Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator

Makoto Matsumoto[1] and Takuji Nishimura[2]
[1]Keio University/Max-Planck-Institut für Mathematik
[2]Keio University

In this paper, a new algorithm named *Mersenne Twister* (MT) for generating uniform pseudorandom numbers is proposed. For a particular choice of parameters, the algorithm provides a super astronomical period of $2^{19937}-1$ and 623-dimensional equidistribution up to 32 bits accuracy, while consuming a working area of only 624 words. This is a new variant of the previously proposed generators TGFSR, modified so as to admit a Mersenne-prime period. The characteristic polynomial has many terms. The distribution up to v bits accuracy for $1 \leq v \leq 32$ is also shown to be good.

Also, an algorithm to check the primitivity of the characteristic polynomial of MT with computational complexity $O(p^2)$ is given, where p is the degree of the polynomial.

We implemented this generator as a portable C-code. It passed several stringent statistical tests, including diehard. The speed is comparable to other modern generators.

These merits are a consequence of the efficient algorithms unique to polynomial calculations over the two-element field.

Categories and Subject Descriptors: F.2.1 [**Theory of Computation**]: Numerical Algorithms and Problems—*Computations in finite fields*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*recurrences and difference equations*; G.3 [**Probability and Statistics**]: random number generation

General Terms: Algorithms, Theory, Experimentation

Additional Key Words and Phrases: finite fields, GFSR, incomplete array, inversive-decimation method, k-distribution, Mersenne Primes, Mersenne Twister, m-sequences, MT19937, multiple-recursive matrix method, primitive polynomials, random number generation, tempering, TGFSR

Dedicated to the Memory of Professor Nobuo YONEDA

Name: Makoto Matsumoto
Address: Gottfried-Claren-Strasse 26 D-53225 BONN, GERMANY. Tel. 49-228-402-0, Fax. 49-228-402-277, email:matsumoto@mpim-bonn.mpg.de
Affiliation: Max-Planck-Institut Für Mathematik
Name:
Address: 3-14-1 Hiyoshi, Yokohama 223 Japan, Tel. +81-45-563-1141, Fax. +81-45-563-5948, email: matumoto@math.keio.ac.jp
Affiliation: Department of Mathematics, Keio University
Name: Takuji Nishimura
Address: 3-14-1 Hiyoshi, Yokohama 223 Japan, Tel. +81-45-563-1141, Fax. +81-45-563-5948, email: nisimura@comb.math.keio.ac.jp
Affiliation: Department of Mathematics, Keio University

1. INTRODUCTION

1.1 A short summary

We propose a new random number generator *Mersenne Twister*. An implemented C-code MT19937 has the period $2^{19937}-1$ and 623-dimensional equidistribution property, which seem to be best among all generators ever implemented. There are two new ideas added to the previous twisted GFSR[?][?] to attain these records. One is an *incomplete array* (see §3.1) to realize a Mersenne-prime period. The other is a fast algorithm to test the primitivity of the characteristic polynomial of a linear recurrence, named *inversive-decimation method* (see §4.3), which works when the degree is a Mersenne exponent. This algorithm does not require even the explicit form of the characteristic polynomial. It needs only (1) the defining recurrence, and (2) some fast algorithm that obtains the present state vector from its 1-bit output stream. The computational complexity of the inversive-decimation method is the order of the algorithm in (2) multiplied by the degree of the characteristic polynomial. To attain higher order equidistribution properties, we used Tezuka's resolution-wise lattice method[?] (see also [?][?]), with Lenstra's algorithm[?][?] for successive minima.

We stress that these algorithms make full use of the polynomial algebra over the two-element field. There are no corresponding efficient algorithms for integers.

1.2 What is high-quality?

There are many pseudorandom number generators implemented in computers, but only a few generators can be used in a serious simulation. Many articles warned the readers of the danger hidden in the modern generators, and actually the poorness of the pseudorandom numbers may lead to wrong results in a simulation[?].

Defective generators are demonstrably defective. We can show the poor randomness of the sequence through statistical tests or theoretical analyses. But, what are good generators? What are "high quality" generators?

At the present state, we do not have a practical theory for good "randomness." Mathematically rigorous theories using computational complexity started by Kolmogorov are not applicable to practical generators, which need to generate pseudorandom numbers with small computation time. The theory states that all the existing implemented generators are not "random" in its sense. There is a more practical notion of "polynomial-time perfect" (see [?] and its references), but this condition is still too restrictive for practically fast generators.

Thus, we desire to have a practical and universal criterion for something like "almost random," but still we do not have one. Accordingly, at the present state, there is no rigorous definition of "good" nor "high quality" generators. In lack of a universal criterion for good randomness, the designers of the random number generators are in an unfortunate situation, as if God broke the Tower of Babel. Each researcher focuses on some special "values" obtained by processing the generated sequence with some statistical, mathematical, or intuitive meanings, and has a belief that to optimize the values is to optimize the pseudorandom number generator. However, this is completely different from optimizing "randomness." The notion that the randomness of any generator can be optimized following any universal set of criteria may itself prove to be an illusion.

Nonetheless, each of the designers insists on his special values. In this paper, we focus only on k-distribution properties, with some justification given in the next section. Others (e.g. [?]) focus on the Liapunov exponent coming from the terminology of dynamical systems. Roughly saying, the Liapunov exponent measures how fast the trajectories diverge in a continuous dynamical system that simulates the random number generator. It is clear that if the Liapunov exponent is small, the next state is predictable from the present state with small errors, and hence the system is far from being random. However, the large Liapunov exponent does not necessarily assure good randomness. The Liapunov exponent gives a decisive criterion on the chaos of a dynamical system. But, "randomness" requires more than "chaotic behavior." Poorness of Liapunov exponent as a measure of randomness manifests itself by the fact that any classical linear congruential generators including the notorious `RANDU` have satisfactorily large Liapunov exponents.

It would be reasonable to say that such a generator, that can be approximated by a continuous dynamical system [?] such as SWB in Marsaglia-Zaman[1] generator [?], is not a good generator.

Lüscher's assertion is that, by discarding large parts, one can get a "high quality" generator, from a poor random number generator such as SWB, because it increases the Liapunov exponent.

We have different opinions. The Liapunov exponent is not a strong measure of randomness. Moreover, there are efficient generators as shown in [?][?][?][?], which are expected to have rather large Liapunov exponents, even discarding nothing. The latter three generators have a theoretical support by the k-distribution property, which will be explained in the next section. So, instead of discarding large parts of poor random sources, we may use such generators.

Discarding does not necessarily extinguish the terrible behavior of trinomial m-sequence reported in [?] and in Table I in [?]. There is a theory in [?] explaining that one should avoid trinomial m-sequences, even after discarding significant parts of the sequence.

We comment that Knuth is going to recommend his generator `ran_array` in his third edition [?], which generates lagged-Fibonacci sequence with some parts regularly discarded, after the Lüscher's method. However, the randomness of the least significant bit of `ran_array` is defective, unless more than 50% of the sequence is discarded. (We shall discuss this defect elsewhere.) Knuth recommends to discard 90%, but then the generation is rather slow.

We hope the users of random number generators consult [?][?][?][?][?] on recent prominent progress of $\mathbb{F}_2$-based generators, which may supplement Knuth's comprehensive book [?].

Another widespread belief is to avoid generators based on Exclusive OR (bitwise addition modulo two). We call this type of generators $\mathbb{F}_2$*-generators* later on. This class includes GFSR[?], Combined Tausworthe[?][?], TGFSR[?], but the latter two

[1]This is an exception among the generators proposed by Marsaglia. It is a sad fact that random number generators with defects are repeatedly cited, although good ones are not so much cited. A combined version of this generator works much better[?], but still has defects from the viewpoint of the lattice structure[?][?]. Marsaglia[?] also proposed a more attractive generalization *multiply with carry*, which may have better lattice structure, see [?][?].

are empirically tested and shown to pass the stringent tests. Sometimes it is asserted (e.g. [?, §4]) that Exclusive OR involves only two bits for generating a next bit, whereas integer addition involves all bits, so addition is better. We do not agree with this. Because addition and subtraction with some modulus are continuous functions, it tends to cause non-randomness if we regard the generated number with topology, i.e., as a uniform random number in the interval [0,1]. Actually this is the reason why the Marsaglia-Zaman generator shows poor randomness (see [?]).

The $\mathbb{F}_2$-generators have significant merits over continuous methods. They use circular-bit-shift and bitmasks [?] [?] [?], which are far from continuous. The decisive merit is that, in the designing stage, we can use the efficient algorithms on polynomial algebras, such as the irreducibility test (see §3.2) or the lattice method[?][?][?][?] (see §5) which the integer algebra method lacks. The new records of the period and the equidistribution property attained in this paper are consequences of these algorithms.

1.3 k-distribution: a reasonable measure of randomness

A more common and admissible belief is that randomness tests of the "spectral test" type is the most powerful and reliable among all the known tests, as stated in [?, §3.3.4]. This test is defined on linear congruential methods, and, surprisingly, all the generators known to show poor randomness in any sense did not pass this test. Roughly saying, the k-dimensional spectral test measures the following value. Let $x_i \, (i = 0, 1, 2, \ldots, P-1)$ be the generated pseudorandom real [0,1] numbers with the period P. Put the P points in the k-dimensional unit cube with coordinates $(x_i, x_{i+1}, \ldots, x_{i+k-1})$ $(i = 0, 1, \ldots, P-1)$, i.e., the consecutive k tuples, for a whole period (the addition in the suffix is considered modulo P). Then, in the k-dimensional cube, P points are scattered. The k-dimensional spectral test measures the distance between the parallel hyperplanes containing all the points. If this size is large, then there is a ball of that size in which the generator never hits a point. Thus the randomness is poor.

Knuth[?] said that these values with dimension $k \leq 10$ are the most powerful and universal measure for randomness, and $k \geq 10$ seem to be less significant practically. This was definitely true at that time, when the linear congruential generators with periods $\sim 2^{32}$ were mainly tested.

Since the period of the generators is increased, we now have to pay more attention to higher dimensional spectral tests. For example, the Marsaglia-Zaman generator, which yields an erroneous result in an Ising model simulation [?], theoretically passes this test up to 43 dimensions. Actually, the generator fails this test in a terrible way for k beyond the lag, i.e. beyond 43 [?]. Thus, nowadays, $k \leq 10$ is not always sufficient. We want k as large as possible. Below, we propose a generator which theoretically passes this type of test at least up to $k = 623$.

