A

Implementing 64-bit Maximally Equidistributed $\mathbb{F}_2$-Linear Generators with Mersenne Prime Period

SHIN HARASE, Ritsumeikan University
TAKAMITSU KIMOTO, Recruit Holdings Co., Ltd.

CPUs and operating systems are moving from 32 to 64 bits, and hence it is important to have good pseudorandom number generators designed to fully exploit these word lengths. However, existing 64-bit very long period generators based on linear recurrences modulo 2 are not completely optimized in terms of the equidistribution properties. Here we develop 64-bit maximally equidistributed pseudorandom number generators that are optimal in this respect and have speeds equivalent to 64-bit Mersenne Twisters. We provide a table of specific parameters with period lengths from $2^{607}-1$ to $2^{44497}-1$.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis; I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Random number generator, Mersenne Twister, Equidistribution, Empirical statistical testing

ACM Reference Format:
Shin Harase and Takamitsu Kimoto. 2017. Implementing 64-bit maximally equidistributed $\mathbb{F}_2$-linear generators with Mersenne prime period. *ACM Trans. Math. Softw.* V, N, Article A (January YYYY), 11 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Monte Carlo simulations are a basic tool in financial engineering, computational physics, statistics, and other fields. To obtain precise simulation results, the quality of pseudorandom number generators is important. At present, the 32-bit Mersenne Twister (MT) generator MT19937 (with period $2^{19937}-1$) [Matsumoto and Nishimura 1998] is one of the most widely used pseudorandom number generators. However, modern CPUs and operating systems are moving from 32 to 64 bits, and hence it is important to have high-quality generators designed to fully exploit 64-bit words.

Many pseudorandom number generators, including Mersenne Twisters, are based on linear recurrences modulo 2; these are called $\mathbb{F}_2$-*linear generators*. One advantage of these generators is that they can be assessed by means of the *dimension of equidistribution with v-bit accuracy*, which is a most informative criterion for high dimensional uniformity of the output sequences. In fact, MT19937 is not completely optimized in this respect. Panneton et al. [2006] developed the Well Equidistributed Long-period

---

This work was partially supported by the Grants-in-Aid for JSPS Fellows 24·7985, for Young Scientists (B) #26730015, for Scientific Research (B) #26310211, and for Challenging Exploratory Research #15K13460 from the Japan Society for the Promotion of Scientific Research. This work was also supported by JST CREST. Authors' addresses: S. Harase, College of Science and Engineering, Ritsumeikan University, 1-1-1 Nojihigashi, Kusatsu, Shiga, 525-8577, Japan; e-mail: harase@fc.ritsumei.ac.jp; T. Kimoto, Recruit Holdings Co., Ltd., 1-9-2 Marunouchi, Chiyoda-ku, Tokyo, 100-6640, Japan; tkimoto@r.recruit.co.jp.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Linear (WELL) generators with periods from $2^{512}-1$ to $2^{44497}-1$, which are completely optimized for this criterion (called *maximally equidistributed*), but the parameter sets were only searched for the case of 32-bit generators. Conversely, there exist several 64-bit $\mathbb{F}_2$-linear generators. Nishimura [2000] developed 64-bit Mersenne Twisters, and the SIMD-oriented Fast Mersenne Twister (SFMT) generator [Saito and Matsumoto 2009] has a function to generate 64-bit unsigned integers. For graphics processing units, Mersenne Twister for Graphic Processors (MTGP) [Saito and Matsumoto 2013] is also a good candidate. However, these generators are not maximally equidistributed. In earlier work, L'Ecuyer [1999] searched for 64-bit maximally equidistributed combined Tausworthe generators (with some additional properties). At present, though, to the best of our knowledge, there exists no 64-bit maximally equidistributed MT-type $\mathbb{F}_2$-linear generator with period $2^{19937}-1$, such as a 64-bit variant of the WELL generators.

The aim of this article is to develop 64-bit maximally equidistributed $\mathbb{F}_2$-linear generators with similar speed as the 64-bit Mersenne Twisters [Nishimura 2000]. The key techniques are (i) state transitions with double feedbacks [Panneton et al. 2006; Saito and Matsumoto 2009] and (ii) linear output transformations with several memory references [Harase 2009]. We provide a table of specific parameters with periods from $2^{607}-1$ to $2^{44497}-1$. The design of our generators is based on a combination of existing techniques, such as the WELL and dSFMT generators [Panneton et al. 2006; Saito and Matsumoto 2009], but we select state transitions differently from those of the original WELL generators to maintain the generation speed. We refer to these generators as *64-bit Maximally Equidistributed* $\mathbb{F}_2$*-Linear Generators (MELGs) with Mersenne prime period*.

In practice, we often convert unsigned integers into 53-bit double-precision floating-point numbers in $[0,1)$ in IEEE 754 format. Our 64-bit generators are useful for this. To generate 64-bit output values, one can either use a pseudorandom number generator whose linear recurrence is implemented with 32-bit integers and then take two successive 32-bit blocks or, instead, use a pseudorandom number generator whose recurrence is implemented directly over 64-bit integers. As described below, the former method may degrade the dimension of equidistribution with v-bit accuracy, compared with simply using 32-bit output values. We consider the case of the 32-bit MT19937 generator in Section 4. For this reason, we develop 64-bit MELGs to directly generate 64-bit unsigned integers.

The article is organized as follows. In the next section, we review $\mathbb{F}_2$-linear generators and their theoretical criteria. We also summarize the framework of Mersenne Twisters. Section 3 is devoted to our main result: 64-bit MELGs. In Section 4, we compare our generators with others in terms of speeds, theoretical criteria, and empirical statistical tests. Section 5 concludes.

2. PRELIMINARIES

2.1. $\mathbb{F}_2$-Linear Generators

We recall the notation of $\mathbb{F}_2$-linear generators; see [L'Ecuyer and Panneton 2009; Matsumoto et al. 2006] for details. Let $\mathbb{F}_2 := \{0,1\}$ be the two-element field, i.e., addition and multiplication are performed modulo 2. We consider the following class of generators.

Definition 2.1 ($\mathbb{F}_2$*-linear generator*). Let $S := \mathbb{F}_2^p$ be a p-dimensional state space (of the possible states of the memory assigned for generators). Let $f : S \to S$ be an $\mathbb{F}_2$-linear state transition function. Let $O := \mathbb{F}_2^w$ be the set of outputs, where w is the word size of the intended machine, and let $o : S \to O$ an $\mathbb{F}_2$-linear output function. For an initial state $s_0 \in S$, at every time step, the state is changed by the recursion

$$s_{i+1} = f(s_i) \quad (i = 0, 1, 2, \ldots), \tag{1}$$

and the output sequence is given by

$$o(s_0), o(s_1), o(s_2), \ldots \in O. \tag{2}$$

We identify O as a set of unsigned w-bit binary integers. A generator with these properties is called an *$\mathbb{F}_2$-linear generator*.

Let $P(z)$ be the characteristic polynomial of f. The recurrence (1) has a period of length $2^p - 1$ (its maximal possible value) if and only if $P(z)$ is a primitive polynomial modulo 2 [Niederreiter 1992; Knuth 1997]. When this value is reached, we say that the $\mathbb{F}_2$-linear generator has *maximal period*. Unless otherwise noted, we assume throughout that this condition holds.

