Improved Long-Period Generators Based on Linear Recurrences Modulo 2

FRANÇOIS PANNETON and PIERRE L'ECUYER
Université de Montréal
and
MAKOTO MATSUMOTO
Hiroshima University

Fast uniform random number generators with extremely long periods have been defined and implemented based on linear recurrences modulo 2. The twisted GFSR and the Mersenne twister are famous recent examples. Besides the period length, the statistical quality of these generators is usually assessed via their equidistribution properties. The huge-period generators proposed so far are not quite optimal in that respect. In this paper, we propose new generators of that form, with better equidistribution and "bit-mixing" properties for equivalent period length and speed. The state of our new generators evolves in a more chaotic way than for the Mersenne twister. We illustrate how this can reduce the impact of persistent dependencies among successive output values, which can be observed in certain parts of the period of gigantic generators such as the Mersenne twister.

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

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Random number generation, linear feedback shift register, GFSR linear recurrence modulo 2, Mersenne twister

1. INTRODUCTION

Most uniform random number generators (RNGs) used in computational statistics and simulation are based on linear recurrences modulo 2 or modulo a large integer. The main advantages of these generators are that fast implementations are available and that their mathematical properties can be studied in detail from a theoretical perspective. In particular, large-period linear RNGs are easy to design and the geometrical structure of the set Ψ_t of all vectors of t successive values produced by

Authors' addresses: François Panneton and Pierre L'Ecuyer, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada, e-mail: panneton@iro.umontreal.ca, lecuyer@iro.umontreal.ca; Makoto Matsumoto, Department of Mathematics, Graduate School of Science, Hiroshima University, Hiroshima 739-8526, Japan.

the generator, from all its possible initial states, can be precisely assessed without generating the points explicitly [L'Ecuyer 2004; Tezuka 1995]. In this paper, we are concerned with RNGs based on linear recurrences in $\mathbb{F}_2$, the finite field with two elements, 0 and 1, in which the arithmetic operations are equivalent to arithmetic modulo 2. Their output sequence is designed to imitate i.i.d. random variables uniformly distributed over the real interval $[0, 1]$. Such recurrences are attractive because their implementation can exploit the binary nature of computers. The set Ψ_t in this context is a finite subset of the unit hypercube $[0, 1]^t$ and it is customary to require that Ψ_t covers this unit hypercube very evenly for all t up to some large integer (as large as possible), because Ψ_t can be viewed as the "sample space" from which the t-dimensional points are drawn instead of drawing them uniformly from $[0, 1]^t$ [L'Ecuyer 2004].

A general framework for representing linear generators over $\mathbb{F}_2$ is given by the matrix linear recurrence

$$\mathbf{x}_i = \mathbf{A}\mathbf{x}_{i-1}, \tag{1}$$

$$\mathbf{y}_i = \mathbf{B}\mathbf{x}_i, \tag{2}$$

$$u_i = \sum_{\ell=1}^{w} y_{i,\ell-1} 2^{-\ell} = .y_{i,0}\, y_{i,1}\, y_{i,2}\, \cdots, \tag{3}$$

where $\mathbf{x}_i = (x_{i,0}, \ldots, x_{i,k-1})^\mathsf{T} \in \mathbb{F}_2^k$ and $\mathbf{y}_i = (y_{i,0}, \ldots, y_{i,w-1})^\mathsf{T} \in \mathbb{F}_2^w$ are the k-bit *state* and the w-bit *output vector* at step i, $\mathbf{A}$ and $\mathbf{B}$ are a $k \times k$ *transition matrix* and a $w \times k$ *output transformation matrix* (both with elements in $\mathbb{F}_2$), k and w are positive integers, and $u_i \in [0, 1)$ is the *output* at step i. All operations in (1) and (2) are performed in $\mathbb{F}_2$ and each element of $\mathbb{F}_2$ is represented as one bit. In this setup, we have

$$\Psi_t = \{(u_0, u_1, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}.$$

By appropriate choices of $\mathbf{A}$ and $\mathbf{B}$, several well-known types of generators can be obtained as special cases of this general class, including the Tausworthe, linear feedback shift register (LFSR), generalized feedback shift register (GFSR), twisted GFSR, Mersenne twister, and linear cellular automata [L'Ecuyer and Panneton 2002; Panneton 2004].

We briefly recall some (important) well-known facts about this class of RNGs. The *characteristic polynomial* of the matrix $\mathbf{A}$ can be written as

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

where $\mathbf{I}$ is the identity matrix and each α_j is in $\mathbb{F}_2$. For each j, the sequences $\{x_{i,j},\, i \geq 0\}$ and $\{y_{i,j},\, i \geq 0\}$ both obey the linear recurrence

$$x_{i,j} = (\alpha_1 x_{i-1,j} + \cdots + \alpha_k x_{i-k,j}) \bmod 2 \tag{4}$$

[Niederreiter 1992; L'Ecuyer 1994]. (The output sequence $\{y_{i,j},\, i \geq 0\}$ could also obey a recurrence of shorter order in certain cases, depending on $\mathbf{B}$.) Here we assume that $\alpha_k = 1$, in which case (4) has *order* k and is purely periodic. The period length of the recurrence (4) has the upper bound $2^k - 1$, which is achieved if and only if $P(z)$ is a primitive polynomial over $\mathbb{F}_2$ [Niederreiter 1992; Knuth

1998]. In this paper, we are only interested in generators having this *maximal-period* property.

The general goal, when constructing this type of generator, is to find matrices $\mathbf{A}$ and $\mathbf{B}$ such that (i) the corresponding recurrence can be implemented very efficiently by exploiting the special structure of these matrices, (ii) the set Ψ_t has excellent uniformity for all t up to some large integer, and (iii) the generator passes reasonable empirical statistical tests. Typically, the matrix $\mathbf{A}$ represents simple inexpensive operation such as shifts, rotations, masks, logical ands, and exclusive-ors (xors) between blocks of bits in the vector $\mathbf{x}_i$. The fastest generators have only a small number of these operations, which implies that there are very few changes to the bits of the state $\mathbf{x}_i$ from one step to the next, and this can often be detected by statistical tests. In particular, if $\mathbf{x}_i$ contains many more 0's than 1's, the same is likely to hold for $\mathbf{x}_{i+1}$. To obtain good and robust generators, the matrices $\mathbf{A}$ and $\mathbf{B}$ in (1) and (2) must perform enough bit transformations. Therefore, a compromise must be made between the speed and the quality. The uniformity of Ψ_t is often measured in terms of its equidistribution properties, whose definitions are recalled in Section 2.