One of the suitable variants of the spectral test applicable to $\mathbb{F}_2$-generators is the *k-distribution to v-bit accuracy test*[2]. Again consider the same procedure to put P points in the k-dimensional cube. We equally divide each [0,1] axis into 2^v pieces (in other words, consider only the most significant v bits). Then, we have equated the unit cube into 2^{kv} small cubes. The sequence is said to be k-distributed to v

[2] There are intensive excellent studies on the k-distribution, e.g., [?][?][?][?][?][?][?].

bit accuracy if the P points occur equally likely in each of these small cubes (§5.1).

It is clear that this property is quite similar to the k-dimensional spectral tests. This property assures nonexistence of the correlation among the consecutive k numbers up to v bit precision. $\mathbb{F}_2$-generators with high distribution properties show no defect at the present state, except for those based on trinomials. Trinomial-based $\mathbb{F}_2$-generators have theoretical defects. They do not pass the weight distribution test (see for example [?][?][?]). The $\mathbb{F}_2$-generators leading to erroneous results in [?] are actually trinomial-based.

A cryptographic interpretation. The k-distribution has also a kind of cryptographic interpretation,[3] as follows. Assume that the sequence is k-distributed to v-bit accuracy. Then, knowledge of the most significant v bits of the first l words does not allow the user to make any statement about the most significant v bits of the next word, if $l < k$. This is because every bit-pattern occurs equally likely in the v bits of the next word, by definition of k-distribution.

Thus, if the simulated system is sensitive only to the history of the k or less previously generated words with v-bit accuracy, then it is theoretically safe.

Now, we conclude that the k-distribution to v-bit accuracy is a very reasonable measure of randomness. We do not mean that every random number generator should have a high k-distribution property. A well-selected linear congruential generator is only 1-distributed, but works well enough for most simulations. We claim that a larger value of k is safer from the theoretical point of view mentioned above.

We also take it into consideration that the characteristic polynomial should have many terms. At least, trinomials should be avoided.

1.4 What we obtained: a fast, compact, huge-period generator MT

We introduce an $\mathbb{F}_2$-type generator named *Mersenne Twister* (MT) which satisfies the above criteria very well, compared to any previously existing generators. This is a variant of the TGFSR algorithm introduced in [?], improved in [?], and here modified so as to admit a Mersenne-prime period. The crux of the modification is to use an *incomplete array* (§3.1). This enables us to have a huge period and high dimensional equidistributions. These generators are contained in a class named *multiple-recursive matrix methods* (see §3.4) introduced in [?], and share some of its advantages investigated in detail in [?].

We give an algorithm to obtain good parameters for MT (§4). A set of good parameters is implemented as a portable *C*-code named *MT19937* (see Appendix C). This code can be used in any machine with a standard C compiler (including 64-bit machines), as an integer or real number generator. Essentially same codes are downloadable from the http-site of Salzburg University, `http://random.mat.sbg.ac.at /news/`.

This generator has a tremendously large prime period $2^{19937}-1$, while consuming a working area of only 624 words. The sequence is 623-distributed to 32 bits accuracy. This has a huge k-distribution property to v-bit accuracy for each v, $v = 1, 2, \ldots, 32$. See Table II in §2.2. Here, $k(v)$ denotes the maximum k such

[3] This does not mean "cryptographically secure", see §1.6.

that the sequence is k-distributed with v-bit accuracy. These values are at least ten times larger than any other implemented generators, and are near to the trivial upper bound $\lfloor \frac{\log_2 P}{v} \rfloor$, which is valid for any sequence of period length P (see §5.1), and the digits of our period are ten times larger than those of previously existing generators.

The generator does not satisfy the trivial upper bound deduced from the period. See the difference between the trivial upper bounds and the attained $k(v)$-values in Table II. But this would not be a serious problem, see the arguments in §2.2.

The characteristic polynomial has many terms (roughly 100 or more), and is not derived from some simple deformation of trinomials. (See discussions in §3.5.)

We tested the generator MT19937 (as 32-bit random integer generators) by well-known diehard tests developed by Marsaglia[?], and got good results. Empirical testing of MT19937 using Load Tests and Ultimate Load Tests was done by S. Wegenkittl from the pLab group[?] at the University of Salzburg, and the results were good. These tests rejected most of short period ($\sim 2^{32}$) generators including the one in the ANSI-C standard library, see [?][?].

Here we compare the speed of MT19937 with other modern generators (Table I in §1.5) in a Sun Workstation. MT is comparable to other generators[4], which have much shorter periods. It is twice as fast as `ran_array` in [?] with the "safest method", namely, discarding 90%.

Thus, we conclude that MT is one of the most promising pseudorandom number generators at the present time.

However, it is desirable to apply other statistical tests, too. Stringent tests to criticize MT are welcome.

1.5 Comparison with other generators

The following table shows a comparison of the speed of MT with other generators.

Table I. Cpu-time for 10^7 generations and the working area

ID	COMBO	KISS	`ran_array`	`rand`	`taus88`	TT800	MT19937
cpu − time (sec.)	11.14	9.24	23.23	9.64	7.95	9.97	10.18
working area (words)	4	5	1000	1	3	25	624
period	$\sim 2^{61}$	$\sim 2^{127}$	$\sim 2^{129}$	$\sim 2^{31}$	$\sim 2^{88}$	$2^{800}-1$	$2^{19937}-1$

The combined generators COMBO and KISS are downloaded from Marsaglia's http-site; `http://stat.fsu.edu/~geo/diehard.html`. The generator `ran_array` is Lüscher's discarding method for a lagged-Fibonacci generator, recommended in [?]. The generator `rand` is a standard random number generator of the C-library. The generator taus88 is a combined Tausworthe generator[5] in [?] (c.f. [?]). TT800, a small cousin of MT19937, is a TGFSR generator[6] in [?].

[4] Wegenkittl reported that the speed of MT19937 in the Dec-Alpha machine is even faster than ANSI-C. This high-speed is probably due to the modern hardware architecture like cache memory and pipeline processing, to which MT19937 fits well.

[5] This is a very fast and economical generator, almost flawless in the tests by [?]. We think this good randomness is because the generator is optimized from the view point of k-distribution.

[6] TT800 in [?] is designed as a real number generator, and has a defect at the least significant bits. The downloadable version of TT800 in Salzburg University `http://random.mat.sbg.ac.at/news/` is improved in this regard. This generator and taus88 were the two flawless generators in the

We measured the time consumed in generating 10^7 random numbers in a Sun Workstation. Since `ran_array` discards 90% of the generated sequence, it is much slower than others.

MT19937 and `ran_array` consume more memory[7], but it would not be a major problem in simulations where not so many random number generators run in parallel. MT19937 has the longest period.

Note that the least significant bits of `rand` are rejected by diehard, and the whole bits are rejected in a terrible way in [?][?].

1.6 Limitation and hints for use

This generator does not create cryptographically secure random numbers as it is. For cryptographic purposes, one needs to convert the output by a secure hashing algorithm (see for example [?]). Otherwise, by a simple linear transformation (T^{-1} where T is the tempering matrix given by (2.2)–(2.5) in §2.1), the output of MT19937 becomes a linear recurring sequence given by (2.1) in §2.1. Then, one can easily guess the present state from a sufficiently large size of the output. See the conditions of Proposition 4.2, and note that the recurrence satisfies these conditions.

This generator is developed for generating [0,1]-uniform real random numbers, with special attention paid to the most significant bits. The rejected generators in [?] are exactly the generators whose most significant bits have a defect (see [?] for SWB, and see the weight distribution test in [?] for the trinomial GFSR). Thus, we think our generator would be most suitable for a Monte Carlo simulation such as [?]. If one needs (0,1]-random numbers, simply discard the zeros.

The least significant bits have been also tested by k-distribution tests, and no defects have been found. Thus, one can also use this generator as a 32-bit random integer generator. The results of diehard tests support this. When one needs 64-bit integers, then one may simply concatenate two words.

2. MT ALGORITHM

2.1 Description of MT

Throughout this paper, bold letters, such as $\mathbf{x}$ and $\mathbf{a}$, denote *word vectors*, which are w-dimensional row vectors over the two-element field $\mathbb{F}_2 = \{0, 1\}$, identified with machine words of size w (with the least significant bit at the right).

The MT algorithm generates a sequence of word vectors, which are considered to be uniform pseudorandom integers between 0 and $2^w - 1$. Dividing by $2^w - 1$, we regard each word vector as a real number in [0,1].

Load Tests in [?], in which most of the short-period linear congruential generators and some of the inversive generators are rejected. TGFSR were also tested in [?][?]. We got many emails from the users of TT800 with favorable comments. As far as we know, no test has rejected this generator. We think that this is a consequence of the good k-distribution property of TT800.

[7]The figure for the working area of `ran_array` is perhaps a bit misleading. This figure of 1000 words was attained by choosing "the safest method" in [?], namely, discarding 90% and using Knuth's code. It is easy to reduce the figure to 100 by rewriting the code, but then it becomes slower.

The algorithm is based on the following linear recurrence

$$\mathbf{x}_{k+n} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)A, \quad (k = 0, 1, \cdots). \tag{2.1}$$

We shall explain the notation. We have several constants: an integer n, which is the degree of the recurrence, an integer r (hidden in the definition of $\mathbf{x}_k^u$), $0 \leq r \leq w-1$, an integer m, $1 \leq m \leq n$, and a constant $w \times w$ matrix A with entries in $\mathbb{F}_2$. We give $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ as initial seeds. Then, the generator generates $\mathbf{x}_n$ by the above recurrence with $k = 0$. By putting $k = 1, 2, \ldots$, the generator determines $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots$. In the right hand side of the recurrence, $\mathbf{x}_k^u$ means "the upper $w-r$ bits" of $\mathbf{x}_k$, and $\mathbf{x}_{k+1}^l$ "the lower r bits" of $\mathbf{x}_{k+1}$. Thus, if $\mathbf{x} = (x_{w-1}, x_{w-2}, \ldots, x_0)$, then by definition $\mathbf{x}^u$ is the $w-r$ bits vector $(x_{w-1}, \ldots, x_r)$ and $\mathbf{x}^l$ is the r bits vector $(x_{r-1}, \ldots, x_0)$. $(\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)$ is just the concatenation; namely, this is a word vector obtained by concatenating the upper $w-r$ bits of $\mathbf{x}_k$ and the lower r bits of $\mathbf{x}_{k+1}$ in this order. Then the matrix A is multiplied from the right by this vector. Finally add $\mathbf{x}_{k+m}$ to this vector ($\oplus$ is bitwise addition modulo two), and then we generate the next vector $\mathbf{x}_{k+n}$.

The reason why we chose the complicated recurrence (1) will be clear in §3.1. Here we note that if $r = 0$, then this recurrence reduces to the previous TGFSR proposed in [?][?], and if $r = 0$ and $A = I$, it reduces to GFSR[?].

