IEICE TRANS. FUNDAMENTALS, VOL.E93–A, NO.7 JULY 2010

PAPER

A Randomness Test Based on T-Complexity*

Kenji HAMANO†a), ***Member and*** **Hirosuke YAMAMOTO**†b), ***Fellow***

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 undistinguishable 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 Maurer'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 pseudo-random 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 should take the continuous uniform distribution U(0, 1). But, the empirical distribution of P-values for random sequences of length 10^6 is strictly discrete even if the number of samples is very

Manuscript received September 7, 2009.
Manuscript revised March 5, 2010.

†The authors are with the Department of Complexity Science and Engineering, The University of Tokyo, Kashiwa-shi, 277-8561 Japan.

*The material in this paper was presented in part at the ISITA2008 [12], and all the figures of this paper are reused from [12] under the permission of the IEEE.

a) E-mail: hamano@it.k.u-tokyo.ac.jp
b) E-mail: Hirosuke@ieee.org

DOI: 10.1587/transfun.E93.A.1346

Copyright © 2010 The Institute of Electronics, Information and Communication Engineers

large.

Doğanaksoy and Göloğlu [2] proposed a randomness test based on the theoretical distribution of LZ-complexity such that a sequence of length N is divided into $N' = \left\lfloor \frac{N}{M} \right\rfloor$ non-overlapping blocks of length M and the χ^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., $\mathrm{O}\left(\frac{M^5}{\log M}\right)$. By contrast, the NIST LZ test uses the LZ-complexity for a whole sequence of length $N = 10^6$. This restriction is a 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 codeword in the T-code set by T-decomposition, which corresponds to the inverse operation of T-augmentation. T-decomposition parses a sequence into T-prefixes, the number of which is the T-complexity of the sequence[†]. 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[††] 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 $\mathcal{A}$ be a finite alphabet set. Let uv and u^k represent the concatenation of two strings u and v and the concatenation of k copies of u, respectively. A series of T-code sets $\mathcal{S}_i$, $i = 1, 2, \ldots$, are constructed using the following recursive formula called T-augmentation.

$$\mathcal{S}_i = \bigcup_{j=0}^{k_i} \{p_i^j u \mid u \in \mathcal{S}_{i-1} \backslash \{p_i\}\} \cup \{p_i^{k_i+1}\}, \qquad (1)$$

where $\mathcal{S}_0 = \mathcal{A}$ and a string p_i is selected from $\mathcal{S}_{i-1}$ and $k_i \in \mathbb{N} \equiv \{1, 2, 3, \cdots\}$. String p_i and integer k_i are called a T-prefix and a 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 $\mathcal{S}^{(k_1,k_2,\ldots,k_i)}_{(p_1,p_2,\ldots,p_i)}$. 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

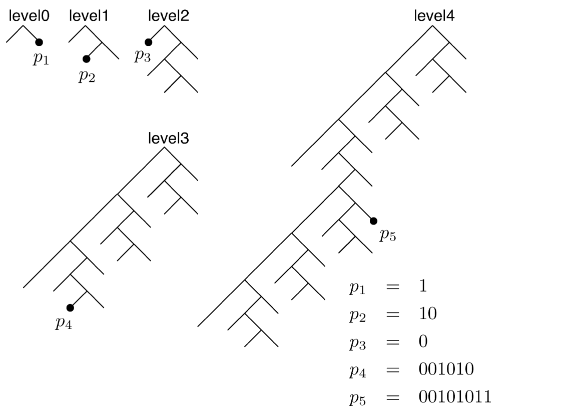

Fig. 1 Intermediate T-code sets $\mathcal{S}_i$, $0 \leq i \leq 4$. (©2008 IEEE [12], Fig. 1)

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 $\mathcal{A} = \{0, 1\}$. 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

$$s = p_n^{k_n} p_{n-1}^{k_{n-1}} \cdots p_1^{k_1} k_0, \qquad (2)$$

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

$$p_i = p_{i-1}^{k_{i-1}^{(i)}} p_{i-2}^{k_{i-2}^{(i)}} \cdots p_1^{k_1^{(i)}} k_0^{(i)}, \qquad (3)$$

where $k_0^{(i)} \in \mathcal{A}$ and $0 \leq k_j^{(i)} \leq k_j$ 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 $\mathcal{S}_n$ has $2^t = \prod_{i=1}^{n}(k_i + 1)$ internal nodes. Hence, the T-complexity coincides with the number of bits 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). Algorithm-A can be implemented with $\mathrm{O}(N \log N)$ time and space complexities when the length of s is N [31].