Our aim in this paper is to build a new class of linear generators over $\mathbb{F}_2$ that reach a compromise in this sense. The proposed generators perform more bit transformations and are better equidistributed than (for example) the Mersenne twister, while having the same period length and approximately the same speed. To achieve that, we construct a matrix $\mathbf{A}$ that implements bit shifts, bit masks, ands, and xors cleverly spread out in the matrix. For $\mathbf{B}$, in most cases we simply use $\mathbf{I}_{w\times k}$, the $w \times w$ identity matrix to which we append $k - w$ columns of zeros. We prefer to put all (or most of) the transformations in $\mathbf{A}$ because these transformations carry over to the following states, whereas those in $\mathbf{B}$ do not.

In the next section of the paper, we recall basic definitions related to the (standard) use of equidistribution to measure the uniformity of Ψ_t for linear generators over $\mathbb{F}_2$. We also specify the uniformity measures adopted in this paper. In Section 3, we mention other linear generators over $\mathbb{F}_2$ that are already available, including combined LFSRs [L'Ecuyer 1999b], GFSRs, the Mersenne twister [Matsumoto and Nishimura 1998; Nishimura 2000], and combined twisted GFSRs [L'Ecuyer and Panneton 2002], and point out drawbacks of some of them. Section 4 describes the design of our new generators. Section 5 gives specific instances found by a computer search, using REGPOLY [L'Ecuyer and Panneton 2002], for well-equidistributed generators having state spaces of various sizes. Section 6 deals with implementation issues and speed. In Section 7, we compare the TT800 generator of [Matsumoto and Kurita 1994] and the Mersenne twister of [Matsumoto and Nishimura 1998] with two of our new generators having the same period lengths ($2^{800} - 1$ and $2^{19937} - 1$, respectively), in terms of their ability to quickly exit a region of the state space where the fraction of zeros in the state (or in the output vector $\mathbf{y}_i$) is far away from 1/2. The new generators perform much better in these experiments. That is, their behavior is more in line with randomness and chaos.

2. EQUIDISTRIBUTION AND MEASURES OF QUALITY

A convenient and rather standard way of measuring the uniformity of Ψ_t for linear RNGs over $\mathbb{F}_2$ is as follows. Recall that Ψ_t has cardinality 2^k. If we divide the interval $[0,1)$ into 2^ℓ equal segments for some positive integer ℓ, this determines a partition of the unit hypercube $[0,1)^t$ into $2^{t\ell}$ cubic cells of equal size, called a (t,ℓ)-*equidissection* in base 2. The set Ψ_t is said to be (t,ℓ)-*equidistributed* if each cell contains exactly $2^{k-t\ell}$ of its points. Of course, this is possible only if

$$\ell \le \ell_t^* \stackrel{\text{def}}{=} \min(w,\ \lfloor k/t \rfloor).$$

When the equidistribution holds for all pairs (t,ℓ) that satisfy this condition, we say that Ψ_t (and the RNG) is *maximally-equidistributed* (ME). ME generators have the best possible equidistribution properties in terms of cubic equidissections. A major motivation for using this type of uniformity measure for linear generators over $\mathbb{F}_2$ is that it can be efficiently computed in that case, without generating the points explicitly. The idea is simple: the first ℓ bits of $u_i, \ldots, u_{i+t-1}$ form a $t\ell$-bit vector that can be written as $\mathbf{M}_{t,\ell}\mathbf{x}_i$ for some $t\ell \times k$ binary matrix $\mathbf{M}_{t,\ell}$, because each output bit can be expressed as a linear combination of the bits of the current state, and the (t,ℓ)-equidistribution holds if and only if the matrix $\mathbf{M}_{t,\ell}$ has full rank [Fushimi and Tezuka 1983; L'Ecuyer 1996]. When k is very large, this matrix becomes expensive to handle, but in that case the equidistribution can be verified by a more efficient method based on the computation of the shortest nonzero vector in a lattice of formal series, as explained in Couture and L'Ecuyer [2000]. Large-period ME (or almost ME) generators have been proposed by L'Ecuyer [1999b], L'Ecuyer and Panneton [2002], and Panneton and L'Ecuyer [2004], for example. The aim of this paper is to propose new ones, with very large periods, and faster than those already available with comparable period lengths.

For non-ME generators, we denote t_ℓ as the largest dimension t for which Ψ_t is (t,ℓ)-equidistributed, and define the *dimension gap* for ℓ bits of resolution as

$$\delta_\ell = t_\ell^* - t_\ell,$$

where $t_\ell^* = \lfloor k/\ell \rfloor$ is an upper bound on the best possible value of t_ℓ. As measures of uniformity, we consider the *worst-case dimension gap* and the *sum of dimension gaps*, defined as

$$\Delta_\infty = \max_{1 \le \ell \le w} \delta_\ell \quad \text{and} \quad \Delta_1 = \sum_{\ell=1}^{w} \delta_\ell.$$

Aside from equidistribution, it has been strongly advocated that good linear generators over $\mathbb{F}_2$ must have characteristic polynomials $P(z)$ whose number of nonzero coefficients is not too far from half the degree, i.e., in the vicinity of $k/2$ [Compagner 1991; Wang and Compagner 1993]. In particular, generators for which $P(z)$ is a trinomial or a pentanomial, which have been widely used in the past, do not satisfy this condition and have been shown to fail rather simple statistical tests [Lindholm 1968; Matsumoto and Kurita 1996]. So, as a secondary quality criterion, we look at the number of nonzero coefficients in $P(z)$, which we denote by N_1.

3. OTHER LINEAR GENERATORS OVER $\mathbb{F}_2$

LFSR generators were studied a long time ago by Tausworthe [1965]. For a *trinomial-based LFSR* generator, we have $w = k$, $\mathbf{B} = \mathbf{I}$, and $\mathbf{A} = \tilde{\mathbf{A}}^s$ (in $\mathbb{F}_2$) for some small positive integer s, where $\tilde{\mathbf{A}}$ is the matrix that implements the recurrence for which $x_{i,0} = x_{i-1,q} \oplus x_{i-1,k-1}$ and $x_{i,\ell} = x_{i-1,\ell-1}$ for $0 < \ell < k$, where $0 < q < k$ and $\oplus$ represents the "exclusive-or" (xor) operation. A *trinomial-based GFSR* generator is defined by the recurrence $\mathbf{v}_{i,0} = \mathbf{v}_{i-1,q} \oplus \mathbf{v}_{i-1,r-1}$ and $\mathbf{v}_{i,\ell} = \mathbf{v}_{i-1,\ell-1}$ for $0 < \ell < r$, where $k = wr$ for some integer r, $\mathbf{x}_i$ is decomposed into w-bit blocks as $\mathbf{x}_i = (\mathbf{v}_{i,0}^\mathsf{T}, \ldots, \mathbf{v}_{i,r-1}^\mathsf{T})^\mathsf{T}$, and $\mathbf{B} = \mathbf{I}$. Its largest possible period length is $2^r - 1$, which is much smaller than 2^k.