We choose a form of the matrix A so that multiplication by A is very fast. A candidate is

$$A = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & a_0 \end{pmatrix},$$

then the calculation of $\mathbf{x}A$ can be done using only bit operations:

$$\mathbf{x}A = \begin{cases} \text{shiftright}(\mathbf{x}) & \text{if } x_0 = 0 \\ \text{shiftright}(\mathbf{x}) \oplus \mathbf{a} & \text{if } x_0 = 1 \end{cases},$$

where $\mathbf{a} = (a_{w-1}, a_{w-2}, \cdots, a_0), \mathbf{x} = (x_{w-1}, x_{w-2}, \cdots, x_0)$. Also, $\mathbf{x}_k^u$ and $\mathbf{x}_{k+1}^l$ of the recurrence (2.1) can be calculated with bitwise AND operation. Thus the calculation of the recurrence (2.1) is realized with bitshift, bitwise EXCLUSIVE-OR, bitwise OR, and bitwise AND operations.

For improving the k-distribution to v bit accuracy, we multiply each generated word by a suitable $w \times w$ invertible matrix T from the right (called *tempering* in [?]). For the tempering matrix $\mathbf{x} \mapsto \mathbf{z} = \mathbf{x}T$, we choose the following successive transformations

$$\mathbf{y} := \mathbf{x} \oplus (\mathbf{x} >> u) \tag{2.2}$$
$$\mathbf{y} := \mathbf{y} \oplus ((\mathbf{y} << s) \text{ AND } \mathbf{b}) \tag{2.3}$$
$$\mathbf{y} := \mathbf{y} \oplus ((\mathbf{y} << t) \text{ AND } \mathbf{c}) \tag{2.4}$$
$$\mathbf{z} := \mathbf{y} \oplus (\mathbf{y} >> l), \tag{2.5}$$

where l, s, t, and u are integers, $\mathbf{b}$ and $\mathbf{c}$ are suitable bitmasks of word size, and $(\mathbf{x} >> u)$ denotes the u-bit shiftright ($(\mathbf{x} << u)$ the u-bit shiftleft). The transformations (2.3) and (2.4) are the same as those used in [?]. The transformations (2.2) and (2.5) are added for MT. The transformations (2.3) and (2.4) can change only the most significant bits (MSBs). The transformation (2.5) is added to improve[8] LSBs. Although we do not list them, the k-distributions of LSBs are satisfactorily large. For example, the least significant six bits of MT19937 are 2492-dimensionally equidistributed.

For executing the recurrence (2.1), it is enough to take an array of n words as a working area. Now, the exact algorithm can be stated as follows. Let $\mathbf{x}[n]$ be an array of n unsigned integers of word size, i be an integer variable, and $\mathbf{u}, \mathbf{ll}, \mathbf{a}$ be unsigned constant integers of word size.

Step 0. $\mathbf{u} \leftarrow \underbrace{1\cdots1}_{w-r}\underbrace{0\cdots0}_{r}$;(bitmask for upper $w-r$ bits)

$\mathbf{ll} \leftarrow \underbrace{0\cdots0}_{w-r}\underbrace{1\cdots1}_{r}$;(bitmask for lower r bits)

$\mathbf{a} \leftarrow a_{w-1}a_{w-2}\cdots a_1a_0$;(the constant vector appearing in the matrix A)

Step 1. $i \leftarrow 0$

$\mathbf{x}[0], \mathbf{x}[1], \cdots, \mathbf{x}[n-1] \leftarrow$ "any non-zero initial values"

Step 2. $\mathbf{y} \leftarrow (\mathbf{x}[i] \text{ AND } \mathbf{u}) \text{ OR } (\mathbf{x}[(i+1) \bmod n] \text{ AND } \mathbf{ll})$;(computing $(\mathbf{x}_i^u | \mathbf{x}_{i+1}^l)$)

Step 3. $\mathbf{x}[i] \leftarrow \mathbf{x}[(i+m) \bmod n] \text{ XOR } (\mathbf{y} >> 1)$

$\text{XOR} \begin{cases} 0 & \text{if LSB of } \mathbf{y} = 0 \\ \mathbf{a} & \text{if LSB of } \mathbf{y} = 1 \end{cases}$;(multiplying A)

Step 4. ;(calculate $\mathbf{x}[i]T$)

$\mathbf{y} \leftarrow \mathbf{x}[i]$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } (\mathbf{y} >> u)$;(shiftright $\mathbf{y}$ by u bits and add to $\mathbf{y}$)

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } ((\mathbf{y} << s) \text{ AND } \mathbf{b})$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } ((\mathbf{y} << t) \text{ AND } \mathbf{c})$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } (\mathbf{y} >> l)$

output $\mathbf{y}$

Step 5. $i \leftarrow (i+1) \bmod n$

Step 6. Goto Step 2.

By rewriting the whole array at one time, we can dispense with modulo-n operations. Thus, we need only very fast operations, such as addition, subtraction, and bit-operations (see the code in Appendix C).

We have the following two classes of parameters: (1) parameters determining the period: integer parameters w (word size), n (degree of recursion), m (middle term), r (separation point of one word), and a vector parameter $\mathbf{a}$ (matrix A), which we call *period parameters*, and (2) parameters for k-distribution to v bits accuracy (i.e. for tempering): integer parameters l, u, s, t and the vector parameters $\mathbf{b}, \mathbf{c}$, which we call *tempering parameters*.

[8]This does not exist in the TT800 code in [?]. The code in Salzburg http was improved in this regard by adding (2.5).

2.2 Good Parameters with large k-distributions

Table II lists some period parameters which attains the maximal period $2^{nw-r}-1$, and tempering parameters with good k-distribution property.

The values $k(v)$ denote the maximum k such that the sequence is k-distributed to v-bit accuracy.

The trivial upper bound $k(v) \leq \lfloor \frac{nw-r}{v} \rfloor$ (see §5.1) is shown in the same table. This table shows that we could not attain these bounds even after tempering. We show some tighter bounds in Appendix B. The table shows that $k(v)$ tends to be near a multiple of n. We prove this only for $k(v)$ less than $2(n-1)$, see Proposition B.2. This proposition explains why $k(v)$ cannot be near to the bound $\lfloor \frac{nw-r}{v} \rfloor$ if $\lfloor \frac{nw-r}{v} \rfloor < 2(n-1)$. We expect that there is a more general obstruction, as in the case of [**?**], but right now we obtain only the results in Appendix B.

One may argue that the gap between the bounds and the attained values is a problem, see [**?**]. In our opinion, "to attain a larger $k(v)$" is usually more important than "to attain the upper bound in a limited working area" (although this depends on the memory-restriction).

Knuth[**?**] kindly suggested to us that we should pay more attention to LSBs also. The value l in the table was determined to get a good k-distribution property for the LSBs. Although we do not list them, the k-distributions of LSBs are also satisfactorily high, similarly to MSBs.

The number of terms of the characteristic polynomial is also shown under the ID.

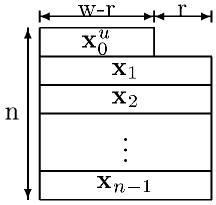
Table II. Parameters and k-distribution of Mersenne Twisters

Generator		The order of equidistribution					
ID	Parameters	$k(1)$	$k(2)$	$k(3)$	$k(4)$	$k(5)$	$k(6)$
		$k(7)$	$k(8)$	$k(9)$	$k(10)$	$k(11)$	$k(12)$
(the number of		$k(13)$	$k(14)$	$k(15)$	$k(16)$	$k(17)$	$k(18)$
terms in the		$k(19)$	$k(20)$	$k(21)$	$k(22)$	$k(23)$	$k(24)$
characteristic		$k(25)$	$k(26)$	$k(27)$	$k(28)$	$k(29)$	$k(30)$
polynomial)		$k(31)$	$k(32)$				
	Upper bounds	11213	5606	3737	2803	2242	1868
	$\lfloor \frac{nw-r}{v} \rfloor$	1601	1401	1245	1121	1019	934
	$(w,n,r) = (32,351,19)$	862	800	747	700	659	622
	$1 \le v \le 32$	590	560	533	509	487	467
		448	431	415	400	386	373
		361	350				
MT11213A	$(w,n,m,r) = (32,351,175,19)$	11213	5606	3560	2803	2111	1756
	a = E4BD75F5	1405	1401	1055	1053	709	704
	u = 11	703	702	701	700	356	352
	s = 7,b = 655E5280	351	351	351	350	350	350
(177)	t = 15,c = FFD58000	350	350	350	350	350	350
	l = 17	350	350				
MT11213B	$(w,n,m,r) = (32,351,175,19)$	11213	5606	3565	2803	2113	1759
	a = CCAB8EE7	1408	1401	1056	1053	715	704
	u = 11	702	702	701	700	355	352
	s = 7,b = 31B6AB00	351	351	351	351	350	350
(151)	t = 15,c = FFE50000	350	350	350	350	350	350
	l = 17	350	350				
	Upper bounds	19937	9968	6645	4984	3987	3322
	$\lfloor \frac{nw-r}{v} \rfloor$	2848	2492	2215	1993	1812	1661
	$(w,n,r) = (32,624,31)$	1533	1424	1329	1246	1172	1107
	$1 \le v \le 32$	1049	996	949	906	866	830
		797	766	738	712	687	664
		643	623				
MT19937	$(w,n,m,r) = (32,624,397,31)$	19937	9968	6240	4984	3738	3115
	a = 9908B0DF	2493	2492	1869	1869	1248	1246
	u = 11	1246	1246	1246	1246	623	623
	s = 7, b = 9D2C5680	623	623	623	623	623	623
(135)	t = 15, c = EFC60000	623	623	623	623	623	623
	l = 18	623	623				
TT800	$(w,n,m,r) = (32,25,7,0)$	800	400	250	200	150	125
	a = 8EBFD028	100	100	75	75	50	50
	u : not exist	50	50	50	50	25	25
	s = 7, b = 2B5B2500	25	25	25	25	25	25
(93)	t = 15, c = DB8B0000	25	25	25	25	25	25
	l =16	25	25				
`ran_array`	Knuth's new recommendation.	129	64	43	32	25	21
		18	16	14	12	11	10
	Here we list the trivial	9	9	8	8	7	7
	upper bounds.	6	6	6	5	5	5
		5	4	4	4	4	4

3. WHY MT? MERITS AND HISTORY

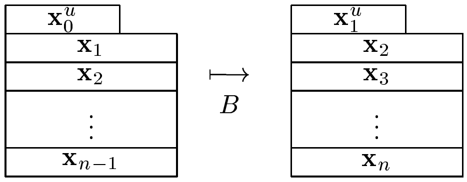
3.1 How we reached MT: Incomplete arrays

As is the case of any $\mathbb{F}_2$-linear generating method, the MT algorithm is just an iteration of a fixed linear transformation on a fixed vector space. In the case of MT, the vector space is the set of the following type of arrays of (0,1)-entries, with r bits missing at the upper-right corner.