[†]Strict definition of T-complexity is given in Sect. 2.

[††]A trie is an ordered-multiway-tree data structure used in computer science [22].

Algorithm-A

```
A1 Let $s$ be a given sequence. $i := 0$. $\mathcal{S}_0 := \mathcal{A}$. $s := sa$, 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 last codeword of $s$.
A5 If the same $p_{i+1}$ occurs $\ell$ times adjacently including the
   second last codeword, let $k_{i+1} := \ell$.
A6 $i := i + 1$. Generate $\mathcal{S}_i$ 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 $\mathcal{S}_0$ $(= \mathcal{A})$. 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 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 last codeword, we have $k_3 = 2$, and

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

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.).

The above Algorithm-A is an off-line algorithm. But we propose a new T-decomposition which parses s sequentially to $p_1 p_2 p_3 \cdots$ such that each p_i satisfies Eq. (3). For simplicity, we consider a simple T-code sets in this section[†]. Let $s = s_1 s_2 \cdots s_N$ 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 a case that a sequence $s = s_1^{26} = 00111110011111000111000100$ is 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.

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

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 $p_j, 1 \le j \le i$, corresponds to a path from the root to node j. Nodes with index 0 don't have correspondence to

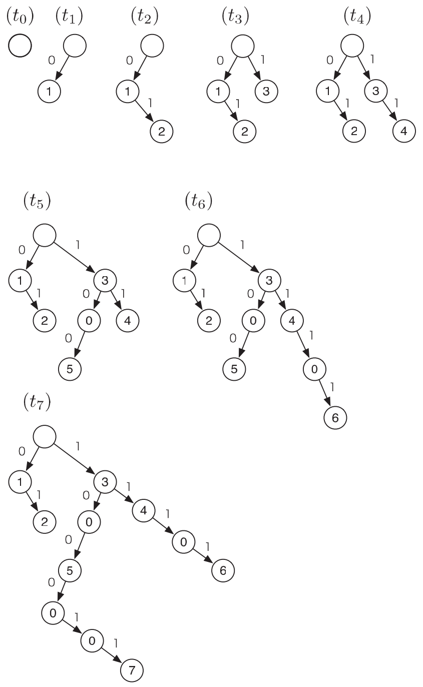

Fig. 2 Trie growth for a sequence $s = 00111110011111000111000100$. (©2008 IEEE [12], Fig. 2)

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 tries t_j, $0 \le j \le 6$, are already constructed from $\{p_1, p_2, \cdots, p_6\}$ as shown in Fig. 2. Then, T-prefix p_7 is obtained from the 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 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 $j \le 4$. 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, $s_{17}^{26} = 011\cdots$. Then, since we reach leaf node 2 in t_4, we can know that the second part of p_7 is p_2, i.e., $p_7 = p_5 p_2 \cdots$. Since p_2 is parsed, we next use 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 "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 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_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, in the case that s_1^{19} is the same as

[†]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 $p_8 = p_5$. In this case, similar to the above case, we move back to the nearest node with a positive index. After similar iterations, p_8 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 $0 \le j < i$ by considering only the nodes with index l satisfying $l \le j$ 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 λ stands for a null string, and "$u := uv$" means that string u is concatenated by string v.

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

```
B1 (Initialization)
   Let $s$ be a given sequence.
   $i := 1$.
   Create $t_0$.
B2 $p_i := \lambda$. $\nu := i - 1$.
B3 Following $s$, trace a path from the root toward a leaf
   node in trie $t_\nu$[†] 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 $v$.
   $p_i := p_i p_j$.
   $\nu := j - 1$.
   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
   back from node $v$ toward the root in trie $t_\nu$. 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 $p_i := p_i\omega$, where $\omega$ is the first symbol of $s$.
   Output $p_i$.
   Remove $\omega$ 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 $2^{15} \le N \le 2^{24}$. 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 the next section.