2.2. Quality Criteria

Following [L'Ecuyer and Panneton 2009; Matsumoto et al. 2006], we recall two quality criteria for $\mathbb{F}_2$-linear generators. A most informative criterion for high dimensional uniformity is the *dimension of equidistribution with v-bit accuracy*. Assume that an $\mathbb{F}_2$-linear generator has the maximal period $2^p - 1$. We identify the output set $O := \mathbb{F}_2^w$ as a set of unsigned w-bit binary integers. We focus on the v most significant bits of the output, and regard these bits as the *output with v-bit accuracy*. This amounts to considering the composition $o_v : S \xrightarrow{o} \mathbb{F}_2^w \to \mathbb{F}_2^v$, where the latter mapping denotes taking the v most significant bits. We define the k-tuple output function as

$$o_v^{(k)} : S \to (\mathbb{F}_2^v)^k, \quad s_0 \mapsto (o_v(s_0), o_v(f(s_0)), \ldots, o_v(f^{k-1}(s_0))).$$

Thus, $o_v^{(k)}(s_0)$ is the vector formed by the v most significant bits of k consecutive output values of the pseudorandom number generators from a state s_0.

Definition 2.2 (*Dimension of equidistribution with v-bit accuracy*). The generator is said to be k-dimensionally equidistributed with v-bit accuracy if and only if $o_v^{(k)} : S \to (\mathbb{F}_2^v)^k$ is surjective. The largest value of k with this property is called the *dimension of equidistribution with v-bit accuracy*, denoted by $k(v)$.

Because $o_v^{(k)}$ is $\mathbb{F}_2$-linear, k-dimensional equidistribution with v-bit accuracy means that every element in $(\mathbb{F}_2^v)^k$ occurs with the same probability, when the initial state s_0 is uniformly distributed over the state space S. As a criterion of uniformity, larger values of $k(v)$ for each $1 \leq v \leq w$ are desirable [Tootill et al. 1973]. We have a trivial upper bound $k(v) \leq \lfloor p/v \rfloor$. The gap $d(v) := \lfloor p/v \rfloor - k(v)$ is called the *dimension defect at v*, and the sum of the gaps $\Delta := \sum_{v=1}^{w} d(v)$ is called the *total dimension defect*. If $\Delta = 0$, the generator is said to be *maximally equidistributed*. For $\mathbb{F}_2$-linear generators, one can quickly compute $k(v)$ for $v = 1, \ldots, w$ by using lattice reduction algorithms over formal power series fields [Harase et al. 2011; Harase 2011]. These are closely related to the lattice reduction algorithm originally proposed by Couture and L'Ecuyer [2000].

As another criterion, the number N_1 of nonzero coefficients for $P(z)$ should be close to $p/2$ [Compagner 1991; Wang and Compagner 1993]. For example, generators for which $P(z)$ is a trinomial or pentanomial fail in statistical tests [Lindholm 1968; Matsumoto and Kurita 1996; Matsumoto and Nishimura 2002]. If N_1 is not large enough, the generator suffers a long-lasting impact for poor initialization known as a 0-excess state $s_0 \in S$, which contains only a few bits set to 1 [Panneton et al. 2006]. Thus, N_1 should be in the vicinity of $p/2$.

2.3. Mersenne Twisters

In general, to ensure a maximal period, testing the primitivity of $P(z)$ is the bottleneck in searching for long-period generators (because the factorization of $2^p - 1$ is required). If p is a Mersenne exponent (i.e., $2^p - 1$ is a Mersenne prime), one can instead use an irreducibility test that is easier and equivalent. Matsumoto and Nishimura [1998] developed a pseudorandom number generator with a Mersenne prime period $2^p - 1$ known as the Mersenne Twister. We briefly review their method. The state space S and state transition $f : S \to S$ are expressed as

$$(\overline{\mathbf{w}}_i^{w-r}, \mathbf{w}_{i+1}, \mathbf{w}_{i+2}, \ldots, \mathbf{w}_{i+N-1}) \mapsto (\overline{\mathbf{w}}_{i+1}^{w-r}, \mathbf{w}_{i+2}, \mathbf{w}_{i+3}, \ldots, \mathbf{w}_{i+N}), \tag{3}$$

where $\mathbf{w}_i \in \mathbb{F}^w$ is a w-bit word vector, $\overline{\mathbf{w}}_i^{w-r} \in \mathbb{F}_2^{w-r}$ denotes the $w - r$ most significant bits of $\mathbf{w}_i$, $N := \lceil p/w \rceil$, and r is a non-negative integer such that $p = Nw - r$, so that the p bits of S are stored in an array of Nw bits in which there are r unused bits. The state transition (3) is implemented as

$$\mathbf{w}_{i+N} := \mathbf{w}_{i+M} \oplus (\overline{\mathbf{w}}_i^{w-r} \mid \underline{\mathbf{w}}_{i+1}^{r})A, \tag{4}$$

where $\underline{\mathbf{w}}_{i+1}^{r} \in \mathbb{F}_2^r$ represents the r least significant bits of $\mathbf{w}_{i+1}$, $\oplus$ is the bitwise exclusive-OR (i.e., addition in $\mathbb{F}_2^w$), and $(\overline{\mathbf{w}}_i^{w-r} \mid \underline{\mathbf{w}}_{i+1}^{r})$, a w-bit vector, is the concatenation of the $(w - r)$-bit vector $\overline{\mathbf{w}}_i^{w-r}$ and the r-bit vector $\underline{\mathbf{w}}_{i+1}^{r}$ in that order. M is an integer such that $0 < M < N - 1$, and $A \in \mathbb{F}_2^{w \times w}$ is a $(w \times w)$-regular matrix (with the format in (9) below). Furthermore, to improve $k(v)$, for the right-hand side of (3), the output transformation $o : S \to O$ is implemented as

$$(\overline{\mathbf{w}}_{i+1}^{w-r}, \mathbf{w}_{i+2}, \ldots, \mathbf{w}_{i+N}) \in S \mapsto \mathbf{w}_{i+N}T \in O, \tag{5}$$

where $T \in \mathbb{F}_2^{w \times w}$ is a suitable $(w \times w)$-regular matrix. This technique is called *tempering*. From this, we obtain an output sequence $\mathbf{w}_N T, \mathbf{w}_{N+1}T, \mathbf{w}_{N+2}T, \ldots \in O$ by multiplying a matrix T by the sequence from (4). Matsumoto and Nishimura [1998] and Nishimura [2000] searched for 32- and 64-bit Mersenne Twisters with period length $2^{19937} - 1$, respectively. In Appendix B of [Matsumoto and Nishimura 1998], it is proved that these generators cannot attain the maximal equidistribution (e.g., $\Delta = 6750$ for MT19937 in [Matsumoto and Nishimura 1998]). In fact, the state transition in (3)–(4) is very simple, but the linear output transformation T in (5) is rather complicated (see [Matsumoto and Nishimura 1998; Nishimura 2000] for details).

3. MAIN RESULT: 64-BIT MAXIMALLY EQUIDISTRIBUTED $\mathbb{F}_2$-LINEAR GENERATORS

3.1. Design

To obtain maximally equidistributed generators without loss of speed, we try to shift the balance of costs in f and o. The key technique is a suitable choice of (i) state transitions with double feedbacks proposed in [Panneton et al. 2006; Saito and Matsumoto 2009] and (ii) linear output transformations with several memory references from [Harase 2009]. Let $N = \lceil p/w \rceil$. We divide $S = \mathbb{F}_2^p$ into two parts $S = \mathbb{F}_2^{p-w} \times \mathbb{F}_2^w$ and consider the state transition $f : S \to S$ with

$$(\overline{\mathbf{w}}_i^{w-r}, \mathbf{w}_{i+1}, \mathbf{w}_{i+2}, \ldots, \mathbf{w}_{i+N-2}, \mathbf{v}_i) \mapsto (\overline{\mathbf{w}}_{i+1}^{w-r}, \mathbf{w}_{i+2}, \mathbf{w}_{i+3}, \ldots, \mathbf{w}_{i+N-1}, \mathbf{v}_{i+1}), \tag{6}$$

where $\mathbf{w}_{i+N-1}$ and $\mathbf{v}_{i+1}$ are determined by the recursions (7) and (8), as described below. The first $p - w$ bits are stored in an array in which r bits are unused, as for the original MTs. Note that the number of words is $N - 1$, not N. The remaining word $\mathbf{v}_i$ is expected to be stored in a register of the CPU and updated as $\mathbf{v}_{i+1}$ at the next step, so that the implementation requires only a single word (see Figure 1 and Algorithm 1 in this subsection). The use of the extra state variable $\mathbf{v}_i$ was originally proposed by