For a w-bit word $\mathbf{x}$, $\mathbf{x}^u$ denotes the most significant $w-r$ bits of $\mathbf{x}$. We call this object an $(n \times w - r)$-*array* or an *incomplete array.* This is a key idea in implementing an efficient recurrence on a Mersenne-exponent dimensional space.

The state transition is given by



,

where $\mathbf{x}_n$ is obtained by the defining recurrence (2.1) for $k = 0$. This transformation B is obviously linear, and by a general theory of linear recurrence (see Appendix A), each entry of the $(n \times w - r)$-array is a linear recurring sequence satisfying the equation corresponding to the characteristic polynomial $\varphi_B(t)$ of the transformation B.

The sequence attains the maximal period $2^p - 1 = 2^{nw-r} - 1$, if and only if $\varphi_B(t)$ is primitive, i.e, t is in the multiplicative group $(\mathbb{F}_2[t]/\varphi_B(t))^\times$ and its order is $2^p - 1$.

A great advantage from attaining this bound is that the state vector assumes every bit-pattern in the $(n \times w - r)$-array, once in a period, except for the zero state. Consequently, the sequence is $(n-1)$-distributed. Since any initial seed except for zero lies on the same orbit, the choice of an initial seed does not affect the randomness for the whole period. This is much different from the original GFSR, in which the initialization is critical [?]. This easiness of initialization is a merit of TGFSR. Note that some of the Marsaglia-Zaman generators also have this property. Knuth's `ran_array` does not have this property. Instead, a clever initialization technique is used for avoiding defective initial seeds.

Since $(n-1)$ is the order of equidistribution, we would like to make n as large as the memory restriction permits. We think that in recent computers n up to 1000 is reasonable. (Of course, it depends on the application. In an application where many generators are involved, this may be memory consuming.) On the other hand, one may claim n up to 10 would be enough. However, for example, one of the Marsaglia-Zaman generators is 43-distributed since the orbit is one (see [?]), but failed in the Ising-Model test [?]. The system simulated there has a "good memory" remembering a large number of previously generated words. There are such applications where the n-dimensional distributions for very large n become important.

Thus, it is better to take n as large as possible. But an essential limitation comes from the difficulty of factorization. We have to calculate the order of t modulo $\varphi_B(t)$, and certify it to be $2^p - 1$. For this, we need all the proper factors of $2^p - 1$ and check that the order of t does not divide them. Unfortunately, even a modern technique can factorize $2^p - 1$ only for around $p < 2000$ (see for example [?]).

On the other hand, the test of primality of an integer is much easier. So, there are many *Mersenne* primes (i.e., primes of the form $2^p - 1$) found, up to $p = 1398269$ (see `http://www.utm.edu:80/research/primes/mersenne.shtml#known`).

However, the previous TGFSR does not admit Mersenne primes. TGFSR generators are MT generators with $r = 0$. In this case, $p = nw$, and $2^{nw} - 1$ can never be a prime, unless n or w is 1. Thus, $nw < 2000$ is an essential limitation.

But if we eliminate r bits from the $(n \times w)$-array, as in MT, then the dimension of the state space is $nw - r$. One can attain any number in this form, including Mersenne exponents. In this case, since $2^{nw-r} - 1$ is a prime, we have already factorized it. This is the reason why we use an $(n \times w - r)$-array.

In determining the next state, each bit of $\mathbf{x}_0^u$ and $\mathbf{x}_1^l$ must be fully reflected, since otherwise the state space is smaller. Thus, results the recurrence (2.1).

Knuth[?] also informed us of the following justification of this recurrence. One might have used $(\mathbf{x}_{k+1}^l | \mathbf{x}_k^u)$ instead of $(\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)$ in the recurrence (2.1). The former seems more natural, since then for example the matrix S in Appendix A coincides with A. But he noticed that when $r = w - 1$, then the sequence can never have maximal period. Actually, it is easy to check that the most significant bit of each generated word satisfies a trinomial linear recurrence with order n, and this does not satisfy the maximality.

3.2 Primitivity is easy for MT

Another justification of the recurrence (2.1) is that the primitivity can be easily checked, by *inversive-decimation methods*(§4.3). Since we chose a Mersenne-exponent $p = nw - r$ as the size of the incomplete array, there is an algorithm to check the primitivity with $O(lp^2)$ computations, where l is the number of nonzero terms in the characteristic polynomial. The easiest case is $l = 3$, and accordingly there is a list up to $p = 132049$ for trinomials [?].

Thus, we might have used this list. One can implement a recurrence with such a characteristic trinomial in an incomplete array (an easy exercise). Why didn't we do that?

The trinomials always show erroneous weight distributions, and some of many-term polynomials obtained by modifying trinomials have same tendency, see §3.5. The characteristic polynomial of MT has no obvious relation with trinomials. The number of terms in the characteristic polynomial is experimentally $l \geq 100$ (see Table II). In spite of these many terms, because of the peculiar form of the recurrence (2.1), the primitivity of this recurrence can be checked with $O(p^2)$ computations (see §4.3), and the generation is also efficient (see Table I).

Note that for large-modulus generators, the primitivity check is a hard number theoretic task (e.g. [?]). This is an advantage of $\mathbb{F}_2$-generators over integer-operation generators.

3.3 k-distribution is easy for MT

It was discovered by Tezuka[?] that the k-distribution to 2-bit accuracy of TGFSR in the first paper [?] is very low. A follow up to this failure was satisfactorily completed in [?].

By the same reason stated in [?], the k-distribution property of the raw sequence generated by the recurrence (2.1) is poor, so we need to modify the output by multiplying by a matrix T (i.e., *tempering*, see section §5). Then we succeed in realizing good k-distributions.

Here we comment that spectral tests with dimension more than 100 are almost impossible for computational reasons, for multiple linear congruential methods, or any existing generators based on large-modulus calculus. On the other hand, for MT, we can execute k-distribution tests to v-bit accuracy, for k more than 600. This is a great advantage of $\mathbb{F}_2$-generators over large-modulus generators.

3.4 MT is one of multiple-recursive matrix methods (MRMM)

Soon after TGFSR was proposed, Niederreiter developed a general class of random number generators including TGFSR, *multiple-recursive matrix methods* (MRMM) [?][?]. MT also belongs to this class. MRMM is to generate a random vector sequence over $\mathbb{F}_2$ by the linear recurrence

$$\mathbf{x}_{k+n} := \mathbf{x}_{k+n-1}A_{n-1} + \cdots \mathbf{x}_{k+1}A_1 + \mathbf{x}_k A_0 \quad (k = 0, 1, \cdots),$$

where $\mathbf{x}_k$ are row vectors and A_i are $w \times w$ matrices. MT belongs to this class, since the defining recurrence (2.1) of MT can be written as

$$\mathbf{x}_{k+n} = \mathbf{x}_{k+m} + \mathbf{x}_{k+1} \begin{pmatrix} 0 & 0 \\ 0 & I_r \end{pmatrix} A + \mathbf{x}_k \begin{pmatrix} I_{w-r} & 0 \\ 0 & 0 \end{pmatrix} A,$$

where I_r, I_{w-r} is the identity matrix of size r, $w-r$, respectively.

Even after tempering, the generated sequence still belongs to this class. It is easy to see from the definition that a sequence of word vectors belongs to this class if and only if $\mathbf{x}_{k+n}$ is determined by a linear transformation from the preceding n vectors $\mathbf{x}_{k+n-1}, \mathbf{x}_{k+1}, \ldots, \mathbf{x}_k$. (Thus, this is nothing but a linear recurring sequence of vector values, as stated in [?]. The important point of [?] is to analyze the properties such as discrepancy.) Since the tempering matrix is a linear isomorphism, it preserves this property. Thus, MT can be said to be a neat implementation of the general concept MRMM.

Unfortunately, the detailed investigation in [?] is not applicable as it is, since he mainly considered only the case with the maximal period $2^{nw}-1$. As we explained, the crux of MT is to admit a Mersenne prime of the form $2^{nw-r}-1$. A modification of Niederreiter's work to cover MT would be possible and valuable. We guess that MT's performance would not be so much different from those considered in [?].

3.5 Number of terms

The number of the terms of the characteristic polynomial is discussed in many papers; for example [?][?]. We do not repeat them here, but just comment that this number is invariant by any $\mathbb{F}_2$-linear transformation on the output sequence, such as tempering.

As discussed in the above-mentioned papers, trinomial-based $\mathbb{F}_2$-generators tend to show poor randomness. They may lead to erroneous results in simulation (see trinomial-based GFSR in [?]).

We here warn the reader that there are "improved" $\mathbb{F}_2$-generators derived from trinomial generators, for example those in [?]. The characteristic polynomials of these generators have many terms, but they divide trinomials with relatively small degree (see [?] for the danger of this fact).

These generators have similar defects to those based on trinomials, and do not pass the weight distribution test in [?]. See also the result of the test in [?] of Kashiwagi's pentanomial generators. The tests said that even taking only every 33rd bit, the sequence still showed strong deviations from randomness. This phenomenon seems essentially different from an analogous one for linear congruential generators, where the lattice structure becomes more evident by taking subsequences.

TGFSR generators including MT have characteristic polynomials with many terms, see Table II.

4. HOW TO FIND PERIOD PARAMETERS

4.1 The difficulty in obtaining the period

Since we have chosen n and r so that $nw - r = p$ is a large Mersenne exponent, the primitivity can be checked by (see for example [?])

$$\begin{cases} t^2 \not\equiv t \mod \varphi_B(t) \\ t^{2^p} \equiv t \mod \varphi_B(t) \end{cases}.$$

It is possible to calculate this directly, as was done previously. (See Appendix A1 for the explicit form of $\varphi_B(t)$). However, this is an $O(lp^2)$-calculation, where l is the number of terms. To take the square modulo $\varphi_B(t)$, we need to divide a polynomial of degree $2p$ by $\varphi_B(t)$. For this, we need $O(p)$-times subtraction by $\varphi_B(t)$, and each subtraction requires $O(l)$-operations. We iterate this p times, which amounts to $O(lp^2)$. In our case, p is very large (> 10000), and according to our experiment, the direct computation may need several years to catch a primitive polynomial.

We contrived an algorithm called the *inversive-decimation method* with $O(p^2)$ operations for the primitivity test for MT. This algorithm may be used for other generators as well, if the generator satisfies the condition of Proposition 4.2 below.

