Les Cahiers du GERAD ISSN: 0711–2440

$\mathbb{F}_2$-Linear Random Number Generators

P. L'Ecuyer
F. Panneton

G–2007–21

March 2007

$\mathbb{F}_2$-Linear Random Number Generators

Pierre L'Ecuyer[*]
François Panneton

Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, Succ. Centre-ville
Montréal (Québec) Canada, H3C 3J7
{lecuyer, panneton}@iro.umontreal.ca

[*] *and GERAD*

March 2007

Les Cahiers du GERAD

G–2007–21

Abstract

Random number generators based on linear recurrences modulo 2 are among the fastest long-period generators currently available. The uniformity and independence of the points they produce, by taking vectors of successive output values from all possible initial states, can be measured by theoretical figures of merit that can be computed quickly, and the generators having good values for these figures of merit are statistically reliable in general. Some of these generators can also provide disjoint streams and substreams efficiently. In this chapter, we review the most interesting construction methods for these generators, examine their theoretical and empirical properties, describe the relevant computational tools and algorithms, and make comparisons.

Résumé

Les générateurs de nombres aléatoires basés sur des récurrences modulo 2 sont parmi les générateurs plus rapides parmi ceux ayant de très grandes périodes. On peut mesurer l'uniformité et l'indépendance des vecteurs de valeurs successives produits par ces générateurs par des figures de mérite théoriques rapidement calculables. Les générateurs ayant de bonnes valeurs pour ces figures de mérite sont habituellement fiables du point de vue statistique. Certains de ces générateurs peuvent aussi fournir des sous-séquences disjointes de manière efficace. Dans cet article, nous examinons et comparons les méthodes de construction les plus intéressantes pour ces générateurs, nous examinons leurs propriétés théoriques et empiriques, et nous décrivons les algorithmes et les outils de calcul pertinents pour contruire et analyser de tels générateurs.

Acknowledgments: This work has been supported by NSERC-Canada grant No. ODGP0110050 and a Canada Research Chair to the first author. This chapter was written while the first author was a visiting scientist at the University of Salzburg, Austria, in 2005, and at IRISA, Rennes, in 2006. A short draft of it appeared in the proceedings of the 2005 Winter Simulation Conference. Richard Simard helped doing the speed tests.

1 Introduction

Given that computers work in binary arithmetic, it seems natural to construct random number generators (RNGs) defined via recurrences in arithmetic modulo 2, so that these RNGs can be implemented efficiently via elementary operations on bit strings, such as shifts, rotations, exclusive-or's (xor's), and bit masks. Very fast RNGs whose output sequences have huge periods can be constructed in this way. Among them, we find the Tausworthe or linear feedback shift register (LFSR), generalized feedback shift register (GFSR), twisted GFSR (TGFSR), Mersenne twister, the WELL, and xorshift generators (Tezuka 1995, L'Ecuyer 1996, Matsumoto and Nishimura 1998, L'Ecuyer and Panneton 2002, L'Ecuyer 2006, Panneton, L'Ecuyer, and Matsumoto 2006, Panneton and L'Ecuyer 2005a, Panneton 2004). A common characterization of all these generators is that they are special cases of a general class of generators whose state evolves according to a (matrix) linear recurrence modulo 2. The bits that form their output are also determined by a linear transformation modulo 2 applied to the state. Since doing arithmetic modulo 2 can be interpreted as working in $\mathbb{F}_2$, the finite field of cardinality 2 with elements $\{0, 1\}$, we shall refer to this general class as $\mathbb{F}_2$-*linear generators*.

Some widely-used RNGs of this form are *not* statistically reliable, but other well-designed instances are good, reliable, and fast. Which ones? What defects do the others hide? What mathematical tools can be used to analyze and practically assess their quality from a theoretical viewpoint? Is it easy to jump ahead quickly in their sequence in order to split it into multiple streams and substreams? In the remainder of this paper, we address these questions and provide a state-of-the-art overview of $\mathbb{F}_2$-linear RNGs.

In the next section, we define a general framework that covers all $\mathbb{F}_2$-linear generators. We provide some basic properties of these RNGs, such as maximal-period conditions, a simple way to jump ahead, and a simple combination method of $\mathbb{F}_2$-linear generators (via a bitwise xor) to construct larger (and often better-behaved) $\mathbb{F}_2$-linear generators. We describe efficient algorithms to compute the characteristic polynomial of an RNG and to check if it has maximal period. In Section 3, we discuss the theoretical measures of uniformity and independence that are typically used in practice as figures of merit to assess their quality. These RNGs turn out to have a lattice structure in spaces of polynomials and formal series over $\mathbb{F}_2$. There are counterparts in those spaces of the spectral test, and other lattice-based tests and properties that have been developed for linear congruential generators. Interestingly, these tests are strongly linked with computing the measures of uniformity of $\mathbb{F}_2$-linear generators. Section 4 outlines this theory. We explain how to construct and analyze the polynomial lattices and how to use them for computing the uniformity measures of interest. In Section 5, we describe specific families of $\mathbb{F}_2$-linear generators proposed over the years, show how they fit the general framework, and summarize what we know about their strengths and weaknesses. In Section 6, we compare specific implementations in terms of their speed and (theoretical) figures of merit, and discuss their behavior in empirical statistical tests. Compared with the most widely used

RNG that offers multiple streams and substreams in simulation software, the best $\mathbb{F}_2$-linear RNGs are faster by a factor of 1.5 to 3, depending on the computing platform. Section 7 concludes the paper.

2 $\mathbb{F}_2$-Linear Generators

2.1 General Framework

We consider an RNG defined by a matrix linear recurrence over the finite field $\mathbb{F}_2$, as follows:

$$\begin{aligned} \mathbf{x}_n &= \mathbf{A}\mathbf{x}_{n-1}, & (1)\\ \mathbf{y}_n &= \mathbf{B}\mathbf{x}_n, & (2)\\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1}2^{-\ell} \;=\; .y_{n,0}\, y_{n,1}\, y_{n,2}\, \cdots, & (3) \end{aligned}$$

where $\mathbf{x}_n = (x_{n,0},\ldots, x_{n,k-1})^{\mathsf{t}} \in \mathbb{F}_2^k$ is the k-bit *state vector* at step n, $\mathbf{y}_n = (y_{n,0},\ldots, y_{n,w-1})^{\mathsf{t}} \in \mathbb{F}_2^w$ is the w-bit *output vector* at step n, k and w are positive integers, $\mathbf{A}$ is a $k\times k$ *transition matrix* with elements in $\mathbb{F}_2$, $\mathbf{B}$ is a $w\times k$ *output transformation matrix* with elements in $\mathbb{F}_2$. The real number $u_n \in [0,1)$ is the *output* at step n. All operations in (1) and (2) are performed in $\mathbb{F}_2$, i.e., modulo 2. This setting is from L'Ecuyer and Panneton (2002). Several popular classes of RNGs fit this framework as special cases, by appropriate choices of the matrices $\mathbf{A}$ and $\mathbf{B}$. Many will be described in Section 5.

The period of this RNG cannot exceed 2^k-1, because there are only 2^k-1 possible values for $\mathbf{x}_n$. When this maximum is reached, we say that the RNG has *maximal period.* To discuss the periodicity and see how we can construct maximal-period $\mathbb{F}_2$-linear RNGs, we use the following basic definitions and properties from linear algebra and finite fields. Let $\mathbb{F}_2[z]$ denote the ring of polynomials with coefficients in $\mathbb{F}_2$. The *characteristic polynomial* of the matrix $\mathbf{A}$ is

$$P(z) = \det(z\mathbf{I} - \mathbf{A}) = z^k - \alpha_1 z^{k-1} - \cdots - \alpha_{k-1}z - \alpha_k,$$

where $\mathbf{I}$ is the identity matrix and each α_j is in $\mathbb{F}_2$. This $P(z)$ is also the characteristic polynomial of the linear recurrence (in $\mathbb{F}_2$)

$$x_n = (\alpha_1 x_{n-1} + \cdots + \alpha_k x_{n-k}). \qquad (4)$$

We shall assume that $\alpha_k = 1$. Usually, we know a priori that this is true by construction of the matrix $\mathbf{A}$. In that case, the recurrence (4) has *order k* and it is purely periodic, i.e., there is some integer $\rho > 0$ such that $(x_\rho,\ldots,x_{\rho+k-1}) = (x_0,\ldots,x_{k-1})$; this ρ is called the *period* of the recurrence. The *minimal polynomial* of $\mathbf{A}$ is the polynomial $Q(z) \in \mathbb{F}_2[z]$ of smallest degree for which $Q(\mathbf{A}) = 0$. Every other polynomial $R(z) \in \mathbb{F}_2[z]$ for which

$R(\mathbf{A}) = 0$ must be a multiple of the minimal polynomial. This implies in particular that $P(z)$ is a multiple of $Q(z)$. In the context of RNG construction, $Q(z)$ and $P(z)$ are almost always identical, at least for good constructions.

The fact that the sequence $\{\mathbf{x}_n,\, n \geq 0\}$ obeys (1) implies that it satisfies the recurrence that corresponds to the minimal polynomial of $\mathbf{A}$ (or any other polynomial that is a multiple of $Q(z)$):

$$\mathbf{x}_n = (\alpha_1 \mathbf{x}_{n-1} + \cdots + \alpha_k \mathbf{x}_{n-k}) \qquad (\text{in } \mathbb{F}_2). \tag{5}$$

This means that the sequence $\{x_{n,j},\, n \geq 0\}$ obeys (4) for each j, $0 \leq j < k$. The sequence $\{y_{n,j},\, n \geq 0\}$, for $0 \leq j < w$, also obeys that same recurrence. However, these sequences may also follow recurrences of order smaller than k. For any periodic sequence in $\mathbb{F}_2$, there is a linear recurrence of *minimal order* obeyed by this sequence, and the characteristic polynomial of that recurrence is called the *minimal polynomial* of the sequence. This minimal polynomial can be computed by the Berlekamp-Massey algorithm (Massey 1969). The sequences $\{x_{n,j},\, n \geq 0\}$ may have different minimal polynomials for different values of j, and also different minimal polynomials than the sequences $\{y_{n,j},\, n \geq 0\}$. But all these minimal polynomials must necessarily divide $P(z)$. If $P(z)$ is *irreducible* (it has no divisor other than 1 and itself), then $P(z)$ must be the minimal polynomial of all these sequences. Reducible polynomials $P(z)$ do occur when we combine generators (Section 2.3); in that case, $P(z)$ is typically the minimal polynomial of the output bit sequences $\{y_{n,j},\, n \geq 0\}$ as well, but the sequences $\{x_{n,j},\, n \geq 0\}$ often have much smaller minimal polynomials (divisors of $P(z)$).