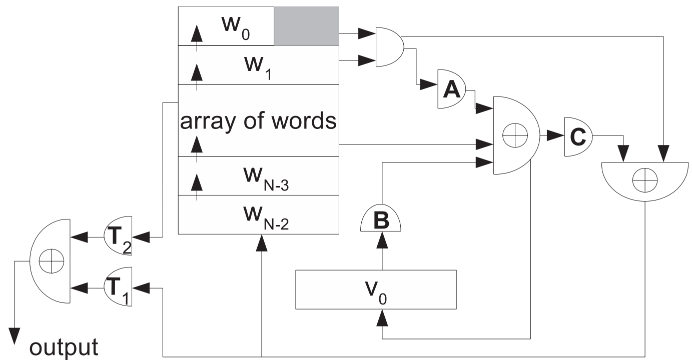

Fig. 1. Circuit-like description of the proposed generators.

Panneton et al. [2006] and refined by Saito and Matsumoto [2009]. (Note that $\mathbf{v}_{i,0}$ in Fig. 1 of [Panneton et al. 2006] corresponds to this variable $\mathbf{v}_i$.) This approach is a key technique for drastically improving N_1 and Δ. By refining the recursion formulas proposed in [Saito and Matsumoto 2009], we implement the state transition f with the following recursions:

$$\mathbf{v}_{i+1} := (\overline{\mathbf{w}}_i^{w-r} \mid \underline{\mathbf{w}}_{i+1}^{r})A \oplus \mathbf{w}_{i+M} \oplus \mathbf{v}_i B, \quad (7)$$

$$\mathbf{w}_{i+N-1} := (\overline{\mathbf{w}}_i^{w-r} \mid \underline{\mathbf{w}}_{i+1}^{r}) \oplus \mathbf{v}_{i+1} C. \quad (8)$$

A, B, and C are $(w \times w)$-matrices defined indirectly as follows:

$$\mathbf{w}A := \begin{cases} (\mathbf{w} \gg 1) & \text{if } w_{w-1} = 0, \\ (\mathbf{w} \gg 1) \oplus \mathbf{a} & \text{if } w_{w-1} = 1, \end{cases} \quad (9)$$

$$\mathbf{w}B := \mathbf{w} \oplus (\mathbf{w} \ll \sigma_1), \quad (10)$$

$$\mathbf{w}C := \mathbf{w} \oplus (\mathbf{w} \gg \sigma_2), \quad (11)$$

where $\mathbf{w} = (w_0, \ldots, w_{w-1}) \in \mathbb{F}_2^w$ and $\mathbf{a} \in \mathbb{F}_2^w$ are w-bit vectors, σ_1 and σ_2 are integers with $0 < \sigma_1, \sigma_2 < w$, and "$\mathbf{w} \ll l$" and "$\mathbf{w} \gg l$" denote left and right logical (i.e., zero-padded) shifts by l bits, respectively. Note that $\mathbf{v}_i$ is one component in a state $s_i \in S$ and is not the output.

To attain the maximal equidistribution exactly, we design a linear output transformation using another word in the state array, which comes from [Harase 2009]. More precisely, for the right-hand side of (6), we consider the following linear output transformation $o : S \to O$ with one more memory reference:

$$(\overline{\mathbf{w}}_{i+1}^{r-w}, \mathbf{w}_{i+2}, \ldots, \mathbf{w}_{i+N-1}, \mathbf{v}_{i+1}) \in S \mapsto \mathbf{w}_{i+N-1}T_1 \oplus \mathbf{w}_{i+L}T_2 \in O. \quad (12)$$

Here T_1 and T_2 are $(w \times w)$-matrices defined by

$$\mathbf{w}T_1 := \mathbf{w} \oplus (\mathbf{w} \ll \sigma_3), \quad (13)$$

$$\mathbf{w}T_2 := (\mathbf{w} \,\&\, \mathbf{b}), \quad (14)$$

where L is an integer with $0 < L < N-2$, σ_3 is an integer with $0 < \sigma_3 < w$, $\&$ denotes bitwise AND, and $\mathbf{b} \in \mathbb{F}_2^w$ is a w-bit vector. A circuit-like description of the proposed generators is shown in Figure 1.

An equivalent formal algorithm can be described as Algorithm 1, in which, instead of shifting, we use a round-robin technique (i.e., a pointer technique) to improve the efficiency of the generation. Let $\mathbf{w}[0..N-2]$ be an array of $N-1$ unsigned integers with w bits, and $\mathbf{v}$ be a w-bit unsigned integer that corresponds to the extra state variable $\mathbf{v}_i$.

Let $\mathbf{x}$ be a temporary variable and $\mathbf{y}$ be an output variable that are w-bit unsigned integers, respectively. Set $\mathbf{m}^{w-r} \leftarrow (\underbrace{1,\ldots,1}_{w-r},\underbrace{0,\ldots,0}_{r})$ and $\hat{\mathbf{m}}^r \leftarrow (\underbrace{0,\ldots,0}_{w-r},\underbrace{1,\ldots,1}_{r})$. Here $\mathbf{m}^{w-r}$ is a bit mask that retains the first $w-r$ bits and sets the other r bits to zero, whereas $\hat{\mathbf{m}}^r$ is its bitwise complement. Before Algorithm 1 begins, we initialize $(\mathbf{w}[0] \;\&\; \mathbf{m}^{w-r}), \mathbf{w}[1], \ldots, \mathbf{w}[N-2], \mathbf{v} \leftarrow$ initial values, not all zero, and set the pointer $i \leftarrow 0$. For more detailed descriptions of the initialization, see Remark 3.1.

ALGORITHM 1: The algorithm of the proposed generators

Input : $\mathbf{w}[0], \mathbf{w}[1], \ldots, \mathbf{w}[N-2], \mathbf{v}$, and the pointer i
Output: w-bit output $\mathbf{y}$
Set $\mathbf{x} \leftarrow (\mathbf{w}[i] \;\&\; \mathbf{m}^{w-r}) \oplus (\mathbf{w}[(i+1) \mod (N-1)] \;\&\; \hat{\mathbf{m}}^r)$. // compute $(\overline{\mathbf{w}}_i^{w-r} \mid \underline{\mathbf{w}}_{i+1}^r)$.
Set $\mathbf{v} \leftarrow \mathbf{x}A \oplus \mathbf{w}[(i+M) \mod (N-1)] \oplus \mathbf{v}B$. // compute Eq. (7).
Set $\mathbf{w}[i] \leftarrow \mathbf{x} \oplus \mathbf{v}C$. // compute Eq. (8).
Set $\mathbf{y} \leftarrow \mathbf{w}[i]T_1 \oplus \mathbf{w}[(i+L) \mod (N-1)]T_2$. // compute Eq. (12).
Increment $i \leftarrow i+1$. If $i \geq N-1$, then $i \leftarrow i \mod (N-1)$. // increment the pointer i.
Return $\mathbf{y}$

3.2. Specific Parameters

We search for specific parameters in the following way. First, we look for M, σ_1, σ_2, and $\mathbf{a} \in \mathbb{F}_2^w$ in (7)–(11) at random such that f attains the maximal period $2^p - 1$. In general, because we can obtain several parameters, we choose a parameter set whose N_1 is large enough and whose output has large $k(v)$ for $v = 1, \ldots, w$ as far as possible in the case where we set $\mathbf{y} \leftarrow \mathbf{w}[i]$ in Algorithm 1 (i.e., T_1 and T_2 are the identity and zero matrices, respectively). In the next step, we search for L, σ_3, and $\mathbf{b} \in \mathbb{F}_2^w$ in (12)–(14) at random such that the generator is "almost" maximally equidistributed (i.e., Δ is almost 0). Finally, to obtain $\Delta = 0$ strictly, we apply a slight modification to the bit mask $\mathbf{b} \in \mathbb{F}_2^w$ by using the backtracking algorithm in [Harase 2009] (with some trial and error). Table I lists specific parameters for 64-bit maximally equidistributed generators with periods ranging from $2^{607} - 1$ to $2^{44497} - 1$. We attach the acronym 64-bit MELGs, for 64-bit "maximally equidistributed $\mathbb{F}_2$-linear generators" with Mersenne prime period, to the proposed generators. The code in C is available at https://github.com/sharase/melg-64.

