Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

A TUTORIAL ON UNIFORM VARIATE GENERATION

Pierre L'Ecuyer
Département d'informatique
Université Laval
Ste-Foy, Qué., Canada, G1K 7P4.

ABSTRACT

In typical stochastic simulations, randomness is produced by generating a sequence of independent uniform variates (usually real-valued between 0 and 1, or integer-valued in some interval) and transforming them in the appropriate way. In this tutorial, we examine practical ways of generating such variates on a computer. We compare them in terms of ease of implementation, efficiency, flexibility, theoretical support, and statistical robustness. We look in particular at the following classes of generators: linear congruential (in scalar and matrix form), lagged-Fibonacci (including generalized feedback shift register) and combined. We also mention others and give a bibliographic survey of the most recent papers on the subject.

1. INTRODUCTION

In the mind of the average computer user, the problem of generating uniform variates by computer has been solved long ago. After all, every computer system offers one or more function(s) to do so. Many software products, like compilers, spreadsheets, statistical or numerical packages, etc. also offer their own. These functions supposedly return numbers that could be used, for all practical purposes, as if they were the values taken by independent random variables, with a uniform distribution between 0 and 1. Many people use them with faith and feel happy with the results. So, why bother ?

Well, other (less naive) people *do not* feel happy with the results and with good reasons. Despite renewed crusades, blatantly bad generators still abound, especially on microcomputers (see e.g. L'Ecuyer (1987), Modianos et al. (1987), Park and Miller (1988), Ripley (1988), Wichmann and Hill (1987)). Other generators widely used on medium-sized computers are perhaps not so spectacularly bad, but still fail some theoretical and/or empirical statistical tests, and/or generate easily detectable regular patterns (L'Ecuyer (1988), Marsaglia (1985)).

Fortunately, many applications appear quite robust to these defects. In the words of Marsaglia (1985), "a random number generator is much like sex: when its good its wonderful, and when its bad its still pretty good". But with the fast increase in desktop computing power, increasingly sophisticated simulation studies are being performed that require more and more "random" numbers and whose results are more sensitive to the quality of the underlying generator (Marsaglia (1985), Ripley (1988)). Sometimes, using a not-so-good generator can give totally misleading results. Carrying on with Marsaglia's analogy, perhaps we could add: "in both cases, careless behavior may lead to disaster". As suggested by the list of references at the end of this paper and in L'Ecuyer (1989), researchers in the field (of variate generation) are actively studying "proper measures to reduce the risks". The goal is to design more robust generators, without having to pay too much in terms of portability, flexibility and efficiency.

In the next sections, we give a quick overview of this ongoing research. We focus mainly on efficient and recently proposed techniques for generating *uniform pseudorandom numbers*. Stochastic simulations typically transform such numbers to generate variates according to more complex distributions (Bratley et al. (1987), Devroye (1986)). Here, "uniform pseudorandom" means that the numbers behave from the outside as if they were the values of i.i.d. random variables, uniformly distributed over some finite set of symbols. This set of symbols is often a set of integers of the form $\{1, \dots, m-1\}$ and the symbols are usually transformed by some function into values between 0 and 1, to approximate the $U(0,1)$ distribution.

Other tutorial-like references on uniform variate generation include Bratley et al. (1987), Knuth (1981), Law and Kelton (1982) and Marsaglia (1985).

2. VIEWS OF RANDOMNESS

2.1. Classical definitions

In the classical (Kolmogorov) sense, a string of bits is *random* if it cannot be described by a shorter string than itself. A generalization is that it cannot be produced efficiently (e.g. in polynomial time), by a program smaller than itself. For references and other definitions, see e.g. Golreich et al. (1986) and Knuth (1981). These definitions are certainly not practical enough for simulation applications, since k *truly* random bits would be necessary for outputting k "random" bits in this sense.

In the early days, *physical devices* (like noise diodes, Geiger counters, etc.) have been attached to computers with the aim of producing such "true" random bits (see references in Chor and Goldreich (1988)). These methods were abandonned for many reasons, e.g., using such specialized hardware is not convenient, a sequence of numbers cannot be repeated without storing it and, more importantly, the numbers produced are not uniformly distributed (Bratley et al. (1987), Chor and Goldreich (1988)). Work is still being done on ways to extract "random-looking" bits from imperfect physical sources of randomness (Chor and Goldreich (1988), Vazirani (1987)), but for the moment, these techniques are still not practical enough for standard simulation applications.

2.2. A framework for PRNGs

The so-called "random number generators" that are used in practice are in fact deterministic functions that produce a periodic sequence of numbers. When their initial state (called the seed) is truly random, they can be viewed as *extensors* of randomness, whose purpose is to save "coin tosses". They stretch a short truly random seed into a long sequence of values that is supposed to appear and behave like a true random sequence. For this reason, they are often called *pseudorandom*. We now set up a framework for studying such generators. In L'Ecuyer and Proulx (1989), a *pseudo-random number generator* (PRNG) is defined as a family $\{G_n,\ n \geq 1\}$ of structures, increasing in size. In this paper, we adopt a simplified definition, in which we fix the size (as is always the case in practice). We simply use the term *generator*.

DEFINITION 1. A *Generator* is a structure $G = (S, \mu, f, U, g)$, where S is a finite set of *states*, μ is a probability distribution on S, called the *initial distribution*, U is a finite set of *output* symbols, $f : S \rightarrow S$ is the *transition function* and $g : S \rightarrow U$ is the *output function*.