It is well-known that the recurrences (4) and (5) have maximal period if and only if $P(z)$ is a primitive polynomial over $\mathbb{F}_2$ (Niederreiter 1992, Knuth 1998). Primitivity is a stronger property than irreducibility: $P(z)$ is *primitive* if and only if it is irreducible and for all prime divisors p_i of $r = 2^k - 1$, $z^{r/p_i} \not\equiv 1 \bmod P(z)$. A good way to verify if a polynomial is primitive is to verify irreducibility first, and then check the second condition. Note that when r is prime (this type of prime is called a *Mersenne prime*), the second condition is automatically verified.

In the context of RNG construction, we are interested essentially only in maximal-period recurrences. The RNG is constructed either from a single maximal-period recurrence, on from a combination of maximal-period recurrences, as we shall explain later. Assuming that we are interested only in primitive polynomials $P(z)$, we can compute $P(z)$ and check its primitivity as follows.

We first run the generator for k steps from some arbitrary non-zero initial state $\mathbf{x}_0$ and we compute the minimal polynomial $Q_0(z)$ of $\{x_{n,0},\, n \geq 0\}$ with the Berlekamp-Massey algorithm. If $Q_0(z)$ has degree less than k, then $P(z)$ is necessarily reducible and we reject this generator; otherwise $P(z) = Q_0(z)$ and it remains to verify its primitivity. For this, we can use the following algorithm from Rieke, Sadeghi, and Poguntke (1998) and Panneton (2004); it verifies the set of necessary and sufficient conditions stated in Knuth (1998), page 30, but it also specifies in what order to perform the polynomial exponentiations:

```
Algorithm P  [Given $P(z)$ of degree $k$, returns TRUE iff $P(z)$ is primitive]:
Factorize $r = 2^k - 1 = p_1^{e_1} \cdots p_b^{e_b}$ where $p_1, \ldots, p_b$ are distinct prime;
Compute $q := r/(p_1 \cdots p_b)$ and $q_b(z) := z^q \bmod P(z)$;
For $i = b, \ldots, 1$, let $q_{i-1}(z) := q_i(z)^{p_i} \bmod P(z)$;
If $q_0(z) \neq 1$ or $q_1(z) = 1$, return FALSE;
For $i = b, \ldots, 2$, {
    Compute $t_i(z) := q_i(z)^{p_{i-1} \cdots p_1} \bmod P(z)$;
    If $t_i(z) = 1$, return FALSE; }
Return TRUE.
```

When k is large, it is worthwhile to first apply an irreducibility test that can detect reducibility faster than this primitivity test. Note that $P(z)$ is reducible if and only if it has an irreducible factor of degree $\leq \lfloor k/2 \rfloor$. A key theorem in finite fields theory states that for any integer $n \geq 1$, the product of all irreducible polynomials whose degree d divides n is equal to $z^{2^n} + z$. This means that $P(z)$ is irreducible if and only if $\text{pgcd}(z^{2^n} + z,\, P(z)) = 1$ for all $n \leq \lfloor k/2 \rfloor$. This gives the following algorithm:

```
Algorithm I  [Given $P(z)$ of degree $k$, returns TRUE iff $P(z)$ is irreducible]:
For $n = 1, \ldots, \lfloor k/2 \rfloor$: if $\text{pgcd}(z^{2^n} + z,\, P(z)) \neq 1$, return FALSE;
Return TRUE.
```

When searching for primitive polynomials for RNG construction, we typically select k and impose a special form on the matrix $\mathbf{A}$, so that a fast implementation is available (see Section 5). Then we search (often at random), in the space of matrices $\mathbf{A}$ that satisfy these constraints, for instances having a primitive characteristic polynomial. The following old result (see, e.g., Lidl and Niederreiter 1986) may give a rough idea of our chances of success. It gives the probability that a random polynomial, generated uniformly over the set of all polynomials of degree k, is primitive. It is important to underline, however, that when generating $\mathbf{A}$ randomly from a special class, the polynomial $P(z)$ does not necessarily have the uniform distribution over the set of polynomials, so the probability that it is primitive might differ from the formula given in the theorem.

Theorem 1 *Among the 2^k polynomials of degree k in $\mathbb{F}_2[z]$, the proportion of primitive polynomials is exactly*

$$\frac{1}{k} \prod_{i=1}^{b} \frac{p_i - 1}{p_i}$$

where $p_1, \ldots, p_b$ are the distinct prime factors of $r = 2^k - 1$.

This result suggests that to improve our chances, it is better to avoid values of r having several small factors. If r is a Mersenne prime, the proportion is exactly $1/k$.

2.2 Jumping Ahead

A key requirement of modern stochastic simulation software is the availability of random number generators with multiple disjoint streams and substreams. These streams and substreams can provide parallel RNGs and are also important to support the use of variance reduction techniques (Law and Kelton 2000, L'Ecuyer, Simard, Chen, and Kelton 2002). They are usually implemented by partitioning the output sequence of a long-period generator into long disjoint subsequences and subsubsequences whose starting points are found by making large jumps in the original sequence.

Jumping ahead directly from $\mathbf{x}_n$ to $\mathbf{x}_{n+\nu}$ for a very large integer ν is easy in principle with this type of generator. It suffices to precompute the matrix $\mathbf{A}^\nu$ mod 2 (this can be done in $O(k^3 \log \nu)$ operations by a standard method) and then multiply $\mathbf{x}_n$ by this binary matrix, modulo 2. The latter step requires $O(k^2)$ operations and $O(k^2)$ words of memory to store the matrix. This approach works fine for relatively small values of k (e.g., up to 100 or so), but becomes rather slow when k is large. For example, the Mersenne twister of Matsumoto and Nishimura (1998) has $k = 19937$ and the above method is impractical in that case.

A more efficient method is proposed by Haramoto, Matsumoto, Nishimura, Panneton, and L'Ecuyer (2006). For a given step size ν, the method represents the state $\mathbf{x}_{n+\nu}$ as $g_\nu(\mathbf{A})\mathbf{x}_n$, where $g_\nu(z) = \sum_{j=0}^{k-1} d_j z^j$ is a polynomial of degree less than k in $\mathbb{F}_2[z]$. The product

$$g_\nu(\mathbf{A})\mathbf{x}_n = \sum_{j=0}^{k-1} d_j \mathbf{A}^j \mathbf{x}_n = \sum_{j=0}^{k-1} d_j \mathbf{x}_{n+j}$$

can be computed simply by running the generator for $k-1$ steps to obtain $\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+k-1}$ and adding (modulo 2) the $\mathbf{x}_{n+j}$'s for which $d_j = 1$. For large k, the cost is dominated by these additions. Their number can be reduced (e.g., by a factor of about 4 when $k = 19937$) by using a sliding window technique, as explained in Haramoto, Matsumoto, Nishimura, Panneton, and L'Ecuyer (2006). This method still requires $O(k^2)$ operations but with a smaller hidden constant and (most importantly) much less memory than the standard matrix multiplication. Yet jumping ahead for $\mathbb{F}_2$-linear generators of large order k (such as the Mersenne twister) remains slow with this method. One way to make the jumping-ahead more efficient is to adopt a combined generator, as discussed in Subsection 2.3, and do the ν-step jumping-ahead separately for each component.

2.3 Combined $\mathbb{F}_2$-Linear Generators

A simple way of combining $\mathbb{F}_2$-linear generators is as follows. For some integer $C > 1$, consider C distinct recurrences of the form (1)–(2), where the cth recurrence has parameters $(k, w, \mathbf{A}, \mathbf{B}) = (k_c, w, \mathbf{A}_c, \mathbf{B}_c)$ and state $\mathbf{x}_{c,n}$ at step n, for $c = 1, \ldots, C$. The output of the combined generator at step n is defined by

$$\begin{aligned}\mathbf{y}_n &= \mathbf{B}_1\mathbf{x}_{1,n} \oplus \cdots \oplus \mathbf{B}_C\mathbf{x}_{C,n},\\ u_n &= \sum_{\ell=1}^{w} y_{n,\ell-1}2^{-\ell},\end{aligned}$$

where $\oplus$ denotes the bitwise exclusive-or (xor) operation. One can show (Tezuka and L'Ecuyer 1991, Tezuka 1995) that the period ρ of this combined generator is the least common multiple of the periods ρ_c of its components. This combined generator is equivalent to the generator (1)–(3) with $k = k_1+\cdots+k_C$, $\mathbf{A} = \mathrm{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_C)$, and $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_C)$. If $P_c(z)$ is the characteristic polynomial of $\mathbf{A}_c$ for each c, then the characteristic polynomial of $\mathbf{A}$ is $P(z) = P_1(z)\cdots P_C(z)$. This polynomial is obviously reducible, so the combined RNG cannot have maximal period $2^k - 1$. However, if we select the parameters carefully so that each component has maximal period $\rho_c = 2^{k_c} - 1$ and if the ρ_c are pairwise relatively prime (the $P_c(z)$ must be distinct irreducible polynomials), then the period of the combined generator is the product of the periods of the components: $\rho = \prod_{c=1}^{C}(2^{k_c} - 1)$. In fact, within one cycle, all combinations of nonzero states for the C components are visited exactly once. When the k_c's are reasonably large, this ρ is not far from $2^k - 1$; the difference is that instead of discarding a single k-bit zero state, we must discard the zero state for each component (i.e., all k-bit states in which at least one of the components is in the zero state). Concrete constructions of this form are given in (Tezuka and L'Ecuyer 1991, Wang and Compagner 1993, L'Ecuyer 1996, Tezuka 1995).