4.2 A criterion for primitivity

Let $\mathcal{S}^\infty$ denote the $\mathbb{F}_2$-vector space of all infinite sequences of 0,1. That is,

$$\mathcal{S}^\infty := \{\chi = (\cdots, x_5, x_4, x_3, x_2, x_1, x_0) \mid x_i \in \mathbb{F}_2\}.$$

Let D (*delay* operator) and H (*decimation* operator) be linear operators from $\mathcal{S}^\infty$ to $\mathcal{S}^\infty$ defined by

$$D(\cdots, x_4, x_3, x_2, x_1, x_0) = (\cdots, x_5, x_4, x_3, x_2, x_1),$$

$$H(\cdots, x_4, x_3, x_2, x_1, x_0) = (\cdots, x_{10}, x_8, x_6, x_4, x_2, x_0).$$

Let $\varphi(t)$ be the characteristic polynomial of a linear recurrence. Then, χ satisfies the recurrence if and only if $\varphi(D)\chi = 0$.

It is easy to check that

$$DH = HD^2.$$

Since the coefficients are in $\mathbb{F}_2$, we have $\varphi(t^2) = \varphi(t)^2$, and thus if $\varphi(D)\chi = 0$ then

$$\varphi(D)H\chi = H\varphi(D^2)\chi = H\varphi(D)^2\chi = 0,$$

i.e., $H\chi$ also satisfies the same recurrence.

It is easy to see that if the period of $\chi \in \mathcal{S}^\infty$ is $2^p - 1$ then $H^p\chi = \chi$, but the converse may not be true. However, the following theorem holds.

THEOREM 4.1. *Let $\varphi(t)$ be a polynomial over $\mathbb{F}_2$ whose degree p is a Mersenne exponent. Take $\chi \in \mathcal{S}^\infty$ such that $\varphi(D)\chi = 0$ and $H\chi \neq \chi$. Then $\varphi(t)$ is primitive if and only if $H^p\chi = \chi$.*

Proof. Let V be the p-dimensional linear space

$$V := \{\chi \in \mathcal{S}^\infty \mid \varphi(D)\chi = 0\},$$

and τ be a linear mapping from $\mathcal{S}^\infty$ to $\mathbb{F}_2$ defined by

$$\tau(\cdots, x_2, x_1, x_0) = x_0.$$

We consider a bilinear pairing $(a|b)$ by

$$\begin{array}{ccccc} \mathbb{F}_2[t]/\varphi(t) & \times & V & \to & \mathbb{F}_2 \\ g(t) & , & \chi & \mapsto & (g(t)|\chi) := \tau(g(D)\chi). \end{array}$$

This is well-defined, and non-degenerate because if $\tau(g(D)\chi) = 0$ for all $g(D)$, then $\chi = 0$ follows from $\tau(D^n\chi) = 0$ for all n.

Let F be a mapping from $\mathbb{F}_2[t]/\varphi(t)$ to $\mathbb{F}_2[t]/\varphi(t)$ given by $F(g(t)) = g(t)^2$. Then it is easy to check that F is the adjoint of H, i.e.,

$$(F(g(t))|\chi) = (g(t)|H\chi)$$

holds (it is enough to consider the case of $g(t) = t$).

The condition of the theorem is

$$\mathrm{Ker}(H^p - 1) \supsetneq \mathrm{Ker}(H - 1),$$

which is now equivalent to

$$\mathrm{Ker}(F^p - 1) \supsetneq \mathrm{Ker}(F - 1).$$

This means the existence of $g(t) \in \mathbb{F}_2[t]/\varphi(t)$ such that $g(t)^{2^p} = g(t)$ and $g(t)^2 \neq g(t)$.

Let l ($\geq$1) be the smallest integer such that $g(t)^{l+1} = g(t)$. Then, since $g(t)^{2^p} = g(t)$, it follows that $l | 2^p - 1$, and $l \neq 1$ by the assumption. Since p is a *Mersenne* exponent, l must be at least $2^p - 1$. Since 0 is an orbit, this means that all non-zero elements lie on one orbit, and it is purely periodic. Since this orbit contains 1, $g(t)$ is invertible and it must be a generator of $(\mathbb{F}_2[t]/\varphi(t))^\times$. Moreover, the order of $(\mathbb{F}_2[t]/\varphi(t))^\times$ must be $2^p - 1$. Then $\mathbb{F}_2[t]/\varphi(t)$ is a field, so $\varphi(t)$ is primitive. □

4.3 The Inversive-decimation method for the primitivity testing

PROPOSITION 4.2. (*Inversive-decimation method*)
Let V be a p-dimensional state vector space over $\mathbb{F}_2$, where p is a Mersenne-exponent. Let $f : V \to V$ be a linear state transition map. Let $b : V \to \mathbb{F}_2$ be a linear map (e.g. looking up one bit from the state). Assume that f and b are computable in $O(1)$-time.
Assume that $\Phi : V \to {\mathbb{F}_2}^p$ given by

$$\Phi : S \mapsto (bf^{p-1}(S), bf^{p-2}(S), \ldots, bf(S), b(S))$$

is bijective, and that the inverse map is computable with time-complexity $O(p)$.
Then, primitivity of the characteristic polynomial of f can be tested with time-complexity $O(p^2)$.

Note that the last condition is essential. The other conditions are automatically satisfied for most of efficient $\mathbb{F}_2$-generators. In order to apply this algorithm, to find a good b satisfying the last condition is the crux.

PROOF. Let χ be the infinite sequence $(\cdots, bf^2(S), bf(S), b(S))$. Since Φ is invertible with order $O(p)$, we can choose such an S with $H(\chi) \neq \chi$.
By Theorem 4.1, it is enough to show that the first p bits of $H^{m+1}(\chi)$ can be obtained from the first p bits of $H^m(\chi)$ with an $O(p)$-calculation. From the first p bits of $H^m(\chi)$, we can obtain a state S_m which yields $H^m(\chi)$ by using the inverse of Φ with $O(p)$ computations. From this state, we can generate the first $2p$ bits of $H^m(\chi)$ with an $O(2p)$ calculation, since f and b are of $O(1)$. Then decimate these $2p$ bits. Now we obtain the first p bits of H^{m+1}, with an $O(p)$ calculation. □

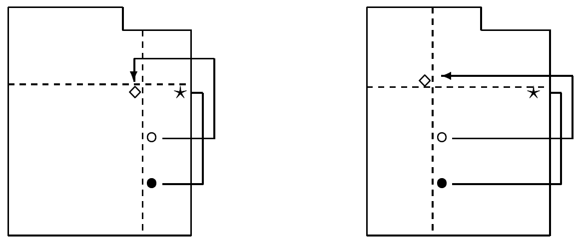
The above proposition can be applied to MT in the following way. For simplicity, assume $r > 0$. Put $S = (\mathbf{x}_{n-1}, \cdots, \mathbf{x}_1, \mathbf{x}_0^u)$, i.e, an initial $(n \times w - r)$-array.
Let b be the map that takes the upper-right corner of this incomplete array. Thus, $b(S) = x_{1,0}$, the least significant bit of $\mathbf{x}_1$. We have to find an inverse morphism to Φ, which calculates from $(x_{p,0}, x_{p-1,0}, \ldots, x_{1,0})$ the state S that produces this p-bit stream at the LSB, with only $O(p)$-calculation.
If $x_{p-n+1,0}, x_{p-n+m,0}$ and $x_{p,0}$ are known, we can calculate $x_{p-n+1,1}$ if $r > 1$, or $x_{p-n,1}$ if $r \leq 1$. This is because by Step 2 and Step 3 of the algorithm in §2.1, the following equation holds between $x_{p-n+1,0}, x_{p-n+m,0}, x_{p,0}$ and $x_{p-n+1,1}$.

$$x_{p-n+1,1} = \begin{cases} x_{p-n+m,0} \oplus x_{p,0} & \text{if } x_{p-n+1,0} = 0 \\ x_{p-n+m,0} \oplus x_{p,0} \oplus a_0 & \text{if } x_{p-n+1,0} = 1. \end{cases}$$

It is clear that the same relation holds between $x_{i-n+1,0}$,$x_{i-n+m,0}$,$x_{i,0}$ and $x_{i-n+1,1}$ for $i = n,\ n+1, \cdots, p$. Thus from $x_{1,0}, \cdots, x_{p,0}$, we can calculate $x_{1,1}, \cdots, x_{p-n+1,1}$. In general, for $i = n, n+1, \cdots, p$, $j = 1, 2, \cdots, w-1$, the following equations hold:

$$x_{i-n+1,j} = \begin{cases} x_{i-n+m,j-1} \oplus x_{i,j-1} & \text{if } x_{i-n+1,0} = 0 \\ x_{i-n+m,j-1} \oplus x_{i,j-1} \oplus a_{j-1} & \text{if } x_{i-n+1,0} = 1 \end{cases} \quad (j < r)$$

$$x_{i-n,j} = \begin{cases} x_{i-n+m,j-1} \oplus x_{i,j-1} & \text{if } x_{i-n+1,0} = 0 \\ x_{i-n+m,j-1} \oplus x_{i,j-1} \oplus a_{j-1} & \text{if } x_{i-n+1,0} = 1 \end{cases} \quad (j \geq r).$$



$j < r$

$$\begin{cases} \diamond = & x_{i-n+1,j} \\ \bullet = & x_{i,j-1} \\ \circ = & x_{i-n+m,j-1} \\ \star = & x_{i-n+1,0} \end{cases}$$

$j \geq r$

$$\begin{cases} \diamond = & x_{i-n,j} \\ \bullet = & x_{i,j-1} \\ \circ = & x_{i-n+m,j-1} \\ \star = & x_{i-n+1,0} \end{cases}$$

If $x_{1,0},\cdots,x_{k,0}$ and $x_{m,j},x_{m+1,j},\cdots,x_{k+n-1,j}$ $(n \leq k+n-1 \leq p)$ are known, then $x_{1,j+1},x_{2,j+1},\cdots,x_{k,j+1}$ (if $j < r-1$), or $x_{0,j+1},x_{1,j+1},\cdots,x_{k-1,j+1}$ (if $j \geq r-1$) can be calculated. Furthermore, from $(x_{i,j-1},\cdots,x_{i,0})$, $(x_{i-n+m,j-1},\cdots,x_{i-n+m,0})$, and $x_{i-n+1,0}$, we can calculate $(x_{i-n,j},\cdots,x_{i-n,r},x_{i-n+1,r-1},\cdots,x_{i-n+1,1}, x_{i-n+1,0})$, i.e., the lower $(j+1)$ bits of $(\mathbf{x}^u_{i-n}|\mathbf{x}^l_{i-n+1})$ at the same time, by

$$\begin{aligned} (x_{i-n,j},\cdots,x_{i-n,r},x_{i-n+1,r-1},\cdots,x_{i-n+1,1},x_{i-n+1,0}) = \\ (0,0,\cdots,0,x_{i-n+1,0}) \\ + \ (x_{i,j-1},\cdots,x_{i,1},x_{i,0},0) \\ + \ (x_{i-n+m,j-1},\cdots,x_{i-n+m,1},x_{i-n+m,0},0) \\ + \begin{cases} 0 & \text{if } x_{i-n+1,0} = 0 \\ (a_{j-1}\cdots,a_1,a_0,0) & \text{if } x_{i-n+1,0} = 1 \end{cases}. \end{aligned}$$

So, by setting $\mathbf{y}_0 = 0$, $\mathbf{y}_i = (0,\cdots,0,x_{i,0})$ $(i = 1,\cdots,p)$ and repeating the following recurrence from $i = p$ until $i = n$,

$$\mathbf{y} \leftarrow \mathbf{y}_i + \mathbf{y}_{i-n+m} + \begin{cases} 0 & \text{if LSB of } \mathbf{y}_{i-n+1} = 0 \\ \mathbf{a} & \text{if LSB of } \mathbf{y}_{i-n+1} = 1 \end{cases}$$
$$(\mathbf{y}^u_{i-n}|\mathbf{y}^l_{i-n+1}) \leftarrow \text{shiftleft}(\mathbf{y}) + (0,\cdots,0,x_{i-n+1,0})$$

we get $S = (\mathbf{y}_{n-1},\cdots,\mathbf{y}_0)$. Now Proposition 4.2 can be applied.