A ME trinomial-based LFSR was proposed already by Tootill et al. [1973], who introduced the ME notion under the name of *asymptotically random*. Very fast algorithms are available for implementing trinomial-based LFSR generators whose parameters satisfy certain conditions, as well as for trinomial-based and pentanomial-based GFSR generators. However, these generators have much too few nonzero coefficients in their characteristic polynomials (N_1 is 3 or 5, which is too small). One way of getting fast generators with a large value of N_1, proposed by Tezuka and L'Ecuyer [1991] and Wang and Compagner [1993] and pursued by L'Ecuyer [1996; 1999b], is to combine several trinomial-based LFSR generators of relatively prime period lengths, by bitwise xor. This gives another LFSR whose characteristic polynomial $P(z)$ is the product of the characteristic polynomials of the components, so it may contain many more nonzero coefficients, e.g., up to 3^J if we combine J trinomial-based LFSRs and $3^J < k$. The polynomial $P(z)$ cannot be primitive in this context, but the period length can nevertheless be very close to 2^k where k is the degree of $P(z)$. This method is quite effective to build generators with $\Delta_\infty = \Delta_1 = 0$ and values of N_1 up to a few hundreds on 32-bit computers, but for values in the thousands or tens of thousands, one needs a large number J of components and this makes the implementation slower.

The twisted GFSR and Mersenne twister [Matsumoto and Kurita 1994; Matsumoto and Nishimura 1998; Nishimura 2000] provide very efficient implementations of linear generators over $\mathbb{F}_2$ with primitive characteristic polynomials of very large degree k. They are based on a recurrence similar to that of the GFSR but slightly more complicated, and a matrix $\mathbf{B} \neq \mathbf{I}$. However, their values of N_1 are typically much smaller than $k/2$ and they often have a large value of Δ_1, due to the fact that their equidistribution is far from optimal in large dimensions. For example, `MT19937` has $k = 19937$, $N_1 = 135$, and $\Delta_1 = 6750$. This can be improved by combining several twisted GFSRs or Mersenne twisters as in L'Ecuyer and Panneton [2002], but at the expense of getting a slower generator.

Our goal in this paper was to build $\mathbb{F}_2$-linear generators with primitive characteristic polynomials, with speed and period length comparable to the Mersenne twister, and for which $N_1 \approx k/2$ and $\Delta_1 = 0$ (or nearly). We attach the acronym WELL, for "Well Equidistributed Long-period Linear", to these new generators.

4. STRUCTURE OF THE NEW PROPOSED GENERATORS

The general recurrence for the WELL generators is defined by the algorithm given in Figure 1, with the following notation. We decompose k as $k = rw - p$

$$
\begin{array}{llllllllll}
\mathbf{z}_0 & \leftarrow & (\mathbf{m}_p \ \& \ \mathbf{v}_{i,r-1}) & \oplus & (\tilde{\mathbf{m}}_p \ \& \ \mathbf{v}_{i,r-2}); & & & & & \\
\mathbf{z}_1 & \leftarrow & \mathbf{T}_0\mathbf{v}_{i,0} & \oplus & \mathbf{T}_1\mathbf{v}_{i,m_1}; & & & & & \\
\mathbf{z}_2 & \leftarrow & \mathbf{T}_2\mathbf{v}_{i,m_2} & \oplus & \mathbf{T}_3\mathbf{v}_{i,m_3}; & & & & & \\
\mathbf{z}_3 & \leftarrow & \mathbf{z}_1 & \oplus & \mathbf{z}_2; & & & & & \\
\mathbf{z}_4 & \leftarrow & \mathbf{T}_4\mathbf{z}_0 & \oplus & \mathbf{T}_5\mathbf{z}_1 & \oplus & \mathbf{T}_6\mathbf{z}_2 & \oplus & \mathbf{T}_7\mathbf{z}_3; & \\
\mathbf{v}_{i+1,r-1} & \leftarrow & \mathbf{v}_{i,r-2} & \& & \mathbf{m}_p; & & & & & \\
\text{for } j = r-2, \ldots, 2, \text{ do} & & \mathbf{v}_{i+1,j} \leftarrow \mathbf{v}_{i,j-1}; & & & & & & & \\
\mathbf{v}_{i+1,1} & \leftarrow & \mathbf{z}_3; & & & & & & & \\
\mathbf{v}_{i+1,0} & \leftarrow & \mathbf{z}_4; & & & & & & & \\
\text{return } \mathbf{y}_i = \mathbf{v}_{i,0}. & & & & & & & & &
\end{array}
$$

Fig. 1. The WELL algorithm

where r and p are the unique integers such that $r > 0$ and $0 \le p < w$. The state vector $\mathbf{x}_i$ is decomposed into w-bit blocks as $\mathbf{x}_i = (\mathbf{v}_{i,0}^{\mathsf{T}}, \ldots, \mathbf{v}_{i,r-1}^{\mathsf{T}})^{\mathsf{T}}$ where the last p bits of $\mathbf{v}_{i,r-1}$ are always zero, and $\mathbf{T}_0, \ldots, \mathbf{T}_7$ are $w \times w$ binary matrices that apply linear transformations to these w-bit blocks. The integers k, p, m_1, m_2, m_3, where $0 < m_1, m_2, m_3 < r$, and the matrices $\mathbf{T}_0, \ldots, \mathbf{T}_7$, are chosen so that $P(z)$, the characteristic polynomial of $\mathbf{A}$, with degree $k = rw - p$, is primitive over $\mathbb{F}_2$. Here, $\mathbf{m}_p$ is a bit mask that keeps the first $w - p$ bits and sets all p other bits to zero, whereas $\tilde{\mathbf{m}}_p$ denotes its bitwise complement, that sets the first p bits to zero. The operators "$\oplus$", "&", and "$\leftarrow$" are the bitwise xor, the bitwise and, and the assignment statement, respectively. The temporary variables $\mathbf{z}_0, \ldots, \mathbf{z}_4$ are w-bit vectors.

Let $\mathbf{T}_{i,j,k} = \mathbf{T}_i\mathbf{T}_k \oplus \mathbf{T}_j\mathbf{T}_k$,

$$
\mathbf{U}_p = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{pmatrix} \quad \text{and } \mathbf{L}_{w-p} = \begin{pmatrix} \mathbf{I}_{w-p} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}
$$

be matrices of sizes $w \times w$, where $\mathbf{I}_p$ and $\mathbf{I}_{w-p}$ are the the identity matrices of size p and $w - p$, respectively. The matrix $\mathbf{A}$ that corresponds to the WELL algorithm has the following block structure, with six $w \times w$ nonzero submatrices on the first line and four on the second line:

$$
\mathbf{A} = \begin{pmatrix}
\mathbf{T}_{5,7,0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{T}_{5,7,1} & \mathbf{0} & \ldots & \mathbf{T}_{6,7,2} & \mathbf{0} & \ldots & \mathbf{T}_{6,7,3} & \ldots & \mathbf{T}_4\mathbf{U}_p & \mathbf{T}_4\mathbf{L}_{w-p} \\
\mathbf{T}_0 & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{T}_1 & \mathbf{0} & \ldots & \mathbf{T}_2 & \mathbf{0} & \ldots & \mathbf{T}_3 & \ldots & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{I} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{I} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\
 & & & & & \ddots & & & & & & & & \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \ldots & \mathbf{L}_{w-p}^{\mathsf{T}} & \mathbf{0}
\end{pmatrix}.
$$

As can be deduced from Figure 1, the submatrices $\mathbf{T}_{5,7,1}$, $\mathbf{T}_{6,7,2}$, and $\mathbf{T}_{6,7,3}$ are the $(m_1 + 1)$th, $(m_2 + 1)$th, and $(m_3 + 1)$th $w \times w$ blocks of the first line in the block decomposition of $\mathbf{A}$. If both $\mathbf{T}_0$ and $\mathbf{T}_4$ have full rank, then $\mathbf{A}$ has rank $wr - p$.