Why would we want to combine generators like this? We already gave one good reason in the previous subsection: efficient jumping-ahead is easier for a combined generator of order k having several components of smaller order than for a non-combined generator with the same k. Another important reason is that matrices $\mathbf{A}$ that give very fast implementations typically lead (unfortunately) to poor quality RNGs from the statistical viewpoint, because of a too simplistic structure. Combined generators provide a way out of this dilemma: select simple components that allow very fast implementations and such that the corresponding combined generator has a more complicated structure, good figures of merit from the theoretical viewpoint, and good statistical properties. Many of the best $\mathbb{F}_2$-linear generators are defined via such combinations. As an illustration, one may have four components of periods $2^{63}-1$, $2^{58}-1$, $2^{55}-1$, $2^{47}-1$, so the state of each component fits a 64-bit integer and the overall period is near 2^{223}.

There could be situations where instead of combining explicitly known $\mathbb{F}_2$-linear components, we would go the other way around; we may want to generate matrices $\mathbf{A}$ randomly from a given class, then find the decomposition of their (reducible) characteristic polynomials, analyze their periodicity and figures of merit, and so on. This approach is used by Brent and Zimmermann (2003), for example. In that case, we can decompose $P(z) = P_1(z)\cdots P_C(z)$, where the $P_c(z)$ are irreducible, and also decompose the matrix $\mathbf{A}$ in its *Jordan normal form*: $\mathbf{A} = \mathbf{P}\tilde{\mathbf{A}}\mathbf{P}^{-1}$, where $\mathbf{P}$ is an invertible matrix and

$\tilde{\mathbf{A}} = \mathrm{diag}(\tilde{\mathbf{A}}_1, \ldots, \tilde{\mathbf{A}}_C)$ is a block-diagonal matrix for which each block $\tilde{\mathbf{A}}_c$ has irreducible characteristic polynomial $P_c(z)$ (Golub and Loan 1996, Strang 1988). Once we have this decomposition, we know that the generator is equivalent to a combined RNG with transition matrix $\tilde{\mathbf{A}}$ and output transformation matrix $\tilde{\mathbf{B}} = \mathbf{BP}$, and we can analyze it in the same way as if we had first selected its components and then combined them. It is important to note that the purpose of the decomposition in this case is not to provide an efficient implementation for the combined generator, nor an efficient algorithm to jump ahead, but only to analyze the periodicity and other theoretical properties of the generator.

3 Quality Criteria

In general, good RNGs must have a long period ρ (say, $\rho \approx 2^{200}$ or more), must run fast, should not waste memory (the state should be represented in no more than roughly $\log_2 \rho$ bits of memory), must be repeatable and portable (able to reproduce exactly the same sequence in different software/hardware environments), and must allow efficient jumping-ahead in order to obtain multiple streams and substreams. But these required properties do not suffice to imitate independent random numbers.

Recall that a sequence of random variables $U_0, U_1, U_2, \ldots$ are i.i.d. $U[0,1)$ if and only if for all integers $i \geq 0$ and $t > 0$, the vector $(U_i, \ldots, U_{i+t-1})$ is uniformly distributed over the t-dimensional unit hypercube $[0,1)^t$. Of course, this cannot hold for algorithmic RNGs that have a finite period. For RNGs that fit our $\mathbb{F}_2$-linear framework, any vector of t successive output values of the generator belongs to the finite set

$$\Psi_t = \{(u_0, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\},$$

i.e., the set of output points obtained when the initial state runs over all possible k-bit vectors. This set Ψ_t always has cardinality 2^k when viewed as a multiset (i.e., if the points are counted as many times as they appear).

If $\mathbf{x}_0$ is drawn at random from the set of k-bit vectors $\mathbb{F}_2^k$, with probability 2^{-k} for each vector, then $(u_0, \ldots, u_{t-1})$ is a random vector having the uniform distribution over Ψ_t. Thus, to approximate well the uniform distribution over $[0,1)^t$, Ψ_t must cover the hypercube $[0,1)^t$ very uniformly (L'Ecuyer 1994, L'Ecuyer 2006). More generally, we may also want to measure the uniformity of sets of the form

$$\Psi_I = \{(u_{i_1}, \ldots, u_{i_t}) \mid \mathbf{x}_0 \in \mathbb{F}_2^k\},$$

where $I = \{i_1, \cdots, i_t\}$ is a fixed ordered set of non-negative integers such that $0 \leq i_1 < \cdots < i_t$. For $I = \{0, \ldots, t-1\}$, we recover $\Psi_t = \Psi_I$

The uniformity of Ψ_I is usually assessed by measures of *discrepancy* between the empirical distribution of its points and the uniform distribution over $[0,1)^t$ (Hellekalek and Larcher 1998, L'Ecuyer and Lemieux 2002, Niederreiter 1992). These measures can be defined in many ways and they are in fact equivalent to goodness-of-fit tests for the multivariate uniform distribution. They must be computable *without enumerating the points*, because the cardinality of Ψ_t makes the enumeration practically infeasible when the period is large enough. For this reason, the uniformity measures are usually tailored to the general structure of the RNG. Measures that are commonly used for $\mathbb{F}_2$-linear RNGs will be described in a moment. The selected discrepancy measure can be computed for each set I in some predefined class $\mathcal{J}$, then these values can be weighted or normalized by factors that may depend on I, and the worst-case (or average) over $\mathcal{J}$ can be adopted as a *figure of merit* used to rank RNGs. The choice of $\mathcal{J}$ and of the weights are arbitrary. They are a question of compromise and practicality. Typically, $\mathcal{J}$ would contain sets I such that t and $i_t - i_1$ are rather small. We generally try to optimize this figure of merit when searching (by computer) for concrete RNG parameters, within a given class of constructions.

For $\mathbb{F}_2$-linear generators, the uniformity of the point sets Ψ_I is typically assessed by measures of equidistribution defined as follows (L'Ecuyer 1996, L'Ecuyer and Panneton 2002, L'Ecuyer 2004, Tezuka 1995). For an arbitrary vector $\mathbf{q} = (q_1, \ldots, q_t)$ of non-negative integers, partition the unit hypercube $[0,1)^t$ into 2^{q_j} intervals of the same length along axis j, for each j. This determines a partition of $[0,1)^t$ into $2^{q_1+\cdots+q_t}$ rectangular boxes of the same size and shape. If a given set Ψ_I has exactly 2^q points in each box of this partition, where the integer q must satisfy $k - q = q_1 + \cdots + q_t$, we say that Ψ_I is $\mathbf{q}$-*equidistributed*. This means that among the 2^k points $(u_{i_1}, \ldots, u_{i_t})$ of Ψ_I, if we consider all $(k-q)$-bit vectors formed by the q_j most significant bits of u_{i_j} for $j = 1, \ldots, t$, each of the 2^{k-q} possibilities occurs exactly the same number of times. Of course, this is possible only if $q_1 + \cdots + q_t \le k$. When $q_1 + \cdots + q_t \ge k$, i.e., when the number of boxes is larger or equal to the number of points, we say that Ψ_I is $\mathbf{q}$-*collision-free* (CF) if no box contains more than one point (L'Ecuyer 1996).

If Ψ_I is $(\ell, \ldots, \ell)$-equidistributed for some $\ell \ge 1$, it is called *t-distributed with ℓ bits of accuracy*, or *(t,ℓ)-equidistributed* (L'Ecuyer 1996) (we will avoid this last notation because it conflicts with that for $(q_1, \ldots, q_t)$-equidistribution). The largest value of ℓ for which this holds is called the *resolution* of the set Ψ_I and is denoted by ℓ_I. It cannot exceed $\ell_t^* = \min(\lfloor k/t \rfloor, w)$. We define the *resolution gap* of Ψ_I as $\delta_I = \ell_t^* - \ell_I$. Potential figures of merit can then be defined by

$$\Delta_{\mathcal{J},\infty} = \max_{I \in \mathcal{J}} \omega_I \delta_I \quad \text{and} \quad \Delta_{\mathcal{J},1} = \sum_{I \in \mathcal{J}} \omega_I \delta_I$$

for some non-negative weights ω_I, where $\mathcal{J}$ is a preselected class of index sets I. The weights are often taken all equal to 1.

We also denote by t_ℓ the largest dimension t for which Ψ_t is t-distributed with ℓ bits of accuracy, and we define the *dimension gap* for ℓ bits of accuracy as

$$\tilde{\delta}_\ell = t_\ell^* - t_\ell,$$

where $t_\ell^* = \lfloor k/\ell \rfloor$ is an upper bound on t_ℓ. We may then consider the *worst-case weighted dimension gap* and the *weighted sum of dimension gaps*, defined as

$$\tilde{\Delta}_\infty = \max_{1\le\ell\le w} \omega_\ell\tilde{\delta}_\ell \quad \text{and} \quad \tilde{\Delta}_1 = \sum_{\ell=1}^{w} \omega_\ell\tilde{\delta}_\ell$$

for some non-negative weights ω_ℓ, as alternative figures of merit for our generators. Often, the weights are all 1 and the word "weighted" is removed from these definitions.

When $\tilde{\Delta}_\infty = \tilde{\Delta}_1 = 0$, the RNG is said to be *maximally equidistributed* (ME) or *asymptotically random* for the word size w (L'Ecuyer 1996, Tezuka 1995, Tootill, Robinson, and Eagle 1973). This property ensures perfect equidistribution of all sets Ψ_t, for any partition of the unit hypercube into subcubes of equal sizes, as long as $\ell \le w$ and the number of subcubes does not exceed the number of points in Ψ_t. As an additional requirement, we may ask that Ψ_t be $(\ell,\dots,\ell)$-collision-free whenever $t\ell \ge k$. Then we say that the RNG is *collision-free* (CF) (L'Ecuyer 1999b). Large-period ME (or almost ME) and ME-CF generators can be found in L'Ecuyer (1999b), L'Ecuyer and Panneton (2002), Panneton and L'Ecuyer (2004), and Panneton, L'Ecuyer, and Matsumoto (2006), for example.

