case, we increment $k_5$ by 1 as shown in Fig.A.1.

Similarly, $p_6$ is obtained as follows. Following the remaining sequence of $s$, $s_{23}^{25} = 111$, we trace trie $t_5$ from the root toward a leaf node, and we reach node 3 in $t_5$. When $p_6 = p_3$, $p_6$ does not end with a literal symbol. In this case, we move back to 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 again use trie $t_1$ since $k_1 = 2$. After similar iterations, $p_6$ is given as $p_1^2 1$.

In the same way as Algorithm-B, $t_i$ can simulate any $t_j$ for $0 \leq j < i$, and hence it is sufficient that only the latest trie is memorized.

The above algorithm can be described formally as follows.

**Algorithm-D**

D1 (Initialization)
Let $s$ be a given sequence.
\[ i := 1. \]
Create $t_0$.
D2 $p_i := \lambda, \nu := i - 1. \]
D3 Following $s$, trace a path from the root toward a leaf node in trie $t_i$ as far as possible. Let $\nu$ represent the farthest node that we can reach.
\begin{itemize}
  \item If $\nu$ is a node with a positive index, then go to D4.
  \item If $\nu$ is a node with index 0, then go to D5.
  \item If $\nu$ is the root, i.e., we cannot move from the root, then go to D6.
\end{itemize}

D4 Let $j$ be the index of node $\nu$.
\[ j := i - 1. \]
If $p_i$ ends with $p_j$, or $s$ is exhausted, then go to D5.
\[ p_i := p_i p_j. \]
Remove $p_j$ from the head of $s$.
\[ \text{If } k_j = 1, \nu := j - 1. \text{ Otherwise, } \nu := j. \]
Go back to D3.
D5 If there exists no node with a positive index between $\nu$ and the root, then go to D6. Otherwise, move back from node $\nu$ toward the root in trie $t_i$. Let $\hat{\nu}$ be the first node with a positive index that we find in the moving back. Newly let $\nu$ represent the node $\hat{\nu}$, and go back to D4.
\[ p_i := p_i \omega, \text{ where } \omega \text{ is the first symbol of } s. \]
Output $p_i$.
\[ \text{If } i > 1, \text{ output } k_{i-1}. \]

---

Kenji Hamano received the B.E. and M.E. degrees in Mathematical Engineering and Information Physics from the University of Tokyo in 1999 and 2001, respectively. In 2001, he joined the Technical Research & Development Institute, Ministry of Defense. In 2010, he received the Ph.D. degree from the University of Tokyo. His research interests are randomness, statistics and cryptography.

Hirotsune Yamamoto was born in Waka-yama, Japan, on November 15, 1952. He received the B.E. degree from Shizuoka University, in 1975 and the M.E. and Ph.D. degrees from the University of Tokyo, in 1977 and 1980, respectively, all in electrical engineering. In 1980, he joined Tokushima University. He was an Associate Professor at Tokushima University, the University of Electro-Communications, and the University of Tokyo, from 1983 to 1987, from 1987 to 1993, and from 1993 to 1999, respectively. Since 1999, he has been a Professor at the University of Tokyo. He was with the School of Engineering and the School of Information Science and Technology from 1993 to 1999 and from 1999 to 2004, respectively, and is currently with the School of Frontier Sciences in the University of Tokyo. In 1989 and 1990, he was a Visiting Scholar at the Information Systems Laboratory, Stanford University. His research interests are in Shannon theory, data compression algorithms, and cryptography. Dr. Yamamoto is a member of the IEEE and the STIA (Society of Information Theory and its Applications). He served as the Chair of the IEEE Information Theory Society Japan Chapter in 2002 and 2003, and the TPC (Technical Program Committee) Co-Chair of ISITA2004 (the 2004 International Symposium on Information Theory and its Applications) and the TPC chair of ISITA2008. He was also the president of the STIA in 2008 and 2009, and an Associate Editor for Shannon Theory, IEEE Transactions on Information Theory from 2007 to 2010. He is currently the Editor-in-Chief for the IEICE Transactions on Fundamentals.