We take $\mathbf{y}_i = \mathbf{v}_{i,0}$ as the output vector at step i. Another possibility would be to take $\mathbf{y}_i = \mathbf{v}_{i,1}$, but in our empirical investigations, we generally obtained better equidistribution by taking $\mathbf{y}_i = \mathbf{v}_{i,0}$. Taking $\mathbf{y}_i = \mathbf{v}_{i,j}$ for $j > 1$ would be

equivalent to taking $\mathbf{y}_i = \mathbf{v}_{i,1}$, because $\mathbf{v}_{i,j} = \mathbf{v}_{i-1,j-1}$ for $j = 2, \ldots, r-1$ whereas $\mathbf{v}_{i,r}$ is $\mathbf{v}_{i-1,r-1}$ truncated to its first $w - p$ bits.

The transformations (matrices) $\mathbf{T}_j$ are selected among the possibilities enumerated in Table I. In this table, the bit vector $\mathbf{d}_s$ has its $(s+1)$-th bit set to zero and all other bits set to one, while "$\mathbf{x} \ll t$" and "$\mathbf{x} \gg t$" denote a left shift and a right shift by t bits, respectively. These transformations can be implemented very efficiently by just a few operations on a w-bit block.

Table I. Possibilities for the transformations $\mathbf{T}_j$

Transformation matrix $\mathbf{M}$	Implementation of $\mathbf{y} = \mathbf{M}\mathbf{x} = \mathbf{M}(x_0, \ldots, x_{k-1})^\mathsf{T}$
$\mathbf{M}_0$	$\mathbf{y} = \mathbf{0}$
$\mathbf{M}_1$	$\mathbf{y} = \mathbf{x}$
$\mathbf{M}_2(t)$ $-w \le t \le w$	$\mathbf{y} = \begin{cases} (\mathbf{x} \gg t) & \text{if } t \ge 0 \\ (\mathbf{x} \ll -t) & \text{otherwise} \end{cases}$
$\mathbf{M}_3(t)$ $-w \le t \le w$	$\mathbf{y} = \begin{cases} \mathbf{x} \oplus (\mathbf{x} \gg t) & \text{if } t \ge 0 \\ \mathbf{x} \oplus (\mathbf{x} \ll -t) & \text{otherwise} \end{cases}$
$\mathbf{M}_4(\mathbf{a})$ $\mathbf{a} \in \mathbb{F}_2^w$	$\mathbf{y} = \begin{cases} (\mathbf{x} \gg 1) \oplus \mathbf{a} & \text{if } x_{w-1} = 1 \\ (\mathbf{x} \gg 1) & \text{otherwise} \end{cases}$
$\mathbf{M}_5(t, \mathbf{b})$ $-w \le t \le w$ $\mathbf{b} \in \mathbb{F}_2^w$	$\mathbf{y} = \begin{cases} \mathbf{x} \oplus ((\mathbf{x} \ll t) \& \mathbf{b}) & \text{if } t \ge 0 \\ \mathbf{x} \oplus ((\mathbf{x} \gg -t) \& \mathbf{b}) & \text{otherwise} \end{cases}$
$\mathbf{M}_6(q, s, t, \mathbf{a})$ $0 \le q, s, t < w$ $\mathbf{a} \in \mathbb{F}_2^w$	$\mathbf{y} = \begin{cases} (((\mathbf{x} \ll q) \oplus (\mathbf{x} \gg (w-q))) \& \mathbf{d}_s) \oplus \mathbf{a} & \text{if } x_t = 1 \\ (((\mathbf{x} \ll q) \oplus (\mathbf{x} \gg (w-q))) \& \mathbf{d}_s) & \text{otherwise} \end{cases}$

This particular form of $\mathbf{A}$ can be motivated intuitively by the following arguments. First, for reasons of efficiency, we want to tap only a small number of blocks $\mathbf{v}_{i-1,j}$ and modify only a small number of blocks $\mathbf{v}_{i,j}$ at each iteration. This is why the majority of rows and columns in the block structure of $\mathbf{A}$ are zero, except for identity matrices in the block subdiagonals, whose role is to shift the unmodified w-bit blocks by one position. At each step, we use six blocks to modify the first two, $\mathbf{v}_{i,0}$ and $\mathbf{v}_{i,1}$. These two blocks will go down the vector and will be retapped several times in future steps, so their modification has more repercussion than if we would modify some other $\mathbf{v}_{i,j}$ for j near r. This gives room for better bit mixing which, according to our empirical observations, increases the chances of having a good equidistribution. We use the same blocks to modify both $\mathbf{v}_{i,0}$ and $\mathbf{v}_{i,1}$, and even use them in the same way to a certain extent, again for efficiency reasons. That is, the algorithm first computes $\mathbf{z}_0$, $\mathbf{z}_1$, and $\mathbf{z}_2$ and then recycles them to obtain $\mathbf{z}_3$ and $\mathbf{z}_4$. We tried several other variants of this general approach, with a constraint on the weighted number of operations required at each step (giving an appropriate weight to each type of operation), and the algorithm retained is the one that gave the best results in terms of equidistribution.

The Mersenne twister (MT) generator of Matsumoto and Nishimura [1998] turns out to be essentially a simplified version of the WELL algorithm, where $\mathbf{T}_1 = \mathbf{T}_2 = \mathbf{T}_3 = \mathbf{T}_{6,7,2} = \mathbf{T}_{6,7,3} = \mathbf{0}$, $\mathbf{T}_0 = \mathbf{T}_{5,7,1} = \mathbf{I}$, and $\mathbf{T}_4$ is a *companion* matrix whose only nonzero elements are on the first line and on the first subdiagonal (which contains all 1's). The proposed method is thus a substantial generalization of MT.