The $(k-q)$-bit vectors involved in assessing the $\mathbf{q}$-equidistribution of Ψ_I can be expressed as a linear function of the k-bit initial state $\mathbf{x}_0$, that is, as $\mathbf{z}_0 = \mathbf{M_q}\mathbf{x}_0$ for some $(k-q)\times k$ binary matrix $\mathbf{M_q}$. Clearly, Ψ_I is $\mathbf{q}$-equidistributed if and only if $\mathbf{M_q}$ has full rank. Thus, $\mathbf{q}$-equidistribution can easily be verified by constructing this matrix $\mathbf{M_q}$ and checking its rank via (binary) Gaussian elimination (Fushimi 1983, L'Ecuyer 1996, Tezuka 1995). This is a major motivation for adopting this measure of uniformity.

To construct the matrix $\mathbf{M_q}$ that corresponds to Ψ_I, one can proceed as follows. For $j \in \{1,\dots,k\}$, start the generator in initial state $\mathbf{x}_0 = \mathbf{e}_j$, where $\mathbf{e}_j$ is the unit vector with a 1 in position j and zeros elsewhere, and run the generator for i_t steps. Record the q_1 most significant bits of the output at step i_1, the q_2 most significant bits of the output at step i_2, …, and the q_t most significant bits of the output at step i_t. These bits form the jth column of the matrix $\mathbf{M_q}$.

In the case of a combined generator as in Section 2.3, the matrix $\mathbf{M_q}$ can be constructed by first constructing the corresponding matrices $\mathbf{M_q}^{(c)}$ for the individual components, and simply juxtaposing these matrices, as suggested in L'Ecuyer (1999b). To describe how this is done, let us denote by $\Psi_I^{(c)}$ the point set that corresponds to component c alone, and let $\mathbf{x}_0^{\mathsf{t}} = ((\mathbf{x}_0^{(1)})^{\mathsf{t}},\dots,(\mathbf{x}_0^{(C)})^{\mathsf{t}})$ where $\mathbf{x}_0^{(c)}$ is the initial state for component c. If $\mathbf{z}_0^{(c)}$ is the $(k-q)$-bit vector relevant for the $\mathbf{q}$-equidistribution of $\Psi_I^{(c)}$, then we have $\mathbf{z}_0^{(c)} = \mathbf{M_q}^{(c)}\mathbf{x}_0^{(c)}$ for some $(k-q)\times k_c$ binary matrix $\mathbf{M_q}^{(c)}$ that can be constructed as explained earlier. Note that the point set Ψ_I can be written as the direct sum

$$\Psi_I = \Psi_I^{(1)} \oplus \cdots \oplus \Psi_I^{(C)} = \{\mathbf{u} = \mathbf{u}^{(1)} \oplus \cdots \oplus \mathbf{u}^{(C)} \mid \mathbf{u}^{(c)} \in \Phi_I^{(c)} \text{ for each } c\},$$

where $\oplus$ denotes the bitwise sum of binary expansions, coordinate by coordinate, and observe that

$$\mathbf{z}_0 = \mathbf{z}_0^{(1)} \oplus \cdots \oplus \mathbf{z}_0^{(C)} = \mathbf{M}_{\mathbf{q}}^{(1)}\mathbf{x}_0^{(1)} \oplus \cdots \oplus \mathbf{M}_{\mathbf{q}}^{(C)}\mathbf{x}_0^{(C)}.$$

This means that $\mathbf{M}_{\mathbf{q}}$ is just the juxtaposition $\mathbf{M}_{\mathbf{q}} = \mathbf{M}_{\mathbf{q}}^{(1)} \cdots \mathbf{M}_{\mathbf{q}}^{(C)}$. That is, $\mathbf{M}_{\mathbf{q}}^{(1)}$ gives the first k_1 columns of $\mathbf{M}_{\mathbf{q}}$, $\mathbf{M}_{\mathbf{q}}^{(2)}$ gives the next k_2 columns, and so on.

For very large values of k, the matrix $\mathbf{M}_{\mathbf{q}}$ is expensive to construct and reduce, but a more efficient method based on the computation of the shortest nonzero vector in a lattice of formal series, studied in Couture and L'Ecuyer (2000), can be used in that case to verify $(\ell, \ldots, \ell)$-equidistribution; see Section 4.

The figures of merit defined above look at the *most significant bits* of the output values, but give little importance to the least significant bits. We could of course extend them so that they also measure the equidistribution of the least significant bits, simply by using different bits to construct the output values and computing the corresponding $\mathbf{q}$-equidistributions. But this becomes quite cumbersome and expensive to compute in general because there are too many ways of selecting which bits are to be considered. Certain classes of $\mathbb{F}_2$-linear generators (the Tausworthe/LFSR RNGs defined in Subsection 5.1) have the interesting property that if all output values are multiplied by a given power of two, modulo 1, all equidistribution properties remain unchanged. In other words, they enjoy the nice property that their least significant bits have the same equidistribution as the most significant ones. We call such generators *resolution-stationary* (Panneton and L'Ecuyer 2005b).

Aside from excellent equidistribution, good $\mathbb{F}_2$-linear generators are also required to have characteristic polynomials $P(z)$ whose number N_1 of nonzero coefficients is not too far from half the degree, i.e., near $k/2$ (Compagner 1991, Wang and Compagner 1993). Intuitively, if N_1 is very small and if the state $\mathbf{x}_n$ happens to contain many 0's and only a few 1's, then there is a high likelihood that the $N_1 - 1$ bits used to determine any given new bit of the next state are all zero, in which case this new bit will also be zero. In other words, it may happen frequently in that case that only a small percentage of the bits of $\mathbf{x}_n$ are modified from one step to the next, so the state can contain much more 0's than 1's for a large number of steps. Then, in the terminology of cryptologists, the recurrence has *low diffusion capacity.* An illustration of this with the Mersenne twister can be found in Panneton, L'Ecuyer, and Matsumoto (2006). In particular, generators for which $P(z)$ is a trinomial or a pentanomial, which have been often used in the past, should be avoided. They fail rather simple statistical tests (Lindholm 1968, Matsumoto and Kurita 1996). The fraction N_1/k of nonzero coefficients in $P(z)$ can be used as a secondary figure of merit for an RNG.

Other measures of uniformity are popular in the context where k is small and the entire point set Ψ_t is used for quasi-Monte Carlo integration (Niederreiter 1992, Hellekalek and Larcher 1998, L'Ecuyer and Lemieux 2002); for example the smallest q for which Ψ_t is a (q, k, t)-net (commonly known as a (t, m, s)-net, using a different notation), the P_α measure and its weighted versions, the diaphony, etc. However, no one knows how to compute these measures efficiently when $k > 50$ (say), which is always the case for good $\mathbb{F}_2$-linear RNGs.

4 Lattice Structure in a Space of Formal Series

The lattice structure of linear congruential generators (LCGs) is well-known in the simulation community (Law and Kelton 2000, Knuth 1998). $\mathbb{F}_2$-linear RNGs do not have a lattice structure in the real space, but they do have a similar form of lattice structure in a space of formal series (Couture and L'Ecuyer 2000, L'Ecuyer 2004, Lemieux and L'Ecuyer 2003, Tezuka 1995), which we now outline. In comparison with the lattices of LCGs, the real space $\mathbb{R}$ is replaced by the space $\mathbb{L}_2$ of formal power series with coefficients in $\mathbb{F}_2$, of the form $\sum_{\ell=\omega}^{\infty} x_\ell z^{-\ell}$ for some integer ω, and the integers are replaced by polynomials over $\mathbb{F}_2$.

Some $\mathbb{F}_2$-linear RNGs (e.g., the LFSR generators) have a *dimension-wise* lattice structure where the lattice contains vectors of t-dimensional formal series, whose coordinate j is the generating function for the binary expansion of the jth output value, for a given initial state (Tezuka and L'Ecuyer 1991, L'Ecuyer 1994, Tezuka 1995, Lemieux and L'Ecuyer 2003). This dimension-wise lattice can be used to study equidistribution, but it only applies to a subclass of $\mathbb{F}_2$-linear RNGs. For this reason, we will not discuss it any further here. We will concentrate instead on the *resolution-wise* lattice introduced by Tezuka (1995), which applies to all $\mathbb{F}_2$-linear generators.

The sequence of values taken by the jth bit of the *output*, from a given initial state $\mathbf{x}_0$, has *generating function*

$$G_j(z) = y_{0,j}z^{-1} + y_{1,j}z^{-2} + \cdots = \sum_{n=1}^{\infty} y_{n-1,j}z^{-n}$$

(which depends on $\mathbf{x}_0$). When multiplying this formal series by $P(z)$, we obtain the polynomial $g_j(z) = G_j(z)P(z)$ in $\mathbb{F}_2[z]/P(z)$ (the space of polynomials of degree less than k, with coefficients in $\mathbb{F}_2$), because the successive terms of the series satisfy a recurrence with this characteristic polynomial. For $\ell = 1, \ldots, w$, let $\mathbf{G}^{(\ell)}(z) = (G_0(z), \ldots, G_{\ell-1}(z))$.

We first consider the case where $P(z)$ is an irreducible polynomial. In that case, if $G_0(z) \neq 0$, then $g_0(z)$ has an inverse modulo $P(z)$ and there is a unique initial state of the RNG that corresponds to the vector

$$\begin{aligned}\bar{\mathbf{G}}^{(\ell)}(z) &= g_0^{-1}(z)\mathbf{G}^{(\ell)}(z)\\ &= (1, g_0^{-1}(z)g_1(z), \ldots, g_0^{-1}(z)g_{\ell-1}(z))/P(z)\end{aligned}$$

(Panneton 2004), Lemma 3.2. Thus, if we rename momentarily $g_0^{-1}(z)g_j(z)$ as $g_j(z)$, we see that it is always possible to select the initial state of the RNG so that $g_0(z) = 1$, i.e,

$$\bar{\mathbf{G}}^{(\ell)}(z) = (1, g_1(z), \ldots, g_{\ell-1}(z))/P(z).$$

When $P(z)$ is irreducible, any given bit of the output follows the same recurrence, with minimal polynomial $P(z)$, but with a lag between the recurrences for the different bits, i.e., they have different starting points. The vector $\bar{\mathbf{G}}^{(\ell)}(z)$ tell us about these lags. More specifically, if $g_i(z) \equiv g_0(z)z^{t_i} \mod P(z)$, then the lag between the recurrences for bit 0 and bit i is t_i.