Table 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

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 ψ 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

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

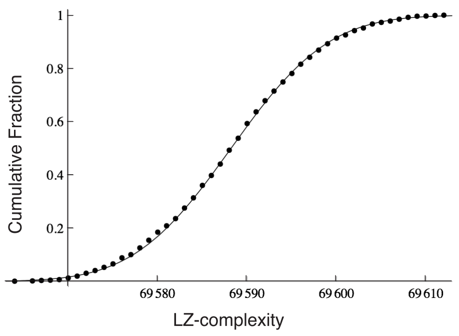

Fig. 3 Empirical distribution of LZ-complexity (dots) and the normal distribution N(69588.2, 8.55788^2) (solid line) for a sequence of length 10^6. (©2008 IEEE [12], Fig. 3)

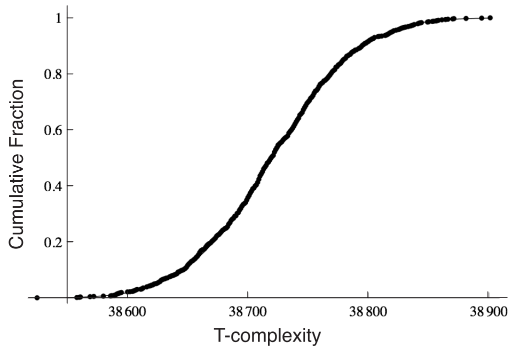

Fig. 4 Empirical distribution of T-complexity (dots) and the normal distribution N(38720.6, 58.2937^2) (solid line) for a sequence of length 10^6. (©2008 IEEE [12], Fig. 4)

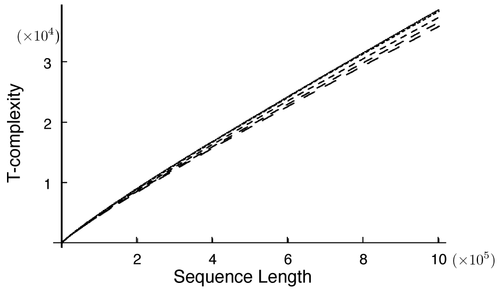

Fig. 5 T-complexity profiles for $\psi = 0.0, 0.2, 0.4, 0.6, 0.8$. Wider broken lines correspond to larger ψ. (©2008 IEEE [12], Fig. 5)

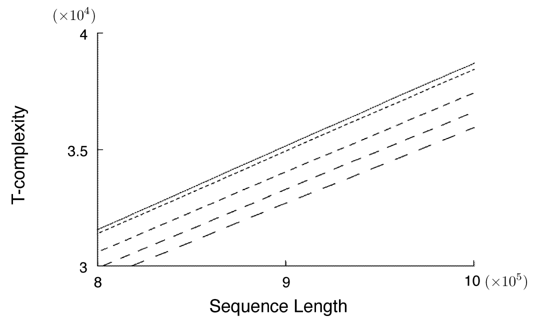

Fig. 6 Magnified view of Fig. 5. (©2008 IEEE [12], Fig. 6)

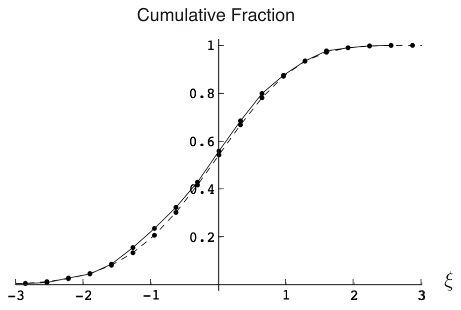

Fig. 7 Empirical distribution of ξ (solid line) and the theoretical distribution (broken line). (©2008 IEEE [12], Fig. 7)