5. SPECIFIC PARAMETERS

Table II lists specific parameters for generators of period length ranging from $2^{512}-1$ to $2^{44497}-1$. The values of the vectors $\mathbf{a}_i$ are $\mathbf{a}_1 =$ `da442d24`, $\mathbf{a}_2 =$ `d3e43ffd`, $\mathbf{a}_3 =$ `8bdcb91e`, $\mathbf{a}_4 =$ `86a9d87e`, $\mathbf{a}_5 =$ `a8c296d1`, $\mathbf{a}_6 =$ `5d6b45cc` and $\mathbf{a}_7 =$ `b729fcec`. Several of the proposed generators are ME (they have $\Delta_\infty = \Delta_1 = 0$). For example, all the generators listed with $k = 512, 521, 607, 1024$ have this property. We did not find ME generators with $\mathbf{B} = \mathbf{I}_{w\times k}$ for $k = 800, 19937, 21701, 23209, 44497$, but all those given in Table II have $\Delta_\infty = 1$, a very small value of Δ_1 (from 1 to 7), and values of N_1 not far from $k/2$. For comparison, the `TT800` generator proposed by Matsumoto and Kurita [1994], with period length $2^{800}-1$, has $\Delta_1 = 261$ and $N_1 = 93$, whereas the Mersenne twister `MT19937` of Matsumoto and Nishimura [1998] has $\Delta_1 = 6750$ and $N_1 = 135$. Table III complements Table II by giving, for each generator, the values of ℓ for which $\delta_\ell > 0$.

For the generators that are not ME, it is easy to make them ME by adding a Matsumoto-Kurita tempering to the output. This is done with the following operations:

$$\begin{aligned} \mathbf{z} &\leftarrow \text{trunc}_w(\mathbf{x}_i) \\ \mathbf{z} &\leftarrow \mathbf{z} \oplus ((\mathbf{z} \ll 7) \,\&\, \mathbf{b}) \\ \mathbf{y}_i &\leftarrow \mathbf{z} \oplus ((\mathbf{z} \ll 15) \,\&\, \mathbf{c}) \end{aligned}$$

where $\mathbf{b}$ and $\mathbf{c}$ are carefully selected w-bit vectors and "trunc_w" means truncation to the first w bits. The output value u_i is generated via equation (3) with this $\mathbf{y}_i$. Adding this tempering to WELL44497a with $\mathbf{b} =$ `93dd1400` and $\mathbf{c} =$ `fa118000` provides the largest ME generator with 32 bits of accuracy known so far. Adding this tempering to WELL19937a with $\mathbf{b} =$ `e46e1700` and $\mathbf{c} =$ `9b868000` transforms it into the ME generator WELL19937c.

To find the generators listed in Table II, we did a random search using REGPOLY [L'Ecuyer and Panneton 2002]. It was designed to look for full period generators that use a small number of binary operations. For example, it would not consider the case where $\mathbf{T}_0 = \ldots = \mathbf{T}_7 = \mathbf{M}_6$ because $\mathbf{M}_6$ is the most expensive of the matrices of Table I. Once a full-period generator is found, its equidistribution properties are verified.

To find a full period generator, we must find a transition matrix having a primitive characteristic polynomial. To do this, we find the minimal polynomial over $\mathbb{F}_2$ of the sequence $\{y_{n,0}\}_{n\geq 0}$ generated by the first bit of the $\mathbf{y}_i$'s. If this polynomial is of degree k, then it is the characteristic polynomial of the transition matrix, otherwise it is a divisor of it. In the former case, we test this polynomial for primitivity using an algorithm proposed by Rieke et al. [1998] if $2^k - 1$ is not prime. If $2^k - 1$ is a Mersenne prime, then we can switch to an irreducibility test which is simpler and equivalent. For this purpose, we used a combination of the sieving algorithm described by Brent et al. [2003] and the Berlekamp [1970] algorithm implemented in the software package ZEN [Chabaud and Lercier 2000]. The method used to compute the equidistribution is the one introduced by Couture and L'Ecuyer [2000].