In parallel computing, an important requirement is the availability of pseudorandom number generators with disjoint streams. These are usually implemented by partitioning the output sequences of a long-period generator into long disjoint subsequences whose starting points are found by making large jumps in the original sequences. For this purpose, Haramoto et al. [2008] proposed a fast jumping-ahead algorithm for $\mathbb{F}_2$-linear generators. We also implemented this algorithm for our 64-bit MELGs. The code is also available at the above website. The default skip size is 2^{256}.

For our generators, we implemented a function to produce double-precision floating-point numbers $u_0, u_1, u_2, \ldots$ in $[0, 1)$ in IEEE 754 format by using the method of Section 2 of [Saito and Matsumoto 2009] (i.e., the 12 bits for sign and exponent are kept constant, and the 52 bits of the significand are taken from the generator output). We note that this method is preferable from the viewpoint of $k(v)$ because it does not introduce approximation errors by division.

Remark 3.1. Matsumoto et al. [2007] reported that many pseudorandom number generators have some nonrandom bit patterns when initial seeds are systematically chosen, especially when the initialization scheme is based on a linear congruential

Table I. Specific Parameters of 64-bit Maximally Equidistributed $\mathbb{F}_2$-Linear Generators (MELGs)

	M / L	σ_1 / σ_3	σ_2	$\mathbf{a}$ / $\mathbf{b}$	N_1 / Δ
$p = 607, w = 64, r = 33, N = 10$					
MELG607-64	5	13	35	81f1fd68012348bc	313
	3	30		66edc62a6bf8c826	0
$p = 1279, w = 64, r = 1, N = 20$					
MELG1279-64	7	22	37	1afefd1526d3952b	641
	5	6		3a23d78e8fb5e349	0
$p = 2281, w = 64, r = 23, N = 36$					
MELG2281-64	17	36	21	7cbe23ebca8a6d36	1145
	6	6		e4e2242b6e15aebe	0
$p = 4253, w = 64, r = 35, N = 67$					
MELG4253-64	29	30	20	fac1e8c56471d722	2129
	9	5		cb67b0c18fe14f4d	0
$p = 11213, w = 64, r = 51, N = 176$					
MELG11213-64	45	33	13	ddbcd6e525e1c757	5455
	4	5		bd2d1251e589593f	0
$p = 19937, w = 64, r = 31, N = 312$					
MELG19937-64	81	23	33	5c32e06df730fc42	9603
	19	16		6aede6fd97b338ec	0
$p = 44497, w = 64, r = 47, N = 696$					
MELG44497-64	373	37	14	4fa9ca36f293c9a9	19475
	95	6		6fbbee29aaefd91	0

generator. To avoid such phenomena, the 2002 version of MT19937 adopted a nonlinear initializer described in Eq. (30) of [Matsumoto et al. 2007]. For our proposed generators, we implemented a similar initialization scheme. Let $\mathbf{w}_0 \in \mathbb{F}_2^w$ be an initial seed. To obtain an initial state $s_0 = (\overline{\mathbf{w}}_0^{w-r}, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{N-2}, \mathbf{v}_0) \in S$, we set $\overline{\mathbf{w}}_0^{w-r}$ as the $w - r$ most significant bits of $\mathbf{w}_0$ and compute

$$\mathbf{w}_i \leftarrow a \times (\mathbf{w}_{i-1} \oplus (\mathbf{w}_{i-1} \gg \sigma_4)) + i \pmod{2^w}$$

for $i = 1, \dots, N-2$ and

$$\mathbf{v}_0 \leftarrow a \times (\mathbf{w}_{N-2} \oplus (\mathbf{w}_{N-2} \gg \sigma_4)) + N - 1 \pmod{2^w},$$

using the usual integer arithmetic $+, -$, and $\times$, where $a = 6364136223846793005$ (in decimal notation) is a multiplier recommended in [Knuth 1997, pp. 104], $\sigma_4 = 62$, and $w = 64$. These parameters are from the 2004 version of 64-bit MT [Nishimura 2000] mentioned in Section 4. In a similar manner, we implemented an initializer whose seed is an array of integers of arbitrary length.

Remark 3.2. As pointed out in Section 3.8 of [Saito and Matsumoto 2013], it might be difficult to run the proposed generators in a multi-threaded environment, such as GPUs. This is because our generators have the heavy dependencies on the partial computation of the extra state variable $\mathbf{v}_i$ in the recursion (7). Thus, in this case, it seems that the original MTs [Matsumoto and Nishimura 1998; Nishimura 2000] are suitable because of the simplicity of recursion. We note that MTGP is designed specifically for this purpose (see [Saito and Matsumoto 2013] for details).

4. PERFORMANCE

We compare the following $\mathbb{F}_2$-linear generators corresponding to 64-bit integer output sequences:

— MELG19937-64: the 64-bit integer output of our proposed generator;

Table II. Figures of Merit N_1, Δ, and CPU Time (in Seconds) Taken to Generate 10^9 64-bit Unsigned Integers

Generators	N_1	Δ	CPU time (Intel)	CPU time (AMD)
MELG19937-64	9603	0	4.2123	6.2920
MT19937-64	285	7820	5.1002	6.6490
MT19937-64 (ID3)	5795	7940	4.8993	6.7930
SFMT19937-64 (without SIMD)	6711	14095	4.2654	5.6123
SFMT19937-64 (with SIMD)	6711	14095	1.8457	2.8806

— MT19937-64: the 64-bit integer output of the 64-bit Mersenne Twister (downloaded from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt64.html);
— MT19937-64 (ID3): the 64-bit integer output of a 64-bit Mersenne Twister based on a five-term recursion (ID3) [Nishimura 2000];
— SFMT19937-64 (without SIMD): the 64-bit integer output of the SIMD-oriented Fast Mersenne Twister SFMT19937 without SIMD [Saito and Matsumoto 2008];
— SFMT19937-64 (with SIMD): the 64-bit integer output of the foregoing with SIMD [Saito and Matsumoto 2008].

The first three generators have period length $2^{19937}-1$. SFMT19937-64 has the period of a multiple of $2^{19937}-1$ (see Proposition 1 in [Saito and Matsumoto 2008] for details). Table II summarizes the figures of merit N_1, Δ, and timings. In this table, we report the CPU time (in seconds) taken to generate 10^9 64-bit unsigned integers for each generator. The timings were obtained using two 64-bit CPUs: (i) a 3.40 GHz Intel Core i7-3770 and (ii) a 2.70 GHz Phenom II X6 1045T. The code was written in C and compiled with GCC using the -O3 optimization flag on 64-bit Linux operating systems. For SFMT19937-64, we measured the CPU time for the case of sequential generation (see [Saito and Matsumoto 2008] for details).