4.4 Description of the algorithm for the primitivity test

We now state the above algorithm to check the maximality of the period for a tuple of parameters $(w,n,m,r,\mathbf{a})$.

Let $\mathbf{x}[n]$, $\mathbf{initial}[n]$ be arrays of n unsigned w-bit integers, i be an integer variable, and **u, ll, a** unsigned w-bit integers.

Step 1. $i \leftarrow 0$.
$\mathbf{initial}[j], \mathbf{x}[j] \leftarrow$ some suitable initial values such that $\mathbf{x}[2] \neq \mathbf{x}[3]$

$\mathbf{u} \leftarrow \underbrace{1\cdots1}_{w-r}\underbrace{0\cdots0}_{r}$
$\mathbf{ll} \leftarrow \underbrace{0\cdots0}_{w-r}\underbrace{1\cdots1}_{r}$
$\mathbf{a} \leftarrow a_{w-1}a_{w-2}\cdots a_1 a_0$

Step 2. Generate $2p-n$ times

Step 3. $\mathbf{x}[0] \leftarrow \mathbf{x}[0], \mathbf{x}[1] \leftarrow \mathbf{x}[1], \mathbf{x}[j] \leftarrow \mathbf{x}[2j-1]$ (where $j = 2, 3, \cdots, p$)

Step 4. $k \leftarrow p$.

Step 5. $\mathbf{y} \leftarrow \mathbf{x}[k] \oplus \mathbf{x}[k-n+m] \oplus \begin{cases} 0 & \text{if LSB of } \mathbf{x}[k-n+1] = 0 \\ \mathbf{a} & \text{if LSB of } \mathbf{x}[k-n+1] = 1 \end{cases}$
$\mathbf{y} \leftarrow \text{shiftleft}(\mathbf{y})$. Set LSB of $\mathbf{y}$ to LSB of $\mathbf{x}[k-n+1]$.
$\mathbf{x}[k-n+1] \leftarrow (\mathbf{u} \text{ AND } \mathbf{x}[k-n+1]) \text{ OR } (\mathbf{ll} \text{ AND } \mathbf{y})$
$\mathbf{x}[k-n] \leftarrow (\mathbf{u} \text{ AND } \mathbf{y}) \text{ OR } (\mathbf{ll} \text{ AND } \mathbf{x}[k-n])$
$k \leftarrow k-1$

Step 6. if $k = n-1$ goto **Step 7** else goto **Step 5**.

Step 7. $i \leftarrow i+1$.
if $i = p$ then goto **Step 8**, else goto **Step 2**.

Step 8. if $(\mathbf{x}[0] \text{ AND } \mathbf{u}) = (\mathbf{initial}[0] \text{ AND } \mathbf{u})$ and $\mathbf{initial}[j] = \mathbf{x}[j]$ $(j = 1, 2, \cdots, n-1)$ then the period is $2^p - 1$ else the period is not $2^p - 1$.

5. HOW TO FIND TEMPERING PARAMETERS

5.1 k-distribution to v-bit accuracy

DEFINITION 5.1. *A pseudorandom sequence $\mathbf{x}_i$ of w-bit integers of period P satisfying the following condition is said to be k-distributed to v-bit accuracy: let $trunc_v(\mathbf{x})$ denote the number formed by the leading v bits of $\mathbf{x}$, and consider P of the kv-bit vectors*

$$(trunc_v(\mathbf{x}_i), trunc_v(\mathbf{x}_{i+1}), \cdots, trunc_v(\mathbf{x}_{i+k-1})) \ (0 \le i < P).$$

Then, each of the 2^{kv} possible combinations of bits occurs the same number of times in a period, except for the all-zero combination that occurs once less often.

For each $v = 1, 2, \cdots, w$, let $k(v)$ denote the maximum number such that the sequence is $k(v)$-distributed to v-bit accuracy. Note that the inequality $2^{k(v)v} - 1 \le P$ holds, since at most P patterns can occur in one period, and the number of possible bit patterns in the most significant v bits of the consecutive $k(v)$ words is $2^{k(v)v}$. Since we admit a flaw at zero, we need to add -1. We call this *the trivial upper bound.* In the case of MT, $P = 2^{nw-r} - 1$ holds, so we have an upper bound $k(v) \le \lfloor \frac{nw-r}{v} \rfloor$.

To compute $k(v)$ we use the *lattice* method developed in [?][?][?]. Note that this method heavily depends on the algorithm in [?] to find the successive minima in the formal power series over $\mathbb{F}_2$ (see also [?]). In [?] we computed the k-distribution by obtaining the rank of a matrix, but this time we could not do that because of the computational complexity $O(p^3)$. The resolution-wise lattice invented in [?] works very well.

5.2 Using the lattice structure

Here briefly we recall the method to obtain $k(v)$ by using the lattice. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i, \ldots$ be a sequence in which each bit satisfies one common linear recurrence with primitive characteristic polynomial $\varphi(t)$. Thus, if we put $\mathbf{x}_i = (x_{i,w-1}, \ldots, x_{i,0})$, then the infinite sequences $\chi_{w-1} = (x_{0,w-1}, x_{1,w-1}, \ldots, x_{i,w-1}, \ldots)$, $\cdots$,$\chi_0 = (x_{0,0}, x_{1,0}, \ldots, x_{i,0}, \ldots)$ are subject to the recurrence given by $\varphi(t)$ (MT satisfies this, see Appendix A2).

Now, the l-th k-tuple up to v-bit accuracy is $(\mathrm{first}_k(D^l\chi_{w-1}), \ldots, \mathrm{first}_k(D^l\chi_{w-v}))$, where first_k denotes the first k bits of the sequence. Hence k-distribution to v bit accuracy is equivalent to the surjectivity of

$$l \mapsto (\mathrm{first}_k(D^l(\chi_{w-1})), \ldots, \mathrm{first}_k(D^l(\chi_{w-v})),$$

as a map from the integers to the nonzero vectors in the $(v \times k)$-dimensional space over $\mathbb{F}_2$. (Then the multiplicity in one small cube would be 2's power to the difference between the dimension of the state space and $v \times k$.) To obtain the maximal $k =: k(v)$ so that the above map is surjective, we use the lattice structure. Let K be the field of Laurent power series:

$$K = \left\{ \sum_{j=-n}^{\infty} \alpha_j t^{-j} \mid \alpha_j \in \mathbb{F}_2, n \in \mathbb{Z} \right\}.$$

We identify each χ_i with a Laurent power series by

$$\chi_i := \sum_{j=0}^{\infty} x_{j,i} t^{-j}.$$

Let A be the polynomial ring $\mathbb{F}_2[t] \subset K$, and consider the sub A-module L of K^v spanned by the $(v+1)$ vectors

$$(\chi_{w-1}, \chi_{w-2}, \ldots, \chi_{w-v}), (1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \cdots, (0, \ldots, 0, 1).$$

This can be proved to be a v-dimensional free A-submodule, i.e., a lattice. We define the successive minima of L as follows. Define a non-Archimedean valuation to $x \in K$ by

$$|x| = \begin{cases} 0 & \text{if } x = 0 \\ 2^k & \text{if } x \neq 0 \text{ and } k \text{ is the largest exponent of nonzero terms.} \end{cases}$$

For each v-dimensional vector $X = (x_1, \cdots, x_v) \in K^v$, define its norm by

$$\|X\| = \max_{1 \le i \le v} |x_i|.$$

Definition 5.2. *Let $X_1, \cdots, X_v$ be points in a lattice $L \in K^v$ of rank v. We call $X_1, \cdots, X_v$ a reduced basis of L over $\mathbb{F}_2$ if the following properties hold:*

(1) X_1 is a shortest nonzero vector in L.

(2) X_i is a shortest vector among the vectors in L but outside the K-span $\langle X_1, \cdots, X_{i-1}\rangle_K$, for each $1 \le i \le v$.

The numbers $\sigma_i = \|X_i\|$ are uniquely determined by the lattice, and $s_i = \log_2 \sigma_i$ $i = 1, \cdots, v$ are called its successive minima.

THEOREM [?]. *The sequence* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i, \ldots$ *is* $(-s_v)$*-distributed to* v*-bit accuracy, where* s_v *is the* v*-th successive minimum of the lattice* L *in* K^v *associated with the sequence.*

Thus, the calculation of $k(v)$ is reduced to obtaining the successive minima. For this, there is an efficient algorithm [?]. Since the dimension of the state space is large for MT, we need several programming techniques for an efficient implementation.

We shall give only one comment: we adopted "lazy evaluation." We keep only one $(n \times w - r)$-array for describing one vector in K^v, and calculate the coefficients of t^{-k} by generating k words. Thus, here again, we depend on the easiness to generate the MT sequence.

The time-complexity of Lenstra's algorithm is $O(v^4p^2)$ (see [?][?]), which might be larger than the time-complexity $O(p^3)$ in obtaining the rank of $p \times p$ matrix for large v. However, according to our experiments, Lenstra's algorithm is much faster. This would be because (1) for the rank of matrix, we needs $p^2 \sim 4 \times 10^8$ bits of memory, and the operations on such large matrices invoke swapping between memory and disk, (2) $O(v^4p^2)$ is the worst case, and in average the order seems to be less.