Table II. Specific well-equidistributed generators

			$\mathbf{T}_0$	$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	Δ_1
m_1	m_2	m_3	$\mathbf{T}_4$	$\mathbf{T}_5$	$\mathbf{T}_6$	$\mathbf{T}_7$	N_1
$k = 512$, $w = 32$, $r = 16$, $p = 0$							
WELL512a			$\mathbf{M}_3(-16)$	$\mathbf{M}_3(-15)$	$\mathbf{M}_3(11)$	$\mathbf{M}_0$	0
13	9	5	$\mathbf{M}_3(-2)$	$\mathbf{M}_3(-18)$	$\mathbf{M}_3(-28)$	$\mathbf{M}_5(-5, \mathbf{a}_1)$	225
$k = 521$, $w = 32$, $r = 17$, $p = 23$							
WELL521a			$\mathbf{M}_3(-13)$	$\mathbf{M}_3(-15)$	$\mathbf{M}_1$	$\mathbf{M}_2(-21)$	0
13	11	10	$\mathbf{M}_3(-13)$	$\mathbf{M}_2(1)$	$\mathbf{M}_0$	$\mathbf{M}_3(11)$	265
WELL521b			$\mathbf{M}_3(-21)$	$\mathbf{M}_3(6)$	$\mathbf{M}_0$	$\mathbf{M}_3(-13)$	0
11	10	7	$\mathbf{M}_3(13)$	$\mathbf{M}_2(-10)$	$\mathbf{M}_2(-5)$	$\mathbf{M}_3(13)$	245
$k = 607$, $w = 32$, $r = 19$, $p = 1$							
WELL607a			$\mathbf{M}_3(19)$	$\mathbf{M}_3(11)$	$\mathbf{M}_3(-14)$	$\mathbf{M}_1$	0
16	15	14	$\mathbf{M}_3(18)$	$\mathbf{M}_1$	$\mathbf{M}_0$	$\mathbf{M}_3(-5)$	295
WELL607b			$\mathbf{M}_3(-18)$	$\mathbf{M}_3(-14)$	$\mathbf{M}_0$	$\mathbf{M}_3(18)$	0
16	8	13	$\mathbf{M}_3(-24)$	$\mathbf{M}_3(5)$	$\mathbf{M}_3(-1)$	$\mathbf{M}_0$	313
$k = 800$, $w = 32$, $r = 25$, $p = 0$							
WELL800a			$\mathbf{M}_1$	$\mathbf{M}_3(-15)$	$\mathbf{M}_3(10)$	$\mathbf{M}_3(-11)$	3
14	18	17	$\mathbf{M}_3(16)$	$\mathbf{M}_2(20)$	$\mathbf{M}_1$	$\mathbf{M}_3(-28)$	303
WELL800b			$\mathbf{M}_3(-29)$	$\mathbf{M}_2(-14)$	$\mathbf{M}_1$	$\mathbf{M}_2(19)$	3
9	4	22	$\mathbf{M}_1$	$\mathbf{M}_3(10)$	$\mathbf{M}_4(\mathbf{a}_2)$	$\mathbf{M}_3(-25)$	409
$k = 1024$, $w = 32$, $r = 32$, $p = 0$							
WELL1024a			$\mathbf{M}_1$	$\mathbf{M}_3(8)$	$\mathbf{M}_3(-19)$	$\mathbf{M}_3(-14)$	0
3	24	10	$\mathbf{M}_3(-11)$	$\mathbf{M}_3(-7)$	$\mathbf{M}_3(-13)$	$\mathbf{M}_0$	407
WELL1024b			$\mathbf{M}_3(-21)$	$\mathbf{M}_3(17)$	$\mathbf{M}_4(\mathbf{a}_3)$	$\mathbf{M}_3(15)$	0
22	25	26	$\mathbf{M}_3(-14)$	$\mathbf{M}_3(-21)$	$\mathbf{M}_1$	$\mathbf{M}_0$	475
$k = 19937$, $w = 32$, $r = 624$, $p = 31$							
WELL19937a			$\mathbf{M}_3(-25)$	$\mathbf{M}_3(27)$	$\mathbf{M}_2(9)$	$\mathbf{M}_3(1)$	4
70	179	449	$\mathbf{M}_1$	$\mathbf{M}_3(-9)$	$\mathbf{M}_3(-21)$	$\mathbf{M}_3(21)$	8585
WELL19937b			$\mathbf{M}_3(7)$	$\mathbf{M}_1$	$\mathbf{M}_3(12)$	$\mathbf{M}_3(-10)$	5
203	613	123	$\mathbf{M}_3(-19)$	$\mathbf{M}_2(-11)$	$\mathbf{M}_3(4)$	$\mathbf{M}_3(-10)$	9679
WELL19937c			WELL19937a with M-K tempering				0
			b=`e46e1700`, **c**=`9b868000`				8585
$k = 21701$, $w = 32$, $r = 679$, $p = 27$							
WELL21701a			$\mathbf{M}_1$	$\mathbf{M}_3(-26)$	$\mathbf{M}_3(19)$	$\mathbf{M}_0$	1
151	327	84	$\mathbf{M}_3(27)$	$\mathbf{M}_3(-11)$	$\mathbf{M}_6(15, 10, 27, \mathbf{a}_4)$	$\mathbf{M}_3(-16)$	7609
$k = 23209$, $w = 32$, $r = 726$, $p = 23$							
WELL23209a			$\mathbf{M}_3(28)$	$\mathbf{M}_1$	$\mathbf{M}_3(18)$	$\mathbf{M}_3(3)$	3
667	43	462	$\mathbf{M}_3(21)$	$\mathbf{M}_3(-17)$	$\mathbf{M}_3(-28)$	$\mathbf{M}_3(-1)$	10871
WELL23209b			$\mathbf{M}_4(\mathbf{a}_5)$	$\mathbf{M}_1$	$\mathbf{M}_6(15, 30, 15, \mathbf{a}_6)$	$\mathbf{M}_3(-24)$	3
610	175	662	$\mathbf{M}_3(-26)$	$\mathbf{M}_1$	$\mathbf{M}_0$	$\mathbf{M}_3(16)$	10651
$k = 44497$, $w = 32$, $r = 1391$, $p = 15$							
WELL44497a			$\mathbf{M}_3(-24)$	$\mathbf{M}_3(30)$	$\mathbf{M}_3(-10)$	$\mathbf{M}_2(-26)$	7
23	481	229	$\mathbf{M}_1$	$\mathbf{M}_3(20)$	$\mathbf{M}_6(9, 14, 5, \mathbf{a}_7)$	$\mathbf{M}_1$	16883
WELL44497b			WELL44497a with M-K tempering				0
			b=`93dd1400`, **c**=`fa118000`				16883

Table III. Nonzero dimension gaps for the proposed WELL's

WELL	Δ_∞	Δ_1	$\{\ell : \delta_\ell = 1\}$
800a	1	3	$\{20, 25, 32\}$
800b	1	3	$\{5, 17, 25\}$
19937a	1	4	$\{2, 7, 15, 28\}$
19937b	1	5	$\{3, 9, 14, 16, 32\}$
21701a	1	1	$\{20\}$
23209a	1	3	$\{6, 23, 24\}$
23209b	1	3	$\{3, 4, 12\}$
44497a	1	7	$\{2, 3, 4, 8, 16, 24, 27\}$

6. IMPLEMENTATION AND PERFORMANCE

Figure 2 gives an implementation in C of WELL1024a, whose parameters are given in Table II. This implementation uses an array of $r = 32$ `unsigned int`'s to store the $\mathbf{v}_{i,j}$ vectors. The integer `state_i` is equal to $i \bmod r$. Since r is a power of 2 in this particular case, the operation "modulo 2^r" can be implemented efficiently using a bit mask, namely `0x0000001f`. The vector $\mathbf{v}_{i,j}$ is stored in `STATE[`$(r - i + j) \bmod r$`]`. By decrementing `state_i` at each function call, we automatically perform the "for" loop of the generator algorithm, saving costly assignment operations.

Implementations of WELL512a, WELL19937a, WELL19937c, WELL44497a, and WELL44497b are available on the web pages of the first two authors. The C code of the last four generators is significantly longer and more complicated than that of WELL512a and WELL1024a, because the order k is no longer a power of 2, and we are using special tricks to avoid performing several costly "mod r" operations.

Table IV. Time to generate and sum 10^9 random numbers

Generator	Time (s)
WELL512a	35.8
WELL1024a	35.7
WELL19937a	37.1
WELL19937c	37.2
WELL44497a	40.9
WELL44497b	38.8
F2wLFSR2_31_800	39.6
F2wLFSR3_7_800	35.2
TT800	42.7
MT19937	30.9
MRG32k3a	97.0

In table IV, we give the CPU time taken by a program that produces 10^9 random numbers and sums them, using different WELL generators from Table II, the Mersenne Twister [Matsumoto and Nishimura 1998], TT800 [Matsumoto and Kurita 1994] F2wLFSR3_7_800, and F2wLFSR2_31_800 [Panneton and L'Ecuyer 2004], which are all fast generators based on linear recurrences modulo 2. For comparison, we also include the widely used MRG32k3a [L'Ecuyer 1999a]. For the last five generators, we used the implementations published in the original papers. The test was performed on a computer equipped with a 2.8Ghz Intel Pentium 4 processor and a Linux operating system, and the program was compiled using `gcc` with the `-O2` optimization flag. We repeated the timing experiments five times and the results

```
#define R 32
#define M1 3
#define M2 24
#define M3 10
#define MAT3POS(t,v) (v^(v>>t))
#define MAT3NEG(t,v) (v^(v<<(-t)))
#define Identity(v)  (v)
#define V0       STATE[  state_i                   ]
#define VM1      STATE[ (state_i+M1) & 0x0000001fUL ]
#define VM2      STATE[ (state_i+M2) & 0x0000001fUL ]
#define VM3      STATE[ (state_i+M3) & 0x0000001fUL ]
#define VRm1     STATE[ (state_i+31) & 0x0000001fUL ]
#define newV0    STATE[ (state_i+31) & 0x0000001fUL ]
#define newV1    STATE[  state_i                   ]
static unsigned int z0, z1, z2, state_i;
static unsigned int STATE[R];

void InitWELL1024a (unsigned int *init){
   int j;
   state_i = 0;
   for (j = 0; j < R; j++) STATE[j] = init[j];
}
double WELL1024a (void){
  z0    = VRm1;
  z1    = Identity(V0)       ^ MAT3POS (8, VM1);
  z2    = MAT3NEG (-19, VM2) ^ MAT3NEG(-14,VM3);
  newV1 = z1                 ^ z2;
  newV0 = MAT3NEG (-11,z0)   ^ MAT3NEG(-7,z1)    ^ MAT3NEG(-13,z2) ;
  state_i = (state_i + 31)   & 0x0000001fUL;
  return ((double) STATE[state_i]  * 2.3283064365386962890e-10);
}
```