MELG19937-64 is maximally equidistributed and also has $N_1 \approx p/2$. These values are the best in this table. In terms of generation speed, MELG19937-64 is comparable to or even slightly faster than the MT19937-64 generators on the above two platforms. SFMT19937-64 without SIMD is comparable to or faster than MELG19937-64, and SFMT19937-64 with SIMD is more than twice as fast as MELG19937-64. However, Δ for SFMT19937-64 is rather large. In fact, the SFMT generators are optimized under the assumption that one will mainly be using 32-bit output sequences, so that the dimensions of equidistribution with v-bit accuracy for 64-bit output sequences are worse than those for 32-bit cases ($\Delta = 4188$). For this, we analyze the structure of SFMT19937 in the online Appendix.

Finally, we convert 64-bit integers into double-precision floating-point numbers $u_0, u_1, u_2, \ldots$ in $[0, 1)$, and submit them to the statistical tests included in the SmallCrush, Crush, and BigCrush batteries of TestU01 [L'Ecuyer and Simard 2007]. Note that these batteries have 32-bit resolution and have not yet been tailored to 64-bit integers. In our C-implementation, for MELG19937-64 and MT19937-64, we generate the above uniform real numbers by `(x >> 11) * (1.0/9007199254740992.0)`, where `x` is a 64-bit unsigned integer output. For SFMT19937-64, we use a function `sfmt_genrand_res53()`, which is obtained by dividing 64-bit unsigned integers `x` by 2^{64}, i.e., `x * (1.0/18446744073709551616.0)`; see the online Appendix for details. In any case, we investigate the 32 most significant bits of 64-bit outputs in TestU01. The generators in Table I passed all the tests; except for the linear complexity tests (unconditional failure) and matrix-rank tests (failure only for small p), which measure the $\mathbb{F}_2$-linear dependency of the outputs and reject $\mathbb{F}_2$-linear generators. This is a limitation of $\mathbb{F}_2$-linear generators. However, for some matrix-rank tests, we can observe differences between MELGs and SFMTs. Table III summarizes the p-values on the matrix-rank test of No. 60 of Crush for five initial states. (SFMT1279-64, SFMT2281-64, and

Table III. p-Values on the Matrix-Rank Test of No. 60 of Crush in TestU01

	1st	2nd	3rd	4th	5th
MELG1279-64	0.89	0.41	0.11	0.70	0.22
MELG2281-64	0.70	0.02	0.62	0.49	0.98
MELG19937-64	0.23	0.13	0.32	0.14	0.85
SFMT1279-64	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$
SFMT2281-64	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-300}$
SFMT19937-64	0.29	0.02	0.06	0.49	0.83

SFMT19937-64 denote the results for double-precision floating-point numbers in $[0, 1)$ converted from the 64-bit integer outputs of SFMT1279, SFMT2281, and SFMT19937 in [Saito and Matsumoto 2008], respectively.)

Remark 4.1. We occasionally see the use of the least significant bits of pseudorandom numbers in applications. An example is the case in which one generates uniform integers from 0 to 15 by taking the bit mask of the 4 least significant bits or modulo 16. For this, we invert the order of the bits (i.e., the i-th bit is exchanged with the $(w-i)$-th bit) in each integer and compute the dimension of equidistribution with v-bit accuracy, dimension defect at v, and total dimension defect for inversion, which are denoted by $k'(v)$, $d'(v)$, and Δ', respectively. In this case, MELG19937-64 is not maximally equidistributed, but $\Delta' = 4047$ and $d'(v)$ is 0 or 1 for each $v \leq 11$. Note that Δ' takes values 9022, 8984, 21341 for MT19937-64, MT19937-64 (ID3), SFMT19937-64, respectively. Δ' of MELG19937-64 is still smaller than Δ of the other generators in Table II. However, as far as possible, we recommend using the most significant bits (e.g., by taking the right-shift in the above example), because our generators are optimized preferentially from the most significant bits.

Remark 4.2. For 32-bit generators, there have been some implementations that produce 64-bit unsigned integers or 53-bit double-precision floating-point numbers (in IEEE 754 format) by concatenating two consecutive 32-bit unsigned integers. We note that such conversions might not be preferable from the viewpoint of $k(v)$. As an example, consider the 32-bit MT19937 generator in the header `<random>` of the C++11 STL in GCC (see [ISO 2012, Chapter 26.5]). Let $\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, \ldots \in \mathbb{F}_2^{32}$ be a 32-bit unsigned integer sequence from 32-bit MT19937. To obtain 64-bit unsigned integers, the GCC implements a random engine adaptor `independent_bit_engine` to produce 64-bit unsigned integers from the concatenations as

$$(\mathbf{z}_0, \mathbf{z}_1), (\mathbf{z}_2, \mathbf{z}_3), (\mathbf{z}_4, \mathbf{z}_5), (\mathbf{z}_6, \mathbf{z}_7), \ldots \in \mathbb{F}_2^{64}. \quad (15)$$

To generate 53-bit double-precision floating-point numbers in $[0, 1)$ (i.e., `uniform_real_distribution(0,1)` for MT19937), the GCC implementation generates 64-bit unsigned integers

$$(\mathbf{z}_1, \mathbf{z}_0), (\mathbf{z}_3, \mathbf{z}_2), (\mathbf{z}_5, \mathbf{z}_4), (\mathbf{z}_7, \mathbf{z}_6), \ldots \in \mathbb{F}_2^{64} \quad (16)$$

by concatenating two consecutive 32-bit integer outputs and divides them by the maximum value 2^{64}. The sequences (15) and (16) can be viewed as $\mathbb{F}_2$-linear generators with the state transition f^2 in (3), so that we can compute $k(v)$. Note that the sequences (15) and (16) are different: (16) is obtained by exchanging the 32 most significant bits for the 32 least significant bits in each 64-bit word in (15), that is, the $\mathbb{F}_2$-linear output functions are described as $s_i \in S \mapsto (o(s_i), o(f(s_i))) \in \mathbb{F}_2^{64}$ in (15) and $s_i \in S \mapsto (o(f(s_i)), o(s_i)) \in \mathbb{F}_2^{64}$ in (16). In fact, their $k(v)$'s are different for $v = 33, \ldots, 64$. As a result, we have $\Delta = 13543$ for $v = 1, \ldots, 64$ in (15) and $\Delta = 13161$ for $v = 1, \ldots, 52$ in (16), which are worse than Δ of MT19937-64. In particular, $k(12) = 623 < \lfloor 19937/12 \rfloor = 1661$ for each case. For this reason, we feel that there is a need to design 64-bit high-quality pseudorandom number generators.

5. CONCLUSIONS

In this article, we have designed 64-bit maximally equidistributed $\mathbb{F}_2$-linear generators with Mersenne prime period and searched for specific parameters with period lengths from $2^{607}-1$ to $2^{44497}-1$. The key techniques are (i) state transitions with double feedbacks and (ii) linear output transformations with several memory references. As a result, the generation speed is still competitive with 64-bit Mersenne Twisters on some platforms. The code in C is available at https://github.com/sharase/melg-64. Pseudorandom number generation is a trade-off between speed and quality. Our generators offer both high performance and computational efficiency.

ACKNOWLEDGMENTS

The authors would like to thank Associate Editor Professor Pierre L'Ecuyer and the anonymous reviewers for their valuable comments and suggestions. The authors also wish to express their gratitude to Professor Makoto Matsumoto at Hiroshima University for many helpful comments and Professor Syoiti Ninomiya at Tokyo Institute for continuous encouragement.

REFERENCES

A. Compagner. 1991. The hierarchy of correlations in random binary sequences. *Journal of Statistical Physics* 63 (1991), 883–896. Issue 5. http://dx.doi.org/10.1007/BF01029989 10.1007/BF01029989.