5.3 Tempering

As stated in §5.1, in the case of MT, $P = 2^{nw-r} - 1$ holds, so we have $k(v) \leq \lfloor \frac{nw-r}{v} \rfloor$. Generators attaining this upper bound for every v $(1 \leq v \leq w)$ are called *asymptotically random*[?]. To attain $k(v)$ near these bounds, we multiply the output vector by a *tempering* matrix T as in [?]. We could not make the realized values meet the theoretical upper bound. We show a tighter bound (Appendix B) than $k(v) \leq \lfloor \frac{nw-r}{v} \rfloor$, but we could not attain that bound neither. In addition, we have no efficient algorithm corresponding to the one in [?] now. So, using the same backtracking technique as [?], accelerated by Tezuka's resolution-wise lattice, we searched for parameters with $k(v)$ as near to $\lfloor \frac{nw-r}{v} \rfloor$ as possible.

Let $\{\mathbf{x}_i\}$ be an MT sequence, and then define a sequence $\{\mathbf{z}_i\}$ by

$$\mathbf{z}_i := \mathbf{x}_i T,$$

where T is a regular $\mathbb{F}_2$-matrix representing the composition of the transformations (2.2), (2.3), (2.4) and (2.5) described in §2.1. Since T is regular, the period of $\{\mathbf{z}_i\}$ is the same as that of $\{\mathbf{x}_i\}$. About the peculiar form of tempering and how to search the tempering parameters, please refer to [?]. We determine the first three by the same method as [?]. Then we choose the last one so that the LSBs have a satisfactory k-distribution.

6. CONCLUSION

We proposed a pseudorandom number generator called Mersenne Twister. A portable C-code MT19937 attains a far longer period and far larger k-distributions than any previously existing generator (see Table II). The form of recurrence is cleverly selected so that both the generation and the parameter search are efficient (§3). The initialization is care-free. This generator is as fast as other common generators, such as the standard ANSI C `rand`, and it passed several statistical tests including diehard. Thus we can say that this is one of the most promising generators at the present time.

As a final remark, we stress that we used efficient algorithms unique to $\mathbb{F}_2[t]$. These algorithms enabled us to obtain better performance than integer-large-modulus generators, from the viewpoint of longer periods (Proposition 4.2) and higher k-distribution property (lattice methods in §5).

APPENDIX A.

A1 Explicit form of the matrix B

The explicit form of the $((nw-r)\times(nw-r))$-matrix B that appeared in §3.1 is as follows,

$$B = \left(\begin{array}{c|ccccc|c} \mathbf{0} & \mathbf{I}_w & \mathbf{0} & \mathbf{0} & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_w & \mathbf{0} & & & \\ \vdots & & & \ddots & & & \\ \mathbf{0} & & & & & & \\ \mathbf{I}_w & & & & & & \\ \mathbf{0} & & & & & & \\ \vdots & & & & \ddots & & \\ \mathbf{0} & & & & \mathbf{0} & \mathbf{I}_w & \mathbf{0} \\ \hline \mathbf{0} & & & & \mathbf{0} & \mathbf{0} & \mathbf{I}_{w-r} \\ \hline \mathbf{S} & & & & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right), \mathbf{S} := \begin{pmatrix} \mathbf{0} & \mathbf{I}_r \\ \mathbf{I}_{w-r} & \mathbf{0} \end{pmatrix} A.$$

For $l = 0, 1, \cdots$,

$$(\mathbf{x}_{l+n}, \mathbf{x}_{l+n-1}, \cdots, \mathbf{x}^u_{l+1}) = (\mathbf{x}_{l+n-1}, \mathbf{x}_{l+n-2}, \cdots, \mathbf{x}^u_l)B,$$

holds, where $\mathbf{x}^u_l$ denotes the $(w-r)$-dimensional column vector consisting of the most significant $(w-r)$ bits of $\mathbf{x}_l$, i.e, $\mathbf{x}^u_l = \mathbf{x}_l \begin{pmatrix} I_{w-r} \\ 0 \end{pmatrix}$.

It is not hard to see that

$$\begin{aligned} \varphi_B(t) \;=\; & (t^n + t^m)^{w-r}(t^{n-1} + t^{m-1})^r + a_0(t^n + t^m)^{w-r}(t^{n-1} + t^{m-1})^{r-1} \\ & + \cdots + a_{r-2}(t^n + t^m)^{w-r}(t^{n-1} + t^{m-1}) + a_{r-1}(t^n + t^m)^{w-r} \\ & + a_r(t^n + t^m)^{w-r-1} + \cdots + a_{w-2}(t^n + t^m) + a_{w-1}, \end{aligned}$$

where a_i's are as in §2.1.

A2 Relation between an MT sequence and its subsequences

Let $\varphi_B(t)$ be the characteristic polynomial B, D_w be a linear operator which shifts $(\cdots, \mathbf{x}_{l+2}, \mathbf{x}_{l+1}, \mathbf{x}^u_l)$ w-times to the right; i.e

$$D_w(\cdots, \mathbf{x}_{l+2}, \mathbf{x}_{l+1}, \mathbf{x}^u_l) = (\cdots, \mathbf{x}_{l+3}, \mathbf{x}_{l+2}, \mathbf{x}^u_{l+1}).$$

Then, since B is the representation matrix of D_w on the space of $(n \times w - r)$-arrays,

$$\varphi_B(D_w)(\cdots, \mathbf{x}_2, \mathbf{x}_1, \mathbf{x}^u_0) = 0 \tag{A.1}$$

holds. Put $p := nw - r$ and suppose that

$$\varphi_B(t) = t^p + b_{p-1}t^{p-1} + \cdots + b_1 t + b_0,$$

$$\mathbf{x}_l = (x_{l,w-1}, x_{l,w-2}, \cdots, x_{l,1}, x_{l,0}), \quad x_{l,i} \in \mathbb{F}_2,$$

$$\mathbf{y} = (\cdots, \mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_0^u),$$

then the equation (A.1) can be written as follows

$$\begin{array}{ccc}
b_0\mathbf{y} & = & b_0(\cdots, x_{1,1}, x_{1,0}, \cdots, x_{0,r+1}, x_{0,r}) \\
+ & & + \\
b_1 D_w \mathbf{y} & = & b_1(\cdots, x_{2,1}, x_{2,0}, \cdots, x_{1,r+1}, x_{1,r}) \\
+ & & + \\
\vdots & & \vdots \\
+ & & + \\
b_{p-1} D_w^{p-1}\mathbf{y} & = & b_{p-1}(\cdots, x_{p,1}, x_{p,0}, \cdots, x_{p-1,r+1}, x_{p-1,r}) \\
+ & & + \\
D_w^p \mathbf{y} & = & b_p(\cdots, x_{p+1,1}, x_{p+1,0}, \cdots, x_{p,r+1}, x_{p,r}) \\
\| & & \| \\
0 & & 0
\end{array}$$

Then, it is easy to see that

$$\varphi_B(D)(\cdots, x_{3,i}, x_{2,i}, x_{1,i}) = 0, \ (i = 0, 1, \cdots, w-1)$$

holds. Thus we get the following proposition.

Proposition A.1. *For an MT sequence* $(\cdots,\mathbf{x}_2,\mathbf{x}_1,\mathbf{x}_0^u)$*, its* i*th-bit subsequence* $(\cdots,x_{3,i},x_{2,i},x_{1,i})$ *satisfies the same linear recurring equation corresponding to* $\varphi_B(t)$*, independently of the choice of* i.

Corollary A.2. *An MT sequence* $(\cdots,\mathbf{x}_2,\mathbf{x}_1,\mathbf{x}_0^u)$ *attains the maximal period* $2^p - 1$ *if and only if its* i*-th bit subsequence* $(\cdots,x_{3,i},x_{2,i},x_{1,i})$ *attains the maximal period* $2^p - 1$.

APPENDIX B. OBSTRUCTIONS TO OPTIMAL DISTRIBUTION

In this appendix we shall show some obstructions for MT to achieve the upper bound on $k(v)$ obtained from the period.

Proposition B.1. *Let* $j < n$ *be an integer. If*

$$\frac{j(j+3)}{2} v > (j+1)w - r$$

holds, then the order of equidistribution $k(v)$ *to* v *bit accuracy of the MT sequence is at most* $jn - 1$.

This says that if $v > w - \frac{r}{2}$ then $k(v)$ is at most $n-1$, and that if $v > \frac{3w-r}{5}$ then $k(v)$ is at most $2n-1$. The former is much more restrictive than $k(v) \leq \lfloor \frac{nw-r}{v} \rfloor$, for large r, such as MT19937 in Table II.

Proof. k-distribution to v-bit accuracy is equivalent to the surjectivity of the linear mapping

$$(\mathbf{x}_0^u, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) \xrightarrow{f} (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k) \begin{pmatrix} TQ & & & \\ & TQ & & \\ & & \ddots & \\ & & & TQ \end{pmatrix},$$

where T denotes the $(w \times w)$-tempering matrix and Q denotes the matrix taking the upper v bits, i.e. $Q = \begin{pmatrix} I_v \\ 0 \end{pmatrix}$. Although we used the recurrence (2.1), as far as the surjectivity of this mapping is concerned, we get the same result even if we use the recurrence

$$\mathbf{x}_{k+n} := (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l) A, \quad (k = 0, 1, \cdots),$$

by the same argument as in the proof of Theorem 2.1 in [?]. So, from now on, we use this recurrence. Now, the dependencies of the $\mathbf{x}_i$'s are described by the following diagram.

$$\begin{array}{ccccccccccccc}
 & \mathbf{x}_0^u & & \mathbf{x}_1 & & \mathbf{x}_2 & & \mathbf{x}_3 & \cdots & & \mathbf{x}_{n-1} & \\
 & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \cdots & \swarrow & \downarrow & \\
\leftarrow & \mathbf{x}_n & & \mathbf{x}_{n+1} & & \mathbf{x}_{n+2} & & \mathbf{x}_{n+3} & \cdots & & \mathbf{x}_{2n-1} & \leftarrow \\
 & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \cdots & \swarrow & \downarrow & \\
\leftarrow & \mathbf{x}_{2n} & & \mathbf{x}_{2n+1} & & \mathbf{x}_{2n+2} & & \mathbf{x}_{2n+3} & \cdots & & \mathbf{x}_{3n-1} & \leftarrow \\
 & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \cdots & \swarrow & \downarrow & \\
\leftarrow & \mathbf{x}_{3n} & & \mathbf{x}_{3n+1} & & \mathbf{x}_{3n+2} & & \mathbf{x}_{3n+3} & \cdots & & \mathbf{x}_{4n-1} & \leftarrow \\
 & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \swarrow & \downarrow & \cdots & \swarrow & \downarrow & \\
 & \vdots & & \vdots & & \vdots & & \vdots & \cdots & & \vdots &
\end{array}$$

Then, for an integer $j < n$, the part $(\mathbf{x}_0^u, \mathbf{x}_1, \ldots, \mathbf{x}_j)$ of the initial values determines the left upper triangular region in the diagram

$$\begin{array}{l}
(\mathbf{x}_0^u, \mathbf{x}_1, \ldots\ldots\ldots, \mathbf{x}_j, \\
\mathbf{x}_n, \mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+j-1}, \\
\qquad\vdots \\
\mathbf{x}_{jn}).
\end{array}$$

Then, if $k(v) \geq jn$, the multiplication by TQ on each vector must be surjective as a whole. Thus the dimension of the domain $(j+1)w - r$ is at least that of the target $\frac{j(j+3)}{2} v$. This shows that if $k(v) \geq jn$ then $\frac{j(j+3)}{2} v \leq (j+1)w - r$. □

Proposition B.2. *If $k(v) \geq n + \max\{r, w-r\} - 1$, then $k(v) \geq 2(n-1)$.*

This shows that $k(v)$ tends to be near to n (at most $\max\{r, w-r\}$ distance) if it is less than $2(n-1)$. This explains partly the absence of intermediate values of $k(v)$ in Table II.