Fig. 2. An implementation of WELL1024a in C

agreed to within 0.5 seconds of accuracy. The numbers are rounded to the nearest integer. Interestingly, the WELL44497b generator (with tempering) is slightly faster than its basic version WELL44497a (without tempering). This strange behavior is probably an artefact of the compiler optimization methods, which operate differently in the two cases. The biggest WELL generators are almost as fast as the smaller ones. On the other hand, they use a larger amount of memory. So it is not necessarily true that WELL44497b should always be preferred over WELL1024, for example, even if it has a much longer period.

The WELL generators mentioned in Table IV successfully passed all the statistical tests included in the batteries Smallcrush, Crush and Bigcrush of TestU01 [L'Ecuyer and Simard 2002], except those that look for linear dependencies in a long sequence of bits, such as the matrix-rank test [Marsaglia 1985] for very large binary matrices and the linear complexity tests [Erdmann 1992]. This is in fact a limitation of *all* $\mathbb{F}_2$-linear generators, including the Mersenne twister, the TT800, etc. Because of their linear nature, the sequences produced by these generators just cannot have the linear complexity of a truly random sequence. This is definitely unacceptable in cryptology, for example, but is quite acceptable for the vast majority of simulation applications if the linear dependencies are of long range and high order.

7. ESCAPING ZEROLAND

In this section, we examine how quickly the WELL generators can escape from a bad initialization, e.g., one that contains only a few bits set to 1. We compare them with other generators, for which the initialization often has a long-lasting impact in the sense that the fraction of bits set to 1 in both the state and the output remains small for a large number of steps if the initial number is very small.

Our experiments are based on the following framework. Let $S_1 = \{\mathbf{e}_j : 1 \le j \le k\}$, where $\mathbf{e}_j$ denotes the j-th unit vector. This is the set of states for which a single bit is set to 1. Let $\mathbf{y}_i^{(j)}$ be the output vector $\mathbf{y}_i$ at step i when the initial state is $\mathbf{x}_0 = \mathbf{e}_j$, and let $u_i^{(j)}$ be the corresponding output value u_i. We denote by $H(\mathbf{x})$ the *Hamming weight* of a bit vector $\mathbf{x}$, i.e., the number of its bits that are set to 1, and define

$$\gamma_{n,p} = \frac{1}{pkw} \sum_{i=n}^{n+p-1} \sum_{j=1}^{k} H(\mathbf{y}_i^{(j)}). \tag{5}$$

This moving average represents the fraction of bits that are 1 in the p successive output vectors $\mathbf{y}_n, \ldots, \mathbf{y}_{n+p-1}$, averaged over all initial seeds in S_1. Under the null hypothesis that the $\mathbf{y}_i$'s are independent bit vectors uniformly distributed over $\mathbb{F}_2^w$, $\gamma_{n,p}$ should be approximately normally distributed with mean 1/2 and variance $1/(4pkw)$.

Figure 3 illustrates the behavior of $\gamma_{n,100}$ for $n = 100, \ldots, 10^5$, for the WELL800a and TT800 generators. We see that WELL800a reaches the neighborhood of 0.5 much faster than TT800. For the latter, $\gamma_{n,100}$ is often below 0.49 for n up to more than 60000. Here, the standard deviation of $\gamma_{n,100}$ should be 1/3200, so 0.49 is 32 standard deviations away from the mean.

In Figure 4, we see a similar comparison between WELL19937a and MT19937, with $\gamma_{n,1000}$ and n up to one million. Again, the WELL generator is much better behaved. With MT19937, $\gamma_{n,1000}$ has an easily visible negative bias for up to at least $n = 700000$ steps.

Although not visible in the scale of these two figures, the WELL generators also need a certain number of steps to bring their states close to the "center" of the state space, where roughly half of the bits are 1's. To show that, Figure 5 gives a close-up view of the early behavior of $\gamma_{n,5}$ for WELL800a and WELL19937a. There is still a visible (and significant) amount of dependence from the initial state, but the number of steps required to clear up the effect is approximately 1000 times smaller than for TT800 and MT19937.

We also computed similar figures but with $u_i^{(j)}$ instead of $H(\mathbf{y}_i^{(j)})/w$ in (5), i.e., the average output value instead of the overall fraction of bits set to 1, and the results were almost identical. That alternative gives more weight to the more significant bits.

The behavior of the Hamming weight that we just examined is closely related to the *diffusion capacity* of generators. Crudely speaking, a generator is said to have good diffusion capacity if it produces two sequences that are very different from each other when started from two seeds that are very close to each other [Shannon 1949]. A precise definition depends on how we measure the closeness of seeds and the differences of sequences.

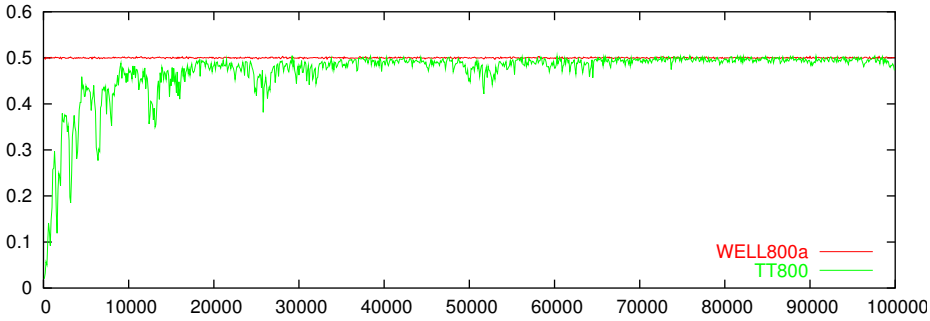

Fig. 3. $\gamma_{n,100}$ for the WELL800a (top line) and TT800 (lower line)