or one depending on its sign. The dependence between adjacent bits becomes stronger as ψ 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 ψ 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 $\mu = 38720.6, \sigma = 58.2937$. The values of μ and σ were obtained experimentally from 4800 Marsaglia's random sequences of length 10^6. If $Z \sim$ N(0, 1), P-value = $\mathrm{erf}\left(|Z|/\sqrt{2}\right)$ follows the uniform distribution U(0, 1), where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-u^2}du$. 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 $K_n^+ = 1.21525$ and $K_n^- = 0.139923$. 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_α 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, α 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_α can be normalized by $\xi = \frac{r_\alpha-(1-\alpha)}{\sqrt{\frac{\alpha(1-\alpha)}{I}}}$. When X_i, $1 \leq i \leq I$, are Bernoulli random variables taking 1 with probability $1-\alpha$ and 0 with probability α, $Y = \sum_{i=1}^I X_i$ follows the binomial distribution with parameters I and α. So, under the null hypothesis, ξ is expected to follow the distribution of $\frac{\frac{Y}{I}-(1-\alpha)}{\sqrt{\frac{\alpha(1-\alpha)}{I}}}$. In Fig. 7, this theoretical distribution is described together with an empirical distribution of ξ based on 10^3 samples of r_α 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 χ^2 test was applied to these two distributions using data shown in Table 2. The calculated χ^2 test statistic is 4.64. Since the upper 5% critical point of the χ^2 distribution with 6-degree of freedom is 12.592, the χ^2 test concluded that the empirical distribution of ξ follows the theoretical distribution. Therefore, a randomness

Table 2 Probabilities of ξ.

Range of ξ	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

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

C1 Set α and I to a given significance level and a given trial number, respectively, e.g. $\alpha = 0.01$ and $I = 10^3$.
C2 Generate a sequence of length N. Compute the T-complexity t of the sequence.
C3 Compute $z = \frac{t-\mu}{\sigma}$.
C4 Compute P-value= $\mathrm{erf}\left(\frac{|z|}{\sqrt{2}}\right)$.
C5 If the number of trials is less than I, go to C2.
C6 Compute $r_\alpha = \dfrac{\#\{\text{P-value} : (\text{P-value}) \geq \alpha\}}{I}$.
C7 Compute $\xi = \frac{r_\alpha - (1-\alpha)}{\sqrt{\frac{\alpha(1-\alpha)}{I}}}$.
C8 Test the null hypothesis $H_0 : \ \xi \sim \mathrm{N}(0, 1)$.
C9 If H_0 is rejected, conclude that sequences are non-random.

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):

$$\begin{cases} X_{n+1} = 65539 X_n \mod 2^{31} \\ X_0 = 1 \end{cases} \tag{4}$$

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 $\left\lfloor \frac{X_i}{2^{23}} \right\rfloor$ (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

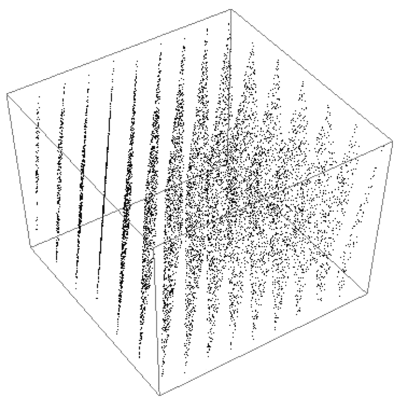

Fig. 8 View of 10^4 triples generated from the MCG given by Eq. (4). (©2008 IEEE [12], Fig. 8)

Table 3 Reject Ratios of the NIST test suite for MCG sequences. Significance level was set to 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

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 $r_\alpha = 0$ ($\xi = -314.6$). 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_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 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

Table 4 Reject Ratios of the NIST test suite for sequences Y. Significance level was set to 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

non-randomness of the sequences Y more easily than LZ-complexity. The 10^3 sequences Y were evaluated using the NIST test suite, the proposed test, and the modified LZ test. The pass ratio of the proposed test was $r_\alpha = 0$ ($\xi = -314.6$). On the other hand, the reject ratios for the NIST test suite were very low as shown in Table 4, and the reject ratio of the modified LZ test was 0.007. Hence, the proposed test is also considerably superior to both the NIST test suite and the modified LZ test to reject the non-random sequences Y.

It is worth noting that if (Y_{3i}, Y_{3i+1}) is perfect random, each of (Y_{3i-1}, Y_{3i}) and (Y_{3i+1}, Y_{3i+2}) is also perfect 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 indispensable as a supplement to the NIST test suite.