Let $\mathbb{L}_2 = \mathbb{F}_2((z^{-1}))$ be the space of formal series of the form $\sum_{n=i}^{\infty} d_{n-1}z^{-n}$ where $i \in \mathbb{Z}$ and $d_{n-1} \in \mathbb{F}_2$ for each n. Let $\mathbb{L}_{2,0}$ be those series for which $i \geq 1$. Suppose that the first ℓ rows of the matrix $\mathbf{B}$ are linearly independent. Then the vectors $\mathbf{v}_1(z) = \bar{\mathbf{G}}^{(\ell)}(z)$, $\mathbf{v}_2(z) = \mathbf{e}_2(z), \ldots, \mathbf{v}_\ell(z) = \mathbf{e}_\ell(z)$ form a basis of a *lattice* $\mathcal{L}_\ell$ in $\mathbb{L}_2$, defined by

$$\mathcal{L}_\ell = \left\{ \mathbf{v}(z) = \sum_{j=1}^{\ell} h_j(z)\mathbf{v}_j(z) \text{ such that } h_j(z) \in \mathbb{F}_2[z] \right\}.$$

This lattice is called the ℓ-bit *resolution-wise lattice* associated with the RNG. The matrix $\mathbf{V}$ whose rows are the $\mathbf{v}_j$'s has an inverse $\mathbf{W} = \mathbf{V}^{-1}$ whose columns

$$\begin{aligned} \mathbf{w}_1(z) &= (P(z), 0, \ldots, 0)^{\mathsf{t}}, \\ \mathbf{w}_2(z) &= (-g_1(z), 1, \ldots, 0)^{\mathsf{t}}, \\ &\cdots \\ \mathbf{w}_\ell(z) &= (-g_{\ell-1}(z), 0, \ldots, 1)^{\mathsf{t}} \end{aligned}$$

form a basis of the *dual lattice*

$$\mathcal{L}_\ell^* = \{\mathbf{h}(z) \in \mathbb{L}_2^\ell : \mathbf{h}(z) \cdot \mathbf{v}(z) \in \mathbb{F}_2[z] \text{ for all } \mathbf{v}(z) \in \mathcal{L}_\ell\},$$

where $\mathbf{h}(z) \cdot \mathbf{v}(z) = \sum_{j=1}^{\ell} h_j(z)v_j(z)$ (the scalar product). This resolution-wise lattice fully describes all the possible output sequences of the RNG via the following theorem. It says that the set of all vectors of generating function that we can get, from all possible initial states $\mathbf{x}_0$, is exactly the set of lattice points that belong to $\mathbb{L}_{2,0}$. (Here we do not assume that $g_0(z) = 1$.)

Theorem 2 *(Couture and L'Ecuyer 2000). We have*

$$\mathcal{L}_\ell \cap \mathbb{L}_{2,0} \quad = \quad \{(g_0(z), \ldots, g_{\ell-1}(z))/P(z) : \mathbf{x}_0 \in \mathbb{F}_2^k\}.$$

For any $\mathbf{h}(z) = (h_1(z), \ldots, h_\ell(z)) \in (\mathbb{F}_2[z])^\ell$, we define the *length* of $\mathbf{h}(z)$ by $\|\mathbf{0}\| = 0$ and

$$\log_2 \|\mathbf{h}(z)\| = \max_{1 \leq j \leq \ell} \deg h_j(z) \qquad \text{for } \mathbf{h}(z) \neq \mathbf{0}.$$

Theorem 3 *(Tezuka 1995, Couture and L'Ecuyer 2000). Ψ_t is t-distributed with ℓ bits of accuracy if and only if*

$$\min_{\mathbf{0} \neq \mathbf{h}(z) \in \mathcal{L}_\ell^*} \log_2 \|\mathbf{h}(z)\| > \ell.$$

This theorem shows that checking equidistribution amounts to computing a shortest nonzero vector in the dual lattice $\mathcal{L}_\ell^*$, just like the spectral test commonly applied to LCGs but with a different lattice. As it turns out, very similar algorithms can be used to compute the shortest vector in both cases (Couture and L'Ecuyer 2000). The algorithm of Lenstra (1985) computes a reduced lattice basis in the sense of Minkowski for a polynomial lattice; the first (shortest) vector of that reduced basis is a shortest nonzero vector in the lattice.

This approach is more efficient than applying Gaussian elimination to the matrix $\mathbf{M_q}$ (see Subsection 3) when t is large. However, it applies only to the point set Ψ_t formed by t successive output values, and not to the more general point sets Ψ_I.

To construct a basis of the dual lattice for all $\ell \le w$, we only need the polynomials $g_0(z), g_1(z), \ldots, g_{w-1}(z)$. These polynomials can be computed as follows. Start the generator in some arbitrary nonzero initial state $\mathbf{x}_0$, run it for $k-1$ steps, and observe the corresponding output bits $\mathbf{y}_n = (y_{n,0}, \ldots, y_{n,w-1})$, for $n = 0, \ldots, k-1$. This gives the first k coefficients of $G_j(z)$ for $j = 0, \ldots, w-1$. The coefficients of each $g_j(z) = \sum_{i=1}^{k} c_{j,i} z_{k-i}$ can then be obtained via (Lemieux and L'Ecuyer 2003, Proposition 3.6):

$$\begin{pmatrix} c_{j,1} \\ c_{j,2} \\ \vdots \\ c_{j,k} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ \alpha_1 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \alpha_{k-1} & \ldots & \alpha_1 & 1 \end{pmatrix} \begin{pmatrix} y_{0,j} \\ y_{1,j} \\ \vdots \\ y_{k-1,j} \end{pmatrix}.$$

Then, to obtain $g_0(z) = 1$, it suffices to compute the inverse of $g_0(z)$ modulo $P(z)$ and to multiply each $g_j(z)$ by this inverse.

When $P(z)$ is reducible, we can no longer use the argument that $g_0(z)$ has an inverse, but everything else still applies. Suppose $P(z) = P_1(z) \cdots P_C(z)$ where the $P_c(z)$'s are *distinct* irreducible polynomials; all interesting RNGs should satisfy this assumption, usually with a small value of C. In that case, the RNG can then be interpreted as a combined $\mathbb{F}_2$-linear generator that fits the framework of Section 2.3 and a basis of the dual lattice can be constructed by decomposition, as we now explain. If $\mathcal{L}_\ell^{(c)}$ denotes the resolution-wise lattice associated with component c alone and $\mathcal{L}_\ell^{(c)*}$ its dual, it can be seen easily that

$$\mathcal{L}_\ell = \mathcal{L}_\ell^{(1)} \oplus \cdots \oplus \mathcal{L}_\ell^{(C)}$$

(the direct sum of lattices) and

$$\mathcal{L}_\ell^* = \mathcal{L}_\ell^{(1)*} \cap \cdots \cap \mathcal{L}_\ell^{(C)*}.$$

To find a basis of this dual lattice, we can first compute a basis of the dual lattice $\mathcal{L}_\ell^{(c)*}$ for each component c, as described earlier. Let $-g_1^{(c)}(z), \ldots, -g_{\ell-1}^{(c)}(z)$ be the polynomials found in the first coordinates of these dual basis vectors (we assume that $g_0^{(c)}(z) = 1$). For each c, compute $Q_c(z) = (P(z)/P_c(z))^{-1} \bmod P_c(z)$; then for $j = 1, \ldots, \ell - 1$, compute

$$g_j(z) = \sum_{c=1}^{C} \left(g_j^{(c)}(z) Q_c(z) P(z)/P_j(z) \right) \bmod P(z),$$

so that $g_j(z) \equiv g_j^{(c)}(z) \bmod P(z)$ for each j. Then define $\mathbf{v}_1 = (1, g_2(z), \ldots, g_{\ell-1}(z))/P(z)$, $\mathbf{v}_j(z) = \mathbf{e}_j(z)$ for $j \geq 2$, $\mathbf{w}_1(z) = (P(z), 0, \ldots, 0)^{\mathsf{t}}$, and $\mathbf{w}_j(z) = \mathbf{e}_j - g_j(z)\mathbf{e}_1$ for $j \geq 2$, as before. Under the assumption that the $P_c(z)$'s are pairwise relatively prime, the proof of Proposition 4.13 of Lemieux and L'Ecuyer (2002) (an expanded version of Lemieux and L'Ecuyer 2003) implies the following result:

Proposition 4 *The vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_\ell$ *form a basis of* $\mathcal{L}_\ell$ *and* $\mathbf{w}_1, \ldots, \mathbf{w}_\ell$ *are a basis of the dual lattice* $\mathcal{L}_\ell^*$.

This way of doing most of the computations for the components separately before putting the results together is more efficient than working directly with the combined generator, especially if the components are much smaller than the combination.

5 Specific Classes of Generators

5.1 The LFSR Generator

The *Tausworthe* or *linear feedback shift register* (LFSR) generator (Tausworthe 1965, L'Ecuyer 1996, Tezuka 1995) is defined by a linear recurrence modulo 2, from which a block of w bits is taken every s steps, for some positive integers w and s:

$$\begin{aligned} x_n &= a_1 x_{n-1} + \cdots + a_k x_{n-k}, & (6) \\ u_n &= \sum_{\ell=1}^{w} x_{ns+\ell-1} 2^{-\ell}. & (7) \end{aligned}$$

where $a_1, \ldots, a_k$ are in $\mathbb{F}_2$ and $a_k = 1$. This fits our framework by taking $\mathbf{A} = (\mathbf{A}_0)^s$ (in $\mathbb{F}_2$) where

$$\mathbf{A}_0 = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_k & a_{k-1} & \ldots & a_1 \end{pmatrix}, \tag{8}$$

and blank entries in this matrix are zeros (we use that convention all along this paper). If $w \leq k$, the matrix $\mathbf{B}$ would contain the first w rows of the $k \times k$ identity matrix. However, we may also have $w > k$, in particular when implementing an LFSR used as a component of a combined generator. In that case, it is convenient to expand $\mathbf{A}$ into a $w \times w$ matrix with the same minimal polynomial (of degree k), as follows: For $j = 1, \ldots, w-k$, add the row $(a_1^{(j)}, \ldots, a_k^{(j)})$, where the coefficients $a_i^{(j)}$ are such that $x_{n+j} = a_1^{(j)} x_{n-1} + \cdots + a_k^{(j)} x_{n-k}$. This can be done in the same way as when we build the matrix $\mathbf{M_q}$ in Section 3. Then, we add $w-k$ columns of zeros.