R. Couture and P. L'Ecuyer. 2000. Lattice computations for random numbers. *Math. Comput.* 69, 230 (2000), 757–765. DOI:http://dx.doi.org/10.1090/S0025-5718-99-01112-6

H. Haramoto, M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer. 2008. Efficient jump ahead for $\mathbb{F}_2$-linear random number generators. *INFORMS J. Comput.* 20, 3 (2008), 385–390.

S. Harase. 2009. Maximally equidistributed pseudorandom number generators via linear output transformations. *Math. Comput. Simul.* 79, 5 (2009), 1512–1519. DOI:http://dx.doi.org/10.1016/j.matcom.2008.06.006

S. Harase. 2011. An efficient lattice reduction method for $\mathbf{F}_2$-linear pseudorandom number generators using Mulders and Storjohann algorithm. *J. Comput. Appl. Math.* 236, 2 (2011), 141 – 149. DOI:http://dx.doi.org/10.1016/j.cam.2011.06.005

S. Harase, M. Matsumoto, and M. Saito. 2011. Fast lattice reduction for $\mathbf{F}_2$-linear pseudorandom number generators. *Math. Comput.* 80, 273 (2011), 395–407.

ISO. 2012. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland. 1338 (est.) pages.

D. E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

P. L'Ecuyer. 1999. Tables of maximally equidistributed combined LFSR generators. *Math. Comp.* 68, 225 (1999), 261–269. DOI:http://dx.doi.org/10.1090/S0025-5718-99-01039-X

P. L'Ecuyer and F. Panneton. 2009. $\mathbb{F}_2$-Linear Random Number Generators. In *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, C. Alexopoulos, D. Goldsman, and J. R. Wilson (Eds.). Springer-Verlag, New York, 169–193.

P. L'Ecuyer and R. Simard. 2007. TestU01: a C library for empirical testing of random number generators. *ACM Trans. Math. Software* 33, 4 (2007), Art. 22, 40. DOI:http://dx.doi.org/10.1145/1268776.1268777

J. H. Lindholm. 1968. An analysis of the pseudo-randomness properties of subsequences of long m-sequences. *IEEE Trans. Inform. Theory* IT-14, 4 (1968), 569–576.

M. Matsumoto and Y. Kurita. 1996. Strong deviations from randomness in m-sequences based on trinomials. *ACM Trans. Model. Comput. Simul.* 6, 2 (apr 1996), 99–106. DOI:http://dx.doi.org/10.1145/232807.232815

M. Matsumoto and T. Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (1998), 3–30. DOI:http://dx.doi.org/10.1145/272991.272995

M. Matsumoto and T. Nishimura. 2002. A nonempirical test on the weight of pseudorandom number generators. (2002), 381–395.

M. Matsumoto, M. Saito, H. Haramoto, and T. Nishimura. 2006. Pseudorandom Number Generation: Impossibility and Compromise. *J. Univer. Comput. Sci.* 12, 6 (2006), 672–690.

M. Matsumoto, I. Wada, A. Kuramoto, and H. Ashihara. 2007. Common Defects in Initialization of Pseudorandom Number Generators. *ACM Trans. Model. Comput. Simul.* 17, 4, Article 15 (sep 2007). DOI:http://dx.doi.org/10.1145/1276927.1276928

H. Niederreiter. 1992. *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM CBMS-NSF Reginal Conference Series in Applied Mathematics, vol. 63. SIAM, Philadelphia.

T. Nishimura. 2000. Tables of 64-bit Mersenne twisters. *ACM Trans. Model. Comput. Simul.* 10, 4 (2000), 348–357. DOI:http://dx.doi.org/10.1145/369534.369540

F. Panneton, P. L'Ecuyer, and M. Matsumoto. 2006. Improved long-period generators based on linear recurrences modulo 2. *ACM Trans. Math. Softw.* 32, 1 (2006), 1–16.

M. Saito and M. Matsumoto. 2008. SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, A. Keller, S. Heinrich, and H. Niederreiter (Eds.). Springer-Verlag, Berlin, 607–622. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html.

M. Saito and M. Matsumoto. 2009. A PRNG specialized in double precision floating point numbers using an affine transition. In *Monte Carlo and quasi-Monte Carlo methods 2008*. Springer, Berlin, 589–602. DOI:http://dx.doi.org/10.1007/978-3-642-04107-5_38

M. Saito and M. Matsumoto. 2013. Variants of Mersenne twistor suitable for graphic processors. *ACM Trans. Math. Software* 39, 2 (2013), Art. 12, 20. DOI:http://dx.doi.org/10.1145/2427023.2427029

J. P. R. Tootill, W. D. Robinson, and D. J. Eagle. 1973. An Asymptotically Random Tausworthe Sequence. *J. ACM* 20, 3 (1973), 469–481. DOI:http://dx.doi.org/10.1145/321765.321778

D. K. Wang and A. Compagner. 1993. On the use of reducible polynomials as random number generators. *Math. Comp.* 60, 201 (1993), 363–374. DOI:http://dx.doi.org/10.2307/2153173

Received May 2015; revised May 2016, March 2017; accepted November 2017

Online Appendix to: Implementing 64-bit Maximally Equidistributed $\mathbb{F}_2$-Linear Generators with Mersenne Prime Period

SHIN HARASE, Ritsumeikan University
TAKAMITSU KIMOTO, Recruit Holdings Co., Ltd.

A. THE 64-BIT OUTPUT SEQUENCES OF SFMT19937

In this appendix, we examine the 64-bit integer output sequences of the SFMT19937 generator (or the double-precision floating-point numbers in $[0, 1)$ converted from them). In fact, the SFMT generators are optimized under the assumption that one will mainly be using 32-bit output sequences, so that the dimensions of equidistribution with v-bit accuracy for 64-bit output sequences are worse than those for 32-bit cases. We therefore study the structure of SFMT19937 and point out its weaknesses. We also apply empirical statistical tests to non-successive values of SFMT19937 and find that the generator fails them.

A.1. $\mathbb{F}_2$-Linear Relations of SFMT19937

In the case of $\mathbb{F}_2$-linear generators, there are always certain bits of output whose sum in $\mathbb{F}_2$ becomes 0 in dimensions higher than $k(v)$. Such relations are said to be $\mathbb{F}_2$-*linear relations*. When there exist $\mathbb{F}_2$-linear relations with small numbers of terms (e.g., ≤ 6), use of the generator might carry risks in some situations [Matsumoto and Nishimura 2002]. We call the number of nonzero terms of an $\mathbb{F}_2$-linear relation the *weight*. In previous work, Harase [2014] showed that MT19937 has low-weight $\mathbb{F}_2$-linear relations in $(k(v) + 1)$-dimensional output values and that, as a result, some non-random bit patterns are detectable in statistical tests. In this subsection, we investigate the $\mathbb{F}_2$-linear relations of SFMT19937 as well.