6. Conclusions

We proposed a new randomness test based on T-complexity. In order to compute the T-complexity of a sequence, we developed an on-line T-decomposition algorithm, called Algorithm-B in this paper. Algorithm-B realizes forward parsing like the LZ78 incremental parsing, whereas the original T-decomposition is backward parsing. Algorithm-B is efficient owing to the use of a trie structure, and hence it can compute the T-complexity of a random sequence of length 10^6 within approximately 0.1 seconds on a computer with 3 GHz CPU. Since T-complexity has almost ideal continuous distribution of P-values for random sequences, the proposed test can overcome the main defect of the NIST LZ test. We showed that the proposed test can detect undesirable pseudo-random numbers generated by the MCG and the non-random sequences Y more easily than the NIST LZ test, and the proposed test can detect such undesirable pseudo-random numbers considerably better than not only the NIST test suite but also the modified LZ test proposed in [2]. Since the output form of our proposed test is the same as that of the NIST test suite, it can easily be combined with the NIST test suite as a supplement to NIST SP 800-22.

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*, a variant of LZ78.

Acknowledgement

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

References

[1] J.S. Coron, "On the security of random sources," Proc. PKC'99, LNCS 1560, pp.29–42, Springer-Verlag, 1999.

[2] A. Doğanaksoy and F. Göloğlu, "On Lempel-Ziv complexity of sequences," Proc. SETA2006, LNCS 4086, pp.180–189, Springer-Verlag, 2006.

[3] FIPS Pub 81, "DES modes of operation," Dec. 1980.

[4] U. Günther, "Data compression and serial communication with generalized T-codes," Journal of Universal Computer Science, vol.2, no.11, pp.769–795, 1996.

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

[6] 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)

[7] 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.

[8] 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, May 2007. (in Japanese)

[9] 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.

[10] K. Hamano and H. Yamamoto, "Construction of randomness testing based on T-codes," IEICE Technical Report, IT2007-55, Feb. 2008. (in Japanese)

[11] K. Hamano and H. Yamamoto, "A new randomness test based on all the autocorrelation values," IEICE Technical Report, ISEC2008-4, May 2008. (in Japanese)

[12] K. Hamano and H. Yamamoto, "A randomness test based on T-codes," Proc. ISITA2008, pp.1095–1100, Dec. 2008.

[13] 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.

[14] K. Hamano and H. Yamamoto, "A differential equation method to derive the formulas of the T-complexity and the LZ-complexity," Proc. IEEE ISIT2009, pp.625–629, June 2009.

[15] K. Hamano and H. Yamamoto, "A new data compression algorithm based on a dictionary method using recursive construction of T-codes," IEICE Technical Report, IT2009-20, July 2009. (in Japanese)

[16] K. Hamano and H. Yamamoto, "Data compression based on a dictionary method using recursive construction of T-codes," Proc. DCC2010, p.531, March 2010.

[17] M. Kaneda, H. Okutomi, and K. Nakamura, "A study on Maurer's "universal statistical" test included in NIST randomness test suite," Proc. SCIS2007, 3E1-3, Jan. 2007. (in Japanese)

[18] 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, March 2007. (in Japanese)

[19] T. Kaneko, "Investigation report on test methods of pseudo random number generator system — Lempel-Ziv compression test —," CRYPTREC Technical Report, no.0206. (in Japanese)

[20] T. Kaneko, "Investigation report on test methods of the the pseudo random number generating system," CRYPTREC Technical Report, no.0211. (in Japanese)

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

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

[23] 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.

[24] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," ACM Trans. Model. Comput. Simul., vol.8, no.1, pp.3–30, Jan. 1998.

[25] 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.

[26] 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.

[27] H. Okutomi, M. Kaneda, K. Yamaguchi, and K. Nakamura, "A study on the randomness evaluation method using NIST randomness test," Proc. SCIS2006, 1E2-3, Jan. 2006. (in Japanese)

[28] 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,"" Proc. SCIS2008, 4A1-6, Jan. 2008. (in Japanese)

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

[30] M.R. Titchener, "Technical note: Digital encoding by way of new T-codes," IEE Proc. Pt.E (Computers and Digital Techniques), vol.131, no.4, pp.151–153, July 1984.

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

[32] 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.

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.