Proof. We apply the simplification of the recurrence as in the above proof. Moreover, instead of the initial vector $(\mathbf{x}_0^u, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$, we use $(\mathbf{x}_0^u, \tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{n-1})$, where $\tilde{}$ denotes the mapping $(\mathbf{x}^u | \mathbf{x}^l) \mapsto (\mathbf{x}^l | \mathbf{x}^u)$, i.e. the multiplication of $R := \begin{pmatrix} 0 & I_{w-r} \\ I_r & 0 \end{pmatrix}$. Then, when we explicitly write down the matrix which gives the upper v bits of the first $n-1+l$ values from the initial value, it becomes

$$M_l := \left.\left(\begin{array}{c|c|c|c|c|c|c|c} & & & & A_1 & & & \\ \hline A_2 & & & & A_3 & & & \\ \hline A_4 & & & & & A_1 & & \\ \hline & A_2 & & & & A_3 & & \\ \hline & A_4 & & & & & \ddots & \\ \hline & & \ddots & & & & & A_1 \\ \hline & & & A_2 & & & & A_3 \\ \hline & & & A_4 & & & & \end{array}\right)\right\} l \text{ pairs},$$

where $\begin{pmatrix} A_2 \\ A_4 \end{pmatrix}$ is the matrix RTQ (T: tempering matrix), partitioned into the first r rows and the last $w-r$ rows, and $\begin{pmatrix} A_1 \\ A_3 \end{pmatrix}$ is the matrix $RATQ$, partitioned to $w-r$ and r. Let us assume that $k(v) = n+j$ for $n+j < 2(n-1)$. This is equivalent to saying that the rank of the matrix M_l with $l = j$ is equal to its width, but that with $l = j+1$ it is not.

The former condition is equivalent to the triviality of the kernel of M_l, when applied to column vectors from the left. So we shall obtain the kernel of these matrices for $l = 1, 2, \ldots$.

Let V_1, V_2 denote the v-dimensional vector space of row vectors, and W, W' denote the row vector space of dimension $w-r$, r, respectively. We shall first consider the kernel of $\begin{pmatrix} & A_1 \\ A_2 & A_3 \end{pmatrix} : V_2 \oplus V_1 \to W \oplus W'$. The existence of A_1 implies that the V_1 component is in $A_1^{-1}(0)$, where the $^{-1}$ means the inverse image. Then, the V_1 component should be mapped by A_3 to an image of A_2, so the projection to the V_2 component of the kernel of M_1 is

$$A_2^{-1}A_3A_1^{-1}(0).$$

The sequence satisfies $k(v) = n-1$ if and only if M_2 has nontrivial kernel, i.e., the kernel of A_4 has nontrivial intersection with $A_2^{-1}A_3A_1^{-1}(0)$. If we denote by

$$\Phi := A_2^{-1}A_3A_1^{-1}A_4$$

the corresponding map from the set of subspaces of V_1 to itself, then this can be stated as $\mathrm{Ker}(A_4) \cap \Phi(0) \neq 0$.

It is not difficult to check that the kernel of M_l is nontrivial if and only if $\mathrm{Ker}(A_4) \cap \Phi^l(0) \neq 0$, so the smallest such $l \leq 2(n-1)$ gives $k(v) = n+l-2$, if it exists. Thus, to prove the proposition, it is enough to show that $\Phi^{l+1}(0) = \Phi^l(0)$ for $l \geq \max\{r, w-r\}+1$, since then the smallest l with $\mathrm{Ker}(A_4) \cap \Phi^l(0) \neq 0$ should satisfy $l \leq \max\{r, w-r\}+1$, and then $k(v) = n+l-2$ implies the proposition. To prove the above stability, we note that Φ is a monotonic function with respect to the inclusion. So,

$$0 \subset \Phi(0) \subset \Phi^2(0) \subset \cdots.$$

Now, the corresponding subspaces in W, W' are increasing, but the dimensions of W, W' are $w-r$, r, respectively. So after applying $\max\{w-r, r\}$ iterations of Φ, they will be stable. By returning to V_1, we know that one more application of Φ stabilizes the space. $\square$

APPENDIX C. C PROGRAM

Here is a C-code for MT19937. This code works both in 32-bit and 64-bit machines. The function `genrand()` returns a uniformly distributed real pseudorandom number (of type `double`, with 32-bit precision) in the closed interval $[0, 1]$. The function `sgenrand()` sets initial values to the array `mt[N]`. Before using `genrand()`, `sgenrand()` must be called with a non-zero unsigned long integer as a seed.

The generator can be modified to a 32-bit unsigned long integer generator by changing two lines, namely, the type of the function `genrand()` and the output-scheme. See the comment inside.

The magic numbers are put in the macros, so that one can easily change them according to Table II. Essentially the same code is downloadable from the http-site in Salzburg University (see `http://random.mat.sbg.ac.at/news/`).

Topher Cooper kindly enhanced the robustness in the initialization scheme. Marc Rieffel (`marc@scp.syr.edu`), who uses MT19937 in a plasma simulation, reported that by replacing the function calls by the macros, the runtime could be reduced by 37%. His code is available from `ftp.scp.syr.edu/pub/hawk/mt19937b-macro.c`, which also improves several other points.

```
/* A C-program for MT19937: Real number version              */
/* genrand() generates one pseudorandom real number (double)   */
/* which is uniformly distributed on [0,1]-interval, for each  */
/* call. sgenrand(seed) set initial values to the working area */
/* of 624 words. Before genrand(), sgenrand(seed) must be      */
/* called once. (seed is any 32-bit integer except for 0).     */
/* Integer generator is obtained by modifying two lines.       */
/*   Coded by Takuji Nishimura, considering the suggestions by */
/* Topher Cooper and Marc Rieffel in July-Aug. 1997. Comments  */
/* should be addressed to: matumoto@math.keio.ac.jp            */

#include<stdio.h>

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0df   /* constant vector a */
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */

/* Tempering parameters */
#define TEMPERING_MASK_B 0x9d2c5680
#define TEMPERING_MASK_C 0xefc60000
#define TEMPERING_SHIFT_U(y)  (y >> 11)
#define TEMPERING_SHIFT_S(y)  (y << 7)
#define TEMPERING_SHIFT_T(y)  (y << 15)
```

```
#define TEMPERING_SHIFT_L(y)  (y >> 18)

static unsigned long mt[N]; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* initializing the array with a NONZERO seed */
void
sgenrand(unsigned long seed)
{
    /* setting initial seeds to mt[N] using         */
    /* the generator Line 25 of Table 1 in          */
    /* [KNUTH 1981, The Art of Computer Programming */
    /*    Vol. 2 (2nd Ed.), pp102]                  */
    mt[0]= seed & 0xffffffff;
    for (mti=1; mti<N; mti++)
        mt[mti] = (69069 * mt[mti-1]) & 0xffffffff;
}

double /* generating reals */
/* unsigned long */ /* for integer generation */
genrand()
{
    unsigned long y;
    static unsigned long mag01[2]={0x0, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if sgenrand() has not been called, */
            sgenrand(4357); /* a default initial seed is used   */

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

        mti = 0;
    }

    y = mt[mti++];
```

```
    y ^= TEMPERING_SHIFT_U(y);
    y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
    y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
    y ^= TEMPERING_SHIFT_L(y);

    return ( (double)y / (unsigned long)0xffffffff ); /* reals */
    /* return y; */ /* for integer generation */
}

/* this main() outputs first 1000 generated numbers  */
main()
{
    int j;

    sgenrand(4357); /* any nonzero integer can be used as a seed */
    for (j=0; j<1000; j++) {
        printf("%5f ", genrand());
        if (j%8==7) printf("\n");
    }
    printf("\n");
}
```

ACKNOWLEDGMENTS

The idea of TGFSR comes from a discussion with N. Yoneda, who passed away in Apr. 1996, held at a pub KURUMAYA near University of Tokyo. This paper is dedicated to his memory. A part of this work was done during the first author's visit to CRM, Montreal University. He is thankful to P. L'Ecuyer and R. Couture for their hospitality and invaluable discussions. The discussions held at the conference there were very helpful. In particular, A. Compagner, P. Hellekalek and H. Leeb gave the first author a number of valuable comments. H. Leeb kindly included MT19937 in their http-site in Salzburg University (see `http://random.mat.sbg.ac.at/news/`). We are thankful to H. Niederreiter for the information on his general and detailed framework of the multiple-recursive matrix method, and to M. Fushimi for the comment on the auto-correlations.

We are deeply grateful to D. E. Knuth for the very useful discussion when he visited Keio University in Nov. 1996, and his subsequent correspondence including a theoretical justification of the recurrence (2.1) in §3.1.

We are thankful to G. Marsaglia and B. Narasimhan, for helping us in executing their diehard test.

Some part of this study was done during the first author's stay in Max-Planck-Institut für Mathematik, BONN, and also during his visit to Salzburg University. He is thankful to the hospitality there. In particular, the discussions with the pLab group at Salzburg greatly helped to improve this paper. Wegenkittl kindly tested MT19937. We thank to Topher Cooper and Marc Rieffel for their helpful suggestions about the C-code implementation (see comments in Appendix C).

We are deeply thankful to P.L'Ecuyer for his kind and careful comments on the whole manuscript. Thanks are also due to the anonymous referees for many constructive comments.

While we were revising this paper, we knew that A. Compagner left us. We hope that he rests in peace.

The first author would like to show his last gratitude to J, on May 28th. The second author would like to show his gratitude to his parents, who kindly recommended him to continue to study in the university.