Note that $P(z)$ is the characteristic polynomial of the matrix $\mathbf{A} = (\mathbf{A}_0)^s$, not that of the recurrence (6), and the choice of s is important for determining the quality of this generator. A frequently encountered case is when a single a_j is nonzero in addition to a_k; then, the characteristic polynomial of $\mathbf{A}_0$ is a trinomial and we have a *trinomial-based* LFSR generator. Typically, s is small to make the implementation efficient. These trinomial-based generators are known to have important statistical weaknesses (Matsumoto and Kurita 1996, Tezuka 1995) but they can be used a components of combined RNGs (Tezuka and L'Ecuyer 1991, Wang and Compagner 1993, L'Ecuyer 1996). They also enjoy the important properties of being resolution-stationary (Panneton and L'Ecuyer 2005b). Tables of specific parameters for maximally equidistributed combined LFSR generators, together with concrete implementations for 32-bit and 64-bit computers, can be found in L'Ecuyer (1999b). These generators are among the fastest ones currently available.

To show how an LFSR generator can be implemented efficiently, we outline an algorithm for the following situation. Suppose that $a_j = 1$ for $j \in \{j_1, \ldots, j_d\}$ and $a_j = 0$ otherwise, with $k/2 \le j_1 < \cdots < j_d = k \le w$ and $0 < s \le j_1$. We work directly with the w-bit vectors $\mathbf{y}_n = (x_{ns}, \ldots, x_{ns+w-1})$, assuming that w is the computer's word length. Under these conditions, a left shift of $\mathbf{y}_n$ by $k - j_i$ bits, denoted $\mathbf{y}_n \ll (k - j_i)$, gives a vector that contains the first $w - k + j_i$ bits of $\mathbf{y}_{n+k-j_i}$ followed by $k - j_i$ zeros (for $i = d$, $j_i = k$ so there is no shift). Adding these d shifted vectors by a bitwise xor, for $j = 1, \ldots, d$, gives a vector $\tilde{\mathbf{y}}$ that contains the first $w - k + j_1$ bits of $\mathbf{y}_{n+k} = \mathbf{y}_{n+k-j_1} \oplus \cdots \oplus \mathbf{y}_{n+k-j_d}$ followed by $k - j_1$ other bits (which do not matter). Now we shift $\tilde{\mathbf{y}}$ by $k - s$ positions to the right, denoted $\tilde{\mathbf{y}} \gg (k - s)$; this gives $k - s$ zeros followed by the last $w - k + s$ bits of $\mathbf{y}_{n+s}$ (the $k - j_1$ bits that do not matter have disappeared, because $s \ge j_1$). Zeroing the last $w - k$ bits of $\mathbf{y}_n$ and then shifting it to the left by s bits gives the first $k - s$ bits of $\mathbf{y}_{n+s}$. Adding this to $\tilde{\mathbf{y}}$ then gives $\mathbf{y}_{n+s}$. This is summarized by the following algorithm, in which & denotes a bitwise "and" and `mask` contains k 1's followed by $w - k$ 0's.

```
Algorithm L [One step of a simple LFSR generator]:
$\tilde{\mathbf{y}} = \mathbf{y}_n$;
For $i = 2, \ldots, d$,  $\tilde{\mathbf{y}} = \tilde{\mathbf{y}} \oplus (\mathbf{y}_n \ll (k - j_i))$;
$\mathbf{y}_{n+s} = (\tilde{\mathbf{y}} \gg (k - s)) \oplus ((\mathbf{y}_n \& \texttt{mask}) \ll s)$;
```

For this to work properly, we must make sure that $\mathbf{y}_0$ is initialized to a valid state, i.e., that the values $x_k, \ldots, x_{w-1}$ satisfy the recurrence $x_j = a_1 x_{j-1} + \cdots + a_k x_{j-k}$ for $j = k, \ldots, w-1$. We can take $(x_0, \ldots, x_{k-1})$ as an arbitrary nonzero vector, and then simply compute $x_k, \ldots, x_{w-1}$ from the recurrence. L'Ecuyer (1996) explains how to implement this.

5.2 The GFSR, Twisted GFSR, and Mersenne Twister

Here we suppose that $\mathbf{A}$ is a $pq \times pq$ matrix with the general form

$$\mathbf{A} = \begin{pmatrix} \mathbf{S}_1 & \mathbf{S}_2 & & \mathbf{S}_{q-1} & \mathbf{S}_q \\ \mathbf{I}_p & & & & \\ & \mathbf{I}_p & & & \\ & & \ddots & & \\ & & & \mathbf{I}_p & \end{pmatrix}$$

for some positive integers p and q, where $\mathbf{I}_p$ is the $p \times p$ identity matrix, and each $\mathbf{S}_j$ is a $p \times p$ matrix. Often, $w = p$ and $\mathbf{B}$ contains the first w rows of the $pq \times pq$ identity matrix. If $\mathbf{S}_r = \mathbf{S}_q = \mathbf{I}_p$ for some r and all the other $\mathbf{S}_j$'s are zero, this generator is the trinomial-based *generalized feedback shift register* (GFSR), for which $\mathbf{x}_n$ is obtained by a bitwise exclusive-or of $\mathbf{x}_{n-r}$ and $\mathbf{x}_{n-q}$ and where $\mathbf{x}_n$ gives the w bits of u_n (Lewis and Payne 1973). This provides an extremely fast RNG. However, its period cannot exceed $2^q - 1$, because each bit of $\mathbf{x}_n$ follows the same binary recurrence of order $k = q$, with characteristic polynomial $P(z) = z^q - z^{q-r} - 1$.

More generally, we can define $\mathbf{x}_n$ as the bitwise exclusive-or of $\mathbf{x}_{n-r_1}, \mathbf{x}_{n-r_2}, \ldots, \mathbf{x}_{n-r_d}$ where $r_d = q$, so that each bit of $\mathbf{x}_n$ follows a recurrence in $\mathbb{F}_2$ whose characteristic polynomial $P(z)$ has $d+1$ nonzero terms. This corresponds to taking $\mathbf{S}_j = \mathbf{I}_p$ for $j \in \{r_1, \ldots, r_d\}$ and $\mathbf{S}_j = \mathbf{0}$ otherwise. However, the period is still bounded by $2^q - 1$, whereas considering the pq-bit state, we should expect a period close to 2^{pq}. This was the main motivation for the *twisted GFSR* (TGFSR) generator. In the original version introduced by Matsumoto and Kurita (1992), $w = p$, $\mathbf{S}_q$ is defined as the transpose of $\mathbf{A}_0$ in (8) with k replaced by p, $\mathbf{S}_r = \mathbf{I}_p$, and all the other $\mathbf{S}_j$'s are zero. The characteristic polynomial of $\mathbf{A}$ is then $P(z) = P_S(z^q + z^{q-r})$, where $P_S(\zeta) = \zeta^p - a_p\zeta^{p-1} - \cdots - a_1$ is the characteristic polynomial of $\mathbf{S}_q$, and its degree is $k = pq$. If the parameters are selected so that $P(z)$ is primitive over $\mathbb{F}_2$, then the TGFSR has period $2^k - 1$. Matsumoto and Kurita (1994) pointed out important weaknesses of the original TGFSR, for which $\mathbf{B}$ contains the first rows of the identity matrix, and introduced an improved version that uses a well-chosen matrix $\mathbf{B}$ whose rows differ from those of the identity. The operations implemented by this matrix are called *tempering* and their purpose is to improve the uniformity of the points produced by the RNG. To our knowledge, this was the first proposal of $\mathbb{F}_2$-linear RNG with a $\mathbf{B}$ that differs from the truncated identity.

The *Mersenne twister* (Matsumoto and Nishimura 1998, Nishimura 2000) (MT) is a variant of the TGFSR where k is slightly less than pq and can be a prime number. It uses a pq-bit vector to store the k-bit state, where $k = pq - r$ is selected so that $r < p$ and $2^k - 1$ is a Mersenne prime. The matrix $\mathbf{A}$ is a $(pq - r) \times (pq - r)$ matrix similar to that of the TGFSR and the implementation is also quite similar. The main reason for using a k of that form is to simplify the search for primitive characteristic polynomials (see Algorithm P). If we take $k = pq$, then we know that we cannot have a Mersenne prime because $2^{pq} - 1$

is divisible by $2^p - 1$ and $2^q - 1$. A specific instance proposed by Matsumoto and Nishimura (1998), and named `MT19937`, has become quite popular; it is fast and has the huge period of $2^{19937} - 1$.

A weakness of this RNG is underlined and illustrated in Panneton, L'Ecuyer, and Matsumoto (2006): if the generator starts in (or reaches) a state that has very few ones, it may take up to several hundred thousands steps before the ratio of ones in the output and/or the average output value are approximately 1/2. For example, for `MT19937`, if we average the output values at steps $n + 1$ to $n + 100$ (a moving average) and average this over all 19937 initial states $\mathbf{x}_0$ that have a single bit at one, then we need at least $n > 700,000$ before the average gets close to 1/2 as it should be (this is graphically illustrated in Panneton, L'Ecuyer, and Matsumoto 2006). Likewise, if two states differ by a single bit, or by only a few bits, a very large number of steps are required on average before the states or the outputs differ by about half of their bits. The source of the problem is that this RNG has a (huge) 19937-bit state and few of these bits are modified from one step to the next, as explained near the end of Section 3; it has only $N_1 = 135$ nonzero coefficients out of 19938 in its characteristic polynomial. Moreover, the figure of merit $\tilde{\Delta}_1$ takes the large value 6750 for this generator.

It has been proved that the TGFSR and Mersenne twister construction methods used in Matsumoto and Kurita (1994), Matsumoto and Nishimura (1998) *cannot* provide ME generators in general. They typically have large equidistribution gaps. But combining them via a bitwise xor *can* yield generators with the ME property. Concrete examples of ME combined TGFSR generators with periods around 2^{466} and 2^{1250} are given in L'Ecuyer and Panneton (2002). These generators have the additional property that the resolution gaps δ_I are also zero for a class of index sets I of small cardinality and whose elements are not too far apart. These combined RNGs are of course somewhat slower than their original (uncombined) counterparts.