SFMT19937 is an $\mathbb{F}_2$-linear generator with $w = 128$ and $p = 19968$. The period is a multiple of $2^{19937} - 1$. Note that the state $s_i \in S$ in Definition 2.1 has 31 more bits than 19937, so that the period may exceed $2^{19937} - 1$ (see also Proposition 1 in [Saito and Matsumoto 2008] for the periodicity). We set $N := p/w = 156$. The state transition $f : S \to S$ is expressed as $(\mathbf{w}_i, \ldots, \mathbf{w}_{i+N-1}) \mapsto (\mathbf{w}_{i+1}, \ldots, \mathbf{w}_{i+N})$ with the recursion

$$\mathbf{w}_{i+N} := \mathbf{w}_i \tilde{A} \oplus \mathbf{w}_{i+M} \tilde{B} \oplus \mathbf{w}_{i+N-2} \tilde{C} \oplus \mathbf{w}_{i+N-1} \tilde{D}. \tag{1}$$

Here $\mathbf{w}_0, \mathbf{w}_1, \ldots$ are 128-bit integers, $M = 122$, and $\tilde{A}$, $\tilde{B}$, $\tilde{C}$, $\tilde{D}$ are suitable (128×128)-matrices. SFMT19937 generates a 128-bit sequence $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots$, as in (2) of Definition 2.1. We denote $\mathbf{w}_i =: (w_{i,0}, \ldots, w_{i,128}) \in \mathbb{F}_2^{128}$. When implementing this generator on 32-bit computers, the 128-bit integers $\mathbf{w}_i$ are devided into four 32-bit integers $\mathbf{x}_i[0], \mathbf{x}_i[1], \mathbf{x}_i[2], \mathbf{x}_i[3]$ as follows:

$$\overbrace{w_{i,0}, \ldots, w_{i,31}}^{\mathbf{x}_i[3]}, \overbrace{w_{i,32}, \ldots, w_{i,63}}^{\mathbf{x}_i[2]}, \overbrace{w_{i,64}, \ldots, w_{i,95}}^{\mathbf{x}_i[1]}, \overbrace{w_{i,96}, \ldots, w_{i,127}}^{\mathbf{x}_i[0]}. \tag{2}$$

Note that the above four 32-bit integers are indexed starting from the least significant bits. Thus, the 32-bit output sequence of SFMT19937 is obtained as

$$\mathbf{x}_0[0], \mathbf{x}_0[1], \mathbf{x}_0[2], \mathbf{x}_0[3], \mathbf{x}_1[0], \mathbf{x}_1[1], \mathbf{x}_1[2], \mathbf{x}_1[3], \ldots \in \mathbb{F}_2^{32}$$

DOI: http://dx.doi.org/10.1145/0000000.0000000

Table I. Dimensions of Equidistribution $k(v)$ for SFMT19937-64 and Upper Bounds $\lfloor 19937/v \rfloor$ ("max")

v	$k(v)$	max	v	$k(v)$	max	v	$k(v)$	max	v	$k(v)$	max
1	19936	19937	9	2214	2215	17	622	1172	25	312	797
2	9968	9968	10	1992	1993	18	622	1107	26	312	766
3	6644	6645	11	1812	1812	19	312	1049	27	312	583
4	4982	4984	12	1556	1661	20	312	996	28	312	738
5	3986	3987	13	934	1533	21	312	949	29	312	712
6	3321	3322	14	934	1424	22	312	906	30	312	664
7	2847	2848	15	622	1329	23	312	866	31	312	643
8	2491	2492	16	622	1246	24	312	830	32	312	623

in this order. SFMT19937 is mainly optimized for 32-bit integer output (see [Saito and Matsumoto 2008] for details). On the other hand, the 64-bit integers of SFMT19937 (denoted by SFMT19937-64) are output as the concatenations

$$(\mathbf{x}_0[1], \mathbf{x}_0[0]), (\mathbf{x}_0[3], \mathbf{x}_0[2]), (\mathbf{x}_1[1], \mathbf{x}_1[0]), (\mathbf{x}_1[3], \mathbf{x}_1[2]), \ldots \in \mathbb{F}_2^{64} \tag{3}$$

in that order. The SFMT generator is $\mathbb{F}_2$-linear as a 128-bit generator, but the 32- and 64-bit output sequences are no longer $\mathbb{F}_2$-linear, according to Definition 2.1. To define $k(v)$ for 32- or 64-bit output of the SFMT generator, we need to modify Definitions 2.1 and 2.2. For brevity, we omit the precise definition of $k(v)$ for the SFMT generator (see [Saito and Matsumoto 2008] for details).

Table I summarizes the dimensions of equidistribution $k(v)$. For SFMT19937-64, $k(33) = \cdots = k(63) = 312$ and $k(64) = 310$, so we omitted them from the table. (Recently, corrections to the tables of $k(v)$ for SFMT generators have been reported; see http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html. Thus, we re-computed $k(v)$ by using MTToolBox [Saito 2013].) Note that the $k(v)$'s are much lower than those for the 32-bit output sequences from SFMT19937. For example, $k(19) = 624$ for 32-bit output sequences.

Here we take a closer look at the most significant bits of (2). In fact, from Eq. (2) and Fig. 1 of [Saito and Matsumoto 2008], $\tilde{B}$ and $\tilde{C}$ are the right-shift (with some bit masks), so $\mathbf{w}_{i+M}\tilde{B}$ and $\mathbf{w}_{i+N-2}\tilde{C}$ do not affect the generation of the most significant bits. As a result, we can find the four-term $\mathbb{F}_2$-linear relations on $\mathbf{x}_0[3], \mathbf{x}_1[3], \mathbf{x}_2[3], \ldots$ as

$$w_{i,j} + w_{i,j+8} + w_{i+155,j+18} + w_{i+156,j} = 0 \quad (j = 0, \ldots, 8). \tag{4}$$

In particular, the relation (4) implies that $k(19)$ for SFMT19937-64 is at most 312, because such an $\mathbb{F}_2$-linear relation destroys the surjectivity of the map in (2) of Proposition 2 of [Saito and Matsumoto 2008]. This is why the dimensions of equidistribution $k(v)$ of SFMT19937-64 rapidly decrease for $v \geq 19$.

Remark A.1. For $\mathbb{F}_2$-linear generators, $\mathbb{F}_2$-linear relations coincide with the vectors in the (dual) lattices of [Couture and L'Ecuyer 2000] (with components in the polynomial ring $\mathbb{F}_2[x]$) associated to output sequences. Harase [2014] proposed a method to detect low-weight $\mathbb{F}_2$-linear relations in $(k(v)+1)$-dimensional output with v-bit accuracy, i.e., among $(k(v)+1)v$ bits. By use of this method, we can also detect the $\mathbb{F}_2$-linear relations on $\mathbf{x}_0[1], \mathbf{x}_1[1], \mathbf{x}_2[1], \ldots$:

$$w_{i,64+j+8} + w_{i,64+j+16} + w_{i+154,64+j} + w_{i+156,64+j+8} = 0 \quad (j = 7, 10, 11), \tag{5}$$

for example.

Remark A.2. For the SFMT generator, when we convert 64-bit integers into double-precision floating-point numbers in $[0, 1)$, the dimensions of equidistribution drastically decrease compared with the conversion from 32-bit integers. Saito and Matsumoto [2009] developed the dSFMT generator, which is specialized for generating double-precision floating-point numbers based on the IEEE 754-2008 format (IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-2008). For double-precision floating-point numbers, dSFMT is faster than SFMT and is also improved from the viewpoint of the dimensions of equidistribution with v-bit accuracy. Thus, as far as SIMD-oriented generators are concerned, for generating double-precision floating-point numbers, the dSFMT generator is preferable to conversion from the 64-bit integers of the SFMT generator (i.e., a function `sfmt_genrand_res53()`, which is obtained by dividing (3) by 2^{64}). Note that dSFMT directly generates double-precision floating-point numbers with 52-bit accuracy and does not support 64-bit integer output sequences. Note also that the dSFMT generator is not maximally equidistributed ($\Delta = 2616$).

A.2. Birthday Spacings Tests for Non-Successive Values

L'Ecuyer and Touzin [2004] and L'Ecuyer and Simard [2014] reported that some multiple recursive generators based on sparse characteristic polynomials have a structural weakness and fail a standard statistical test (called the *birthday spacings test*; see [Marsaglia 1985; Knuth 1997; L'Ecuyer and Simard 2001; 2007; Lemieux 2009]). Harase [2014] pointed out that 32-bit MT19937 fails this test for non-successive values in accordance with the existence of low-weight $\mathbb{F}_2$-linear relations. In this subsection, we show that SFMT19937-64 fails the birthday spacing tests in a similar fashion.