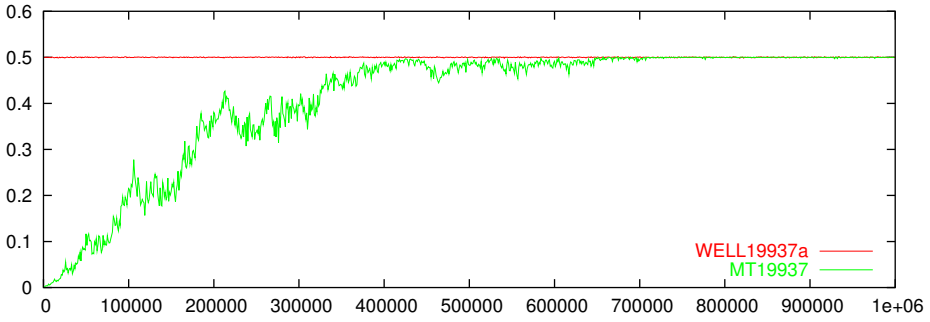

Fig. 4. $\gamma_{n,1000}$ for the WELL19937a (top line) and MT19937 (lower line)

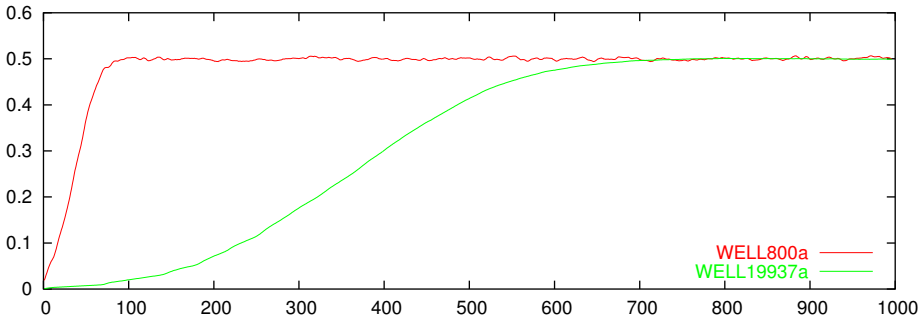

Fig. 5. $\gamma_{n,5}$ for the WELL800a (top line) and WELL19937a (lower line)

For the generators considered in this paper, we may define the distance between two seeds or two output vectors as the number of bits that differ between them. This is the *Hamming distance*. If two seeds $\mathbf{x}'_0$ and $\mathbf{x}''_0$ are at distance 1, then $\mathbf{x}'_0 \oplus \mathbf{x}''_0 = \mathbf{e}_j$ for some j. If $\mathbf{y}'_i$ and $\mathbf{y}''_i$ are the corresponding output vectors at step i, then because of the linearity, we have $\mathbf{y}'_i \oplus \mathbf{y}''_i = \mathbf{B}\mathbf{A}^i(\mathbf{x}'_0 \oplus \mathbf{x}''_0) = \mathbf{B}\mathbf{A}^i\mathbf{e}_j = \mathbf{y}_i^{(j)}$, so the distance between $\mathbf{y}'_i$ and $\mathbf{y}''_i$ is the Hamming weight of $\mathbf{y}_i^{(j)}$. In this sense, the average Hamming weight defined in (5) also measures the diffusion capacity of

the generator.

What is the potential impact of this weakness of MT and TT generators on simulation results for practical applications? If the generator's state is initialized with k independent uniform (truly) random bits, the probability of hitting a bad state is probably quite small (a more precise statement would require a formal definition of "bad state" and further analysis). But the chances are still there that the generator eventually hits a bad state during a simulation. Moreover, such huge generators are rarely seeded by k truly random bits in practice, and seeds that contain many zeros are perhaps more likely to be used because they are easier to write down. This could lead to totally wrong simulation results. The WELL generators provide a substantial improvement on this aspect.

8. CONCLUSION

Random number generators based on linear recurrences modulo 2 are popular and very convenient for simulation, mainly because of their high speed and because the quality of specific instances can be studied in detail from the mathematical viewpoint. Other generators from this family have been proposed in the past, for instance the Mersenne twister and the combined LFSRs. Some of them have huge period lengths, so their states must be represented over a large number of bits, and require only a few operations on 32-bit (or 64-bit) words to go from one state to the next. This means that only a small fraction of the bits in the state are modified at each step. In this paper, we have proposed a class of generators that do better than the previous ones on this aspect, without compromising the period length and speed. The improvement was assessed by a measure of equidistribution, by counting the fraction of nonzero coefficients in the characteristic polynomial of the recurrence, and by an empirical comparison of the diffusion capacity of the old and new generators. Of course, like all linear generators over $\mathbb{F}_2$, these generators cannot pass a statistical test that measures the linear complexity of a long sequence of bits that they produce. But they are nevertheless very useful and safe for the vast majority of simulation applications, especially when the generator's speed is important.

ACKNOWLEDGMENTS

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant No. ODGP0110050, NATEQ-Québec grant No. 02ER3218, and a Canada Research Chair to the second author, as well as JSPS Grant-In-Aid No. 14654021 to the third author. The first author benefited from NSERC and NATEQ scholarships. We thank Takuji Nishimura, who gave us useful ideas to improve the performance of our algorithms for checking primitivity and equidistribution, Étienne Marcotte, who produced the figures, Hiroshi Haramoto, who found some mistakes in an earlier version, and the anonymous reviewers whose comments helped improve the paper.

REFERENCES

Berlekamp, E. R. 1970. Factoring polynomials over large finite fields. *Math. Comp. 24*, 713–735.

Brent, R. P., Larvala, S., and Zimmermann, P. 2003. A fast algorithm for testing reducibility

of trinomials mod 2 and some new primitive trinomials of degree 3021377. *Math. Comp. 72,* 243, 1443–1452.

Chabaud, F. and Lercier, R. 2000. A toolbox for fast computation in finite extension over finite rings. Software user's guide. See `http://zenfact.sourceforge.net/`.

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

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

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

Fushimi, M. and Tezuka, S. 1983. The k-distribution of generalized feedback shift register pseudorandom numbers. *Communications of the ACM 26,* 7, 516–523.

Knuth, D. E. 1998. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third ed. Addison-Wesley, Reading, Mass.

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. Random number generation. In *Handbook of Computational Statistics*, J. E. Gentle, W. Haerdle, and Y. Mori, Eds. Springer-Verlag, Berlin, 35–70. Chapter II.2.

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

L'Ecuyer, P. and Simard, R. 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`.

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*. Elsevier Science Publishers, North-Holland, Amsterdam, 3–10.

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

Matsumoto, M. and Kurita, Y. 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 Nishimura, T. 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*. SIAM CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 63. SIAM, Philadelphia.

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

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

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

Shannon, C. E. 1949. Communication theory of secrecy systems. *Bell System Technical Journal 28*, 656–715.

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*. Kluwer Academic Publishers, Norwell, Mass.

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

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

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

Not received yet