5.3 The WELL RNGs

These RNGs were developed by Panneton (2004) and are described by Panneton, L'Ecuyer, and Matsumoto (2006). The idea was to "sprinkle" a small number of very simple operations on w-bit words (where w is taken as the size of the computer word), such as xor, shift, bit mask, etc., into the matrix $\mathbf{A}$ in a way that the resulting RNG has maximal period and runs about as fast as the Mersenne twister, but also has (under these constraints) the best possible equidistribution properties, and a characteristic polynomial with around 50% nonzero coefficients.

The state $\mathbf{x}_n = (\mathbf{v}_{n,0}^{\mathsf{t}}, \ldots, \mathbf{v}_{n,r-1}^{\mathsf{t}})^{\mathsf{t}}$ is comprised of r blocks of $w = 32$ bits $\mathbf{v}_{n,j}$, and the recurrence is defined by a set of linear transformations that apply to these blocks, as described in Panneton, L'Ecuyer, and Matsumoto (2006). Essentially, the transformations modify $\mathbf{v}_{n,0}$ and $\mathbf{v}_{n,1}$ by using several of the other blocks. They are selected so that $P(z)$, a polynomial of degree $k = rw - p$, is primitive over $\mathbb{F}_2$. The output is defined by $\mathbf{y}_n = \mathbf{v}_{n,0}$.

The authors list specific parameters for WELL generators with periods ranging from $2^{512}-1$ to $2^{44497}-1$. Many of them are ME and the others are nearly ME. Their characteristic polynomials have nearly 50% coefficients equal to 1. These RNGs have much better diffusion capacity than the Mersenne twister and have comparable speed.

5.4 Xorshift Generators

Marsaglia (2003) has proposed a class of very fast RNGs whose recurrence can be implemented by a small number of xorshift operations only, where a *xorshift operation* consists in replacing a w-bit block in the state by a (left or right) shifted version of itself (by a positions, where $0 < a < w$) xored with the original block. The constant w is the computer's word size (usually 32 or 64). The specific generators he proposed in his paper use three xorshift operations at each step. As it turns out, xorshifts are linear operations so these generators fit our $\mathbb{F}_2$-linear setting.

Panneton and L'Ecuyer (2005a) analyzed the theoretical properties of a general class of xorshift generators that contains those proposed by Marsaglia. They studied maximal-period conditions, limits on the equidistribution, and submitted xorshift generators to empirical statistical testing. They concluded that three-xorshift generators are unsafe and came up with generators based on 7 and 13 xorshifts, whose speed is only 20% slower than those with three xorshifts to generate $U(0,1)$ real numbers. Aside from the tests that detect $\mathbb{F}_2$-linearity, these RNGs pass other standard statistical tests.

Brent (2004) has proposed a family of generators that combine a xorshift RNG with a Weyl generator. The resulting generator is no longer $\mathbb{F}_2$-linear and it behaves well empirically (L'Ecuyer and Simard 2006).

5.5 Linear Recurrences in $\mathbb{F}_{2^w}$

Fix a positive integer w (e.g., $w = 32$) and let $q = 2^w$. Panneton (2004) and Panneton and L'Ecuyer (2004) consider fast RNGs based on recurrences in the finite field $\mathbb{F}_q$, which can be written as

$$m_n \quad = \quad b_1 m_{n-1} + \cdots + b_r m_{n-r}$$

for some integer r, where the arithmetic is performed in $\mathbb{F}_q$. The maximal period $\rho = 2^{rw}-1$ is reached if and only if $\tilde{P}(z) = z^r - b_1 z^{r-1} - \cdots - b_{r-1}z - b_r$ is a primitive polynomial over $\mathbb{F}_q$.

To implement this recurrence, these authors select an algebraic element ζ of $\mathbb{F}_q$, take $\{1, \zeta, \ldots, \zeta^{r-1}\}$ as a basis of $\mathbb{F}_q$ over $\mathbb{F}_2$, and represent the elements $m_n = v_{n,0} + v_{n,1}\zeta + \cdots + v_{n,w-1}\zeta^{w-1}$ of $\mathbb{F}_q$ by the bit vectors $\mathbf{v}_n = (v_{n,0}, v_{n,1}, \ldots, v_{n,w-1})^{\mathsf{t}}$. The state of the RNG is thus represented by a rw-bit vector and the output is constructed as in (3), from the bits of $\mathbf{v}_n$. (More generally, one could define the output by taking $\mathbf{y}_n = (\mathbf{v}_n, \mathbf{v}_{n-1}, \ldots, \mathbf{v}_{n-r+1})$

for some $r \geq 1$.) This construction fits our $\mathbb{F}_2$-linear framework (1–3) and generalizes the TGFSR generators. Panneton and L'Ecuyer (2004) call them *LFSR generators in* $\mathbb{F}_{2^w}$.

The same authors also propose a slightly different construction called *polynomial LCG in* $\mathbb{F}_{2^w}$, and based on the recurrence

$$q_n(z) \quad = \quad zq_{n-1}(z) \mod \tilde{P}(z)$$

in $\mathbb{F}_q[z]$ (the ring of polynomials with coefficients in $\mathbb{F}_q$), where $\tilde{P}(z) \in \mathbb{F}_q[z]$ is a primitive polynomial. To implement this, each coefficient of $q_n(z)$ is represented by a w-bit vector just as for m_n and the output is defined in a similar way. Again, this fits the $\mathbb{F}_2$-linear framework (1–3).

Panneton (2004) (see also Panneton and L'Ecuyer 2005a) goes further by proving certain properties of the equidistribution of these RNGs. For instance, he shows that if $\tilde{P}(z)$ is irreducible over $\mathbb{F}_q$ and can be written as

$$\tilde{P}(z) = p_0(z) + \zeta p_1(z) + \cdots + \zeta^\gamma p_\gamma(z)$$

where each $p_i(z)$ is in $\mathbb{F}_2[z]$, then the RNG cannot be t-distributed with ℓ bits of accuracy if $t > r$ and $\ell > \gamma$. As a corollary, since the TGFSR has $\tilde{P}(z) = p_0(z) + \zeta p_1(z)$, it cannot be t-distributed with more than a single bit of accuracy in any dimension $t > r$. He also shows that if $\tilde{P}(z)$ is irreducible over $\mathbb{F}_q$ and has at least three nonzero coefficients, then among the $2^{rw} - 1$ two-dimensional point sets $\Psi_{\{0,j\}}$ where $1 \leq j < 2^{kw}$, exactly $2^w - 1$ are not 2-distributed with w bits of accuracy. For example, if $w = 32$ and $r = 25$ (so $k = 800$), only one two-dimensional projection out of 2^{768} is not equidistributed!

Panneton (2004) and Panneton and L'Ecuyer (2004) propose tables of good parameters for LFSRs and polynomial LCGs in $\mathbb{F}_q$. These parameters were found by computer searches based on the figure of merit $\tilde{\Delta}_1$. They also provide concrete implementations in the C language. These implementations are fast, comparable to the Mersenne twister for instance, but one drawback is that they use precomputed multiplication tables that require a non-negligible amount of memory. (In the case of multiple streams, a single copy of the tables is shared by all the streams.) The output transformation by a non-trivial matrix $\mathbf{B}$ is integrated into these multiplication tables to improve the efficiency.

6 Speed and Performance in Statistical Tests

6.1 Speed Comparisons

Table 1 reports the speed of some RNGs available in the Java-based SSJ simulation package (L'Ecuyer and Buist 2005). The timings are for the SSJ implementations on a 2.4 GHz 64-bit AMD-Athlon computer with SUN's JDK 1.5, C implementations on the same processor, and C implementations running on a 2.8 GHz 32-bit Intel processor. The first and second

Table 1: CPU time (sec) to generate 10^9 random numbers, and CPU time to jump ahead 10^6 times, with some RNGs available in SSJ

RNG	$\rho \approx$	CPU time in SSJ (Java)		CPU time in C	
		gen.	jump	gen. 64	gen. 32
LFSR113	2^{113}	31	0.1	10	39
LFSR258	2^{258}	35	0.2	12	58
WELL512	2^{512}	33	234	12	38
WELL1024	2^{1024}	34	917	11	37
MT19937	2^{19937}	36	—	16	42
MRG31k3p	2^{185}	51	0.9	21	71
MRG32k3a	2^{191}	70	1.1	21	99

columns of the table give the generator's name and its approximate period. All these generators are implemented for a 32-bit computer, although the C implementation of the two MRG generators (last two lines) used on the 64-bit computer was different; it exploits the 64-bit arithmetic. The SSJ implementations of all generators have more overhead because they support multiple streams, can generate either integers or real numbers, etc. The jumping ahead in SSJ is implemented via a multiplication by $\mathbf{A}^{\nu}$ as explained in Section 2.2. For the combined LFSR generators, the linear recurrence that corresponds to the matrix $\mathbf{A}^{\nu}$ is implemented directly using the algorithm of Section 5.1, for each component of the combination. It is much faster for this reason. Column 3 of the table gives the CPU time (sec) to generate 10^9 random numbers and add them up, whereas column 4 gives the CPU time needed to jump ahead 10^6 times by a very large number of steps (to get a new stream), in SSJ. For comparison, columns 5 and 6 give the time to generate 10^9 numbers with the C implementation available in TestU01 (L'Ecuyer and Simard 2006), first on the same computer (gen. 64), and then on the 32-bit computer (gen. 32).

The first five RNGs are $\mathbb{F}_2$-linear and the last two are combined multiple recursive generators (MRGs). The first two are combined LFSRs proposed by L'Ecuyer (1999b) for 32-bit and 64-bit computers, with four and five components, respectively. The two WELL RNGs are proposed in Panneton, L'Ecuyer, and Matsumoto (2006). Other WELL generators with much longer periods (up to nearly 2^{44497}) proposed in that paper have approximately the same speed as those given here to generate random numbers, but are much slower than WELL1024 for jumping ahead because of their larger value of k. For the Mersenne twister `MT19937`, proposed by Matsumoto and Nishimura (1998), jumping ahead is also too slow and is not implemented in SSJ. All these $\mathbb{F}_2$-linear RNGs have roughly the same speed for generating random numbers. Other ones with about the same speed are also proposed by Panneton and L'Ecuyer (2004) and Matsumoto and Kurita (1994), e.g., with periods near 2^{800}.