For the birthday spacings test, we select two positive integers n and d and generate n "independent" points $\mathbf{u}_0, \dots, \mathbf{u}_{n-1}$ in the t-dimensional hypercube $[0, 1)^t$. We partition the hypercube into d^t cubic boxes of equal size by dividing $[0, 1)$ into d equal segments. These boxes are numbered from 0 to $d^t - 1$ in lexicographic order. Let $I_1 \leq I_2 \leq \cdots \leq I_n$ be the numbers of the boxes where these points have fallen, sorted by increasing order. Define the spacings $S_j := I_{j+1} - I_j$, for $j = 1, \dots, n-1$, and let Y be the total number of collisions of these spacings, i.e., the number of values of $j \in \{1, \dots, n-2\}$ such that $S_{(j+i)} = S_{(i)}$, where $S_{(1)}, \dots, S_{(n-1)}$ are the spacings sorted by increasing order. We test the null hypothesis $\mathcal{H}_0$ that the generator's output is perfectly random. If d^t is large and $\lambda = n^3/(4d^t)$ is not too large, Y is approximately a Poisson distribution with mean λ under $\mathcal{H}_0$. Further, we generate N independent replications of Y, add them, and compute the p-value by using the sum, which is approximately a Poisson distribution with mean $N\lambda$, under $\mathcal{H}_0$. If $d = 2^v$, the t-dimensional output with v-bit accuracy is tested.

Next, for a sequence $u_0, u_1, u_2, \dots \in [0, 1)$, we extract non-successive values and construct t-dimensional output vectors $\mathbf{u}_i = (u_{(j_t+1)i+j_1}, \dots, u_{(j_t+1)i+j_t})$ for $i = 0, \dots, n-1$ with a lacunary filter $I = \{j_1, \dots, j_t\}$. We also drop the r most significant bits, left-shift the others by τ positions, and return a floating-point number in $[0, 1)$. In other words, the output sequence is $2^\tau u_i \bmod 1$. If $\tau = 0$, this is the usual output sequence, and if $\tau > 0$, the least significant bits of the integer outputs are investigated. We use the birthday spacings tests implemented in the TestU01 package [L'Ecuyer and Simard 2014].

From the 64-bit integer output of SFMT19937-64, we generate double-precision floating-point numbers $u_0, u_1, \dots \in [0, 1)$, which are obtained by the function `sfmt_genrand_res53()`. We select the lacunary filter $I = \{1, 311, 313\}$. Note that these indices are twice as much as in (4), because we split 128-bit integers (2) into two 64-bit integers and interleave them. We select the parameter set $(N, n, \tau, d, t) = (5, 20000000, 0, 2^{21}, 3)$, which comes from the parameter set of test No. 12 of Crush in TestU01. The second row of Table II shows the right p-values for five initial values. SFMT19937-64 decisively fails the birthday spacings tests. Further, we select the

Table II. p-Values on the Birthday Spacings Tests with Lacunary Filters I for SFMT19937-64

	1st	2nd	3rd	4th	5th
$I = \{1, 311, 313\}$ $(N = 5)$	1.2×10^{-241}	9.4×10^{-281}	6.5×10^{-288}	5.1×10^{-274}	1.3×10^{-256}
$I = \{0, 308, 312\}$ $(N = 5)$	0.09	0.60	3.1×10^{-3}	4.0×10^{-3}	5.7×10^{-3}
$I = \{0, 308, 312\}$ $(N = 20)$	5.2×10^{-3}	7.5×10^{-4}	3.9×10^{-5}	8.0×10^{-7}	9.6×10^{-4}

filter $I = \{0, 308, 312\}$ corresponding to (5), and the parameter sets $(N, n, \tau, d, t) = (5, 20000000, 7, 2^{21}, 3)$ and $(N, n, \tau, d, t) = (20, 20000000, 7, 2^{21}, 3)$. Note that $\tau > 0$. The bottom two rows of Table II show the right p-values for five initial values. The results also seem to be very suspicious.

Remark A.3. As mentioned in Section 7 of [Saito and Matsumoto 2008], in the case of $\mathbb{F}_2$-linear generators, the dimension of equidistribution $k(v)$ with v-bit accuracy means that there is no constant $\mathbb{F}_2$-linear relation among the $k(v)v$ bits, but there does exist an $\mathbb{F}_2$-linear relation among the $(k(v)+1)v$ bits, where $k(v)v$ and $(k(v)+1)v$ bits are taken from the consecutive $k(v)$ and $k(v)+1$ integers, respectively, by extracting the v most significant bits from each. For our 64-bit MELGs, we investigated $\mathbb{F}_2$-linear relations among the above $(k(v)+1)v$ bits by using the method of Harase [2014]. We detected no low-weight $\mathbb{F}_2$-linear relation (e.g., the number of weights ≤ 20) among the above $(k(v)+1)v$ bits within the v most significant bits for $1 \leq v \leq 32$. (In the case of MELG19937-64, the above weights are ≥ 9500 for $1 \leq v \leq 32$.) Thus, according to [Matsumoto and Nishimura 2002], this result implies that the existence of a bad lacunary filter I for the 64-bit MELGs is avoided so long as we focus on the consecutive $(k(v)+1)v$ bits for $1 \leq v \leq 32$, which correspond to just one dimension beyond $k(v)$.

REFERENCES

R. Couture and P. L'Ecuyer. 2000. Lattice computations for random numbers. *Math. Comput.* 69, 230 (2000), 757–765. DOI:http://dx.doi.org/10.1090/S0025-5718-99-01112-6

S. Harase. 2014. On the $\mathbb{F}_2$-linear relations of Mersenne Twister pseudorandom number generators. *Math. Comput. Simul.* 100 (2014), 103–113. DOI:http://dx.doi.org/10.1016/j.matcom.2014.02.002

D. E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

P. L'Ecuyer and R. Simard. 2001. On the performance of birthday spacings tests with certain families of random number generators. *Math. Comput. Simulation* 55, 1-3 (2001), 131–137. DOI:http://dx.doi.org/10.1016/S0378-4754(00)00253-6 The Second IMACS Seminar on Monte Carlo Methods (Varna, 1999).

P. L'Ecuyer and R. Simard. 2007. TestU01: a C library for empirical testing of random number generators. *ACM Trans. Math. Software* 33, 4 (2007), Art. 22, 40. DOI:http://dx.doi.org/10.1145/1268776.1268777

P. L'Ecuyer and R. Simard. 2014. On the lattice structure of a special class of multiple recursive random number generators. *INFORMS J. Comput.* 26, 3 (2014), 449–460. DOI:http://dx.doi.org/10.1287/ijoc.2013.0576

P. L'Ecuyer and R. Touzin. 2004. On the Deng-Lin random number generators and related methods. *Statistics and Computing* 14 (January 2004), 5–9. DOI:http://dx.doi.org/10.1023/B:STCO.0000009417.88960.81

C. Lemieux. 2009. *Monte Carlo and quasi-Monte Carlo sampling*. Springer, New York. xvi+373 pages.

G. Marsaglia. 1985. A current view of random number generators. In *Computer Science and Statistics, Sixteenth Symposium on the Interface*. Elsevier Science Publisher, North-Holland, Amsterdam, The Netherlands, 3–10.

M. Matsumoto and T. Nishimura. 2002. A nonempirical test on the weight of pseudorandom number generators. (2002), 381–395.

M. Saito. 2013. MTToolBox. (2013). https://github.com/MSaito/MTToolBox.

M. Saito and M. Matsumoto. 2008. SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, A. Keller, S. Heinrich, and H. Niederreiter (Eds.). Springer-Verlag, Berlin, 607–622. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html.

M. Saito and M. Matsumoto. 2009. A PRNG specialized in double precision floating point numbers using an affine transition. In *Monte Carlo and quasi-Monte Carlo methods 2008*. Springer, Berlin, 589–602. DOI:http://dx.doi.org/10.1007/978-3-642-04107-5_38