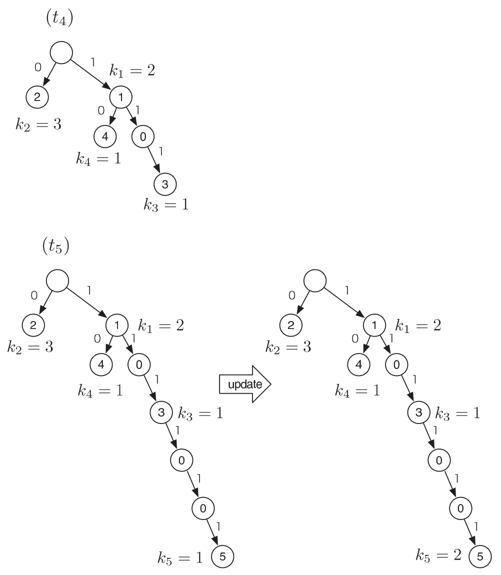

Fig. A· 1 Trie growth for a sequence $s = 11000111101111111111111111$.

We now consider a problem to parse a sequence s sequentially to $p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots$, where each p_i satisfies Eq. (3) and k_i is a positive integer. As an example, let us consider a case that $s = s_1^{25} = 1100011110111111111111111$ is parsed as $p_1^{k_1} p_2^{k_2} p_3^{k_3} p_4^{k_4} p_5^{k_5} p_6^{k_6}$, where

$$k_1 = 2,\ k_2 = 3,\ k_3 = 1,\ k_4 = 1,\ k_5 = 2,\ k_6 = 1,$$

and each p_i is given as follows.

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

Note that each p_i satisfies Eq. (3) and it can be obtained by using the same tries as Algorithm-B. But, in this case, the node with index i must store the value of k_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 tries t_j, $0 \le j \le 4$, are already constructed from $\{p_1, p_2, p_3, p_4\}$ and $\{k_1, k_2, k_3, k_4\}$ as shown in Fig. A· 1. Then p_5 is obtained 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 can know that the first part of p_5 consists of p_3, i.e., $p_5 = p_3 \cdots$. Since p_3 is parsed and $k_3 = 1$, the second p_j must satisfy $j \le 2$. Hence we next use 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 reach leaf node 1 in t_2, we can know that the second part of p_5 is p_1, i.e., $p_5 = p_3 p_1 \cdots$. Since node 1 has $k_1 = 2$, p_1 can become the third p_j. Hence we next use 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, $s_{15}^{25} = 11 \cdots$. Then, since we reach leaf node 1 in t_1, we can know that 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 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 $p_3 p_1^2 1$.

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, $s_{17}^{25} = 111111\cdots$. 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 $0 \le j < i$, 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 $t_\nu$[†] 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 D4.
     • If $v$ is a node with index 0, then go to D5.
     • If $v$ is the root, i.e., we cannot move from the root,
       then go to D6.
D4 Let $j$ be the index of node $v$.
   If $j = i - 1$, then go to D7.
   If $p_i$ ends with $p_j^{k_j}$ or $s$ is exhausted, then go to D5.
   $p_i := p_i p_j$.
   Remove $p_j$ from the head of $s$.
   If $k_j = 1$, $\nu := j - 1$. Otherwise, $\nu := j$.
   Go back to D3.
D5 If there exists no node with a positive index between $v$
   and the root, then go to D6. Otherwise, move back from
   node $v$ toward the root in trie $t_\nu$. 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 D4.
D6 $p_i := p_i \omega$, where $\omega$ is the first symbol of $s$.
   Output $p_i$.
   If $i > 1$, output $k_{i-1}$.
   Remove $\omega$ from the head of $s$.
   $k_i := 1$.
   If $s = \lambda$, output $k_i$, exit.
   Update trie $t_{i-1}$ to $t_i$ by adding $p_i$ into $t_{i-1}$.
   $i := i + 1$.
   Go back to D2.
D7 $k_{i-1} := k_{i-1} + 1$.
   Remove $p_{i-1}$ from the head of $s$.
   If $s = \lambda$, output $k_{i-1}$, exit.
   Go back to D3.
```

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

Using the above Algorithm-D, the mean and standard deviation of T-complexity become 38718.6 and 58.6585, respectively for 4800 Marsaglia's random sequences of length 10^6. These values are almost the same as the case of Algorithm-B. This comes from 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 test, we used Algorithm-B based on T-complexity for simple T-codes rather than Algorithm-D based on T-complexity for generalized T-codes.

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.

Hirosuke Yamamoto was born in Wakayama, 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 cryptology. Dr. Yamamoto is a member of the IEEE and the SITA (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 SITA 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.