The timings of the two MRGs in the table are reported for comparison. The first one (`MRG31k3p`) was proposed by L'Ecuyer and Touzin (2000) while the second one (`MRG32k3a`) was proposed by L'Ecuyer (1999a) and is used in several simulation packages to provide multiple streams and substreams. This latter RNG has been heavily tested over the years and is very robust. On the other hand, the $\mathbb{F}_2$-linear generators are faster.

6.2 Statistical Testing

All the RNGs in Table 1 have been submitted to empirical statistical testing using the batteries Smallcrush, Crush, and Bigcrush of the TestU01 package (L'Ecuyer and Simard 2002, L'Ecuyer and Simard 2006). They passed all the tests in these batteries with the following notable exceptions: All $\mathbb{F}_2$-linear generators fail the tests that look for linear relationships in the sequences of bits they produce, namely the matrix-rank test (Marsaglia 1985) for huge binary matrices and the linear complexity tests (Erdmann 1992). The reason for this general failure is obvious: We know from their definitions that these generators produce bit sequences that obey linear recurrences, so they cannot have the linear complexity of a truly random sequence. This is definitely a limitation of these RNGs. But whenever the bit sequences are transformed nonlinearly by the application (e.g., to generate real-valued random numbers from non-uniform distributions), the linear relationships between the bits usually disappear, and the linearity is then very unlikely to cause a problem. For situations where simulation results can be noticeably affected by the linear dependencies among the bits, to make these RNGs safer without slowing them down too much, we could either combine them with a generator from another class (such as an MRG, for instance), or combine them with a small nonlinear RNG implemented via precomputed tables as suggested by L'Ecuyer and Granger-Piché (2003), or add a nonlinear output transformation that is fast to compute.

7 Conclusion

$\mathbb{F}_2$-linear RNGs are convenient for simulation because they are fast and the high-dimensional uniformity of their point sets can be measured by theoretical figures of merit that can be computed efficiently. Combined $\mathbb{F}_2$-linear generators with relatively small components have the important advantage of faster jumping-ahead, because the (smaller) components can be dealt with separately. Some $\mathbb{F}_2$-linear generators proposed in the literature have huge periods, but it is not always true that larger is better. A huge state has the disadvantages of using more memory (this can be important when there is a large number of streams in a simulation). It also makes jumping ahead much slower, and it requires more operations to modify a large fraction of the bits in the state. Of course, very long bit sequences produced by $\mathbb{F}_2$-linear generators will always fail statistical tests that measure their linear complexity. This can be viewed as a weak limitation, which could be overcome by adding a nonlinear output transformation or combining the $\mathbb{F}_2$-linear RNG with a generator from another class.

References

Brent, R. P. 2004. Note on Marsaglia's xorshift random number generators. *Journal of Statistical Software* 11 (5): 1–4. See `http://www.jstatsoft.org/v11/i05/brent.pdf`.

Brent, R. P., and P. Zimmermann. 2003. Random number generators with period divisible by a Mersenne prime. In *Computational Science and its Applications—ICCSA 2003*, Volume 2667, Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Compagner, A. 1991. The hierarchy of correlations in random binary sequences. *Journal of Statistical Physics* 63:883–896.

Couture, R., and P. L'Ecuyer. 2000. Lattice computations for random numbers. *Mathematics of Computation* 69 (230): 757–765.

Erdmann, E. D. 1992. Empirical tests of binary keystreams. Master's thesis, Department of Mathematics, Royal Holloway and Bedford New College, University of London.

Fushimi, M. 1983. Increasing the orders of equidistribution of the leading bits of the Tausworthe sequence. *Information Processing Letters* 16:189–192.

Golub, G. H., and C. F. V. Loan. 1996. *Matrix computations*. third ed. John Hopkins University Press.

Haramoto, H., M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer. 2006. Efficient jump ahead for $\mathbf{F}_2$-linear random number generators. GERAD Report G-2006-62, submitted.

Hellekalek, P., and G. Larcher. (Eds.) 1998. *Random and quasi-random point sets*, Volume 138 of *Lecture Notes in Statistics*. New York: Springer.

Knuth, D. E. 1998. *The art of computer programming, volume 2: Seminumerical algorithms*. Third ed. Reading, Mass.: Addison-Wesley.

Law, A. M., and W. D. Kelton. 2000. *Simulation modeling and analysis*. Third ed. New York: McGraw-Hill.

L'Ecuyer, P. 1994. Uniform random number generation. *Annals of Operations Research* 53:77–120.

L'Ecuyer, P. 1996. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation* 65 (213): 203–213.

L'Ecuyer, P. 1999a. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research* 47 (1): 159–164.

L'Ecuyer, P. 1999b. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation* 68 (225): 261–269.

L'Ecuyer, P. 2004. Polynomial integration lattices. In *Monte Carlo and Quasi-Monte Carlo Methods 2002*, ed. H. Niederreiter, 73–98. Berlin: Springer-Verlag.

L'Ecuyer, P. 2006. Uniform random number generation. In *Simulation*, ed. S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 55–81. Amsterdam, The Netherlands: Elsevier. Chapter 3.

L'Ecuyer, P., and E. Buist. 2005. Simulation in Java with SSJ. In *Proceedings of the 2005 Winter Simulation Conference*, 611–620: IEEE Press.

L'Ecuyer, P., and J. Granger-Piché. 2003. Combined generators with components from different families. *Mathematics and Computers in Simulation* 62:395–404.

L'Ecuyer, P., and C. Lemieux. 2002. Recent advances in randomized quasi-Monte Carlo methods. In *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*, ed. M. Dror, P. L'Ecuyer, and F. Szidarovszky, 419–474. Boston: Kluwer Academic.

L'Ecuyer, P., and F. Panneton. 2002. Construction of equidistributed generators based on linear recurrences modulo 2. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, ed. K.-T. Fang, F. J. Hickernell, and H. Niederreiter, 318–330. Berlin: Springer-Verlag.

L'Ecuyer, P., and R. Simard. 2002. *TestU01: A software library in ANSI C for empirical testing of random number generators*. Software user's guide. Available at `http://www.iro.umontreal.ca/~lecuyer`.

L'Ecuyer, P., and R. Simard. 2006. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*. to appear.

L'Ecuyer, P., R. Simard, E. J. Chen, and W. D. Kelton. 2002. An object-oriented random-number package with many long streams and substreams. *Operations Research* 50 (6): 1073–1075.

L'Ecuyer, P., and R. Touzin. 2000. Fast combined multiple recursive generators with multipliers of the form $a = \pm 2^q \pm 2^r$. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 683–689. Pistacaway, NJ: IEEE Press.

Lemieux, C., and P. L'Ecuyer. 2002. Randomized polynomial lattice rules for multivariate integration and simulation. extended version, available at `http://www.iro.umontreal.ca/~lecuyer`.

Lemieux, C., and P. L'Ecuyer. 2003. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing* 24 (5): 1768–1789.

Lenstra, A. K. 1985. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences* 30:235–248.

Lewis, T. G., and W. H. Payne. 1973. Generalized feedback shift register pseudorandom number algorithm. *Journal of the ACM* 20 (3): 456–468.

Lidl, R., and H. Niederreiter. 1986. *Introduction to finite fields and their applications.* Cambridge: Cambridge University Press.

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

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

Marsaglia, G. 2003. Xorshift RNGs. *Journal of Statistical Software* 8 (14): 1–6. See `http://www.jstatsoft.org/v08/i14/xorshift.pdf`.

Massey, J. L. 1969. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theor* IT-15:122–127.

Matsumoto, M., and Y. Kurita. 1992. Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation* 2 (3): 179–194.

Matsumoto, M., and Y. Kurita. 1994. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation* 4 (3): 254–266.

Matsumoto, M., and Y. Kurita. 1996. Strong deviations from randomness in m-sequences based on trinomials. *ACM Transactions on Modeling and Computer Simulation* 6 (2): 99–106.

Matsumoto, M., and T. Nishimura. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8 (1): 3–30.

Niederreiter, H. 1992. *Random number generation and quasi-Monte Carlo methods*, Volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia: SIAM.

Nishimura, T. 2000. Tables of 64-bit Mersenne twisters. *ACM Transactions on Modeling and Computer Simulation* 10 (4): 348–357.

Panneton, F. 2004, August. *Construction d'ensembles de points basée sur des récurrences linéaires dans un corps fini de caractéristique 2 pour la simulation Monte Carlo et l'intégration quasi-Monte Carlo.* Ph. D. thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada.

Panneton, F., and P. L'Ecuyer. 2004. Random number generators based on linear recurrences in F_{2^w}. In *Monte Carlo and Quasi-Monte Carlo Methods 2002*, ed. H. Niederreiter, 367–378. Berlin: Springer-Verlag.

Panneton, F., and P. L'Ecuyer. 2005a. On the xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation* 15 (4): 346–361.

Panneton, F., and P. L'Ecuyer. 2005b. Resolution-stationary random number generators. Technical report. GERAD Report G-2006-60.

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

Rieke, A., A.-R. Sadeghi, and W. Poguntke. 1998, August. On primitivity tests for polynomials. In *Proceedings of the 1998 IEEE International Symposium on Information Theory.* Cambridge, MA.

Strang, G. 1988. *Linear algebra and its applications*. third ed. Philadelphia, PA: Saunders.

Tausworthe, R. C. 1965. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation* 19:201–209.

Tezuka, S. 1995. *Uniform random numbers: Theory and practice.* Norwell, Mass.: Kluwer Academic Publishers.

Tezuka, S., and P. L'Ecuyer. 1991. Efficient and portable combined Tausworthe random number generators. *ACM Transactions on Modeling and Computer Simulation* 1 (2): 99–112.

Tootill, J. P. R., W. D. Robinson, and D. J. Eagle. 1973. An asymptotically random Tausworthe sequence. *Journal of the ACM* 20:469–481.

Wang, D., and A. Compagner. 1993. On the use of reducible polynomials as random number generators. *Mathematics of Computation* 60:363–374.