A generator operates as follows:

(1) Select the initial state $s_0 \in S$ according to μ; let $u_0 := g(s_0)$;

(2) for $i := 1, 2, \ldots$, let $s_i := f(s_{i-1})$ and $u_i := g(s_i)$.

The sequence of *observations* $(u_0, u_1, u_2, \ldots)$ is the output of the generator. The initial state s_0 is called the *seed*. We assume that efficient procedures are available to compute f and g and to generate the seed s_0 according to μ. Of course, the aim of a generator will be to output a much longer sequence than its input seed s_0. The output sequence should also *look* to some extent as if the u_i's were the values of i.i.d. random variables, uniformly distributed over U. In practice, this should be supported by a sound theoretical basis and assessed empirically by powerful statistical tests (see sections 3 and 5).

2.3. PT-perfect generators

An *ideal* generator would be such that nobody, using reasonable computing resources and reasonable time, could distinguish between the generator's output and a sequence of truly i.i.d. uniform variates over U better than by flipping a fair coin to guess which is which. Note that this is reminiscent of Turing's test for intelligence.

L'Ecuyer and Proulx (1989) (and other references given there) give a more precise definition, based on computational complexity, which applies to a family $\{G_n,\ n \geq 1\}$ of generators. Informally, the family is called *PT-perfect* if G_n "runs" in polynomial-time (in n) and if no polynomial-time (in n) statistical test can distinguish the output of the generator from a truly random sequence (or equivalently, no polynomial-time algorithm can predict u_{i+1} from $(u_0, \ldots, u_i)$ better than by picking a value uniformly from U.

The generators mostly used in simulation (linear congruential, multiple recursive, GFSR, ...) are *not* PT-perfect. Efficient algorithms have been designed to infer their sequence by looking at the first few numbers. See the references in L'Ecuyer and Proulx (1989). But in practice, they remain the most useful generators for simulation. They are efficient and show good statistical behavior with respect to most reasonable empirical tests, especially when combined.

Binary (or m-ary) expansions of algebraic numbers (roots of polynomials with integral coefficients) or of some transcendental numbers (including π) do not define either PT-perfect generators. Kannan et al. (1988) give efficient algorithms to compute further digits given a long enough initial segment of the expansion.

PT-perfect generators were introduced by researchers in cryptology. These people proposed various generators that are *conjectured* to be PT-perfect. Typically, these generators are much too slow for simulation use. Also, the very existence of any PT-perfect generator has not been proven.

3. GENERATORS ON FINITE FIELDS

Most generators used in practice are in fact based on finite field arithmetic (Knuth (1981), Lidl and Niederreiter (1986), Niederreiter (1986)). Typically, they are special cases or variants of the following matrix formulation.

3.1. Matrix congruential generators

Let $GF(n)$ denote the Galois field with n elements (usually identified with the set of integers $\{0,1,\ldots,n-1\}$). Let $m = p^e$ be some integer which is a power of a prime p, and S be the set of k-dimentional vectors with components in $GF(m)$, i.e. $S = \{X = (x_1,\ldots,x_k)' \mid x_i$ integer and $0 \le x_i < m$ for $1 \le i \le k\}$. Let $A = (a_{ij})$ be a $k \times k$ matrix with elements in $GF(m)$. It defines a linear transformation $f : S \to S$ by $f(X) = AX \bmod m$ (where the mod operation is taken elementwise). Let S^* be the set obtained by removing the vector 0 from S and let μ be an initial distribution on S^*. Choosing U and $g : S \to U$ defines a generator (we will examine ways to do that later on). Here, the generator's state evolves as

$$X_n := AX_{n-1} \bmod m, \qquad (1)$$

with X_0 being the initial state (or seed).

When m is prime ($e = 1$), the maximal possible period for the X_n's is the cardinality of S^*, i.e. $r = m^k - 1$. It is attained iff A is a primitive element of $GF(m^k)$ (i.e. iff all powers of A in arithmetic mod m, plus the matrix 0, generate a vector space with m^k elements, isomorphic to $GF(m^k)$). A sufficient condition is that the characteristic polynomial

$$\begin{aligned} f(x) &= |xI - A| \bmod m \\ &= \left(x^k - \sum_{i=1}^{k} a_i x^{k-i}\right) \bmod m, \qquad (2) \end{aligned}$$

where the coefficients a_i are elements of $GF(m)$, is a primitive polynomial modulo m. Let $r = (m^k-1)/(m-1)$. Knuth (1981) gives the following *sufficient* conditions for $f(x)$ to be primitive modulo m:

(a) $((-1)^{k+1}a_k)^{(m-1)/q} \bmod m \neq 1$ for each prime factor q of $m-1$;

(b) $((x^r \bmod f(x)) \bmod m) = ((-1)^{k+1}a_k) \bmod m$;

(c) $((x^{r/q} \bmod f(x)) \bmod m)$ has degree > 0 for each prime factor q of r, $1 < q < r$. ∎

For large values of m^k, factorizing r is often very hard. It becomes the bottleneck in checking the above conditions (L'Ecuyer and Blouin (1988a, 1988b)). A good idea then is too seek couples (m,k) such that r is prime, since checking primality is much easier than factoring (Morain (1988)). Given m, k and the factorizations of $m-1$ and r, it is relatively easy to find maximal period generators, simply by random search for proper a_i's. For prime m, there are exactly

$$N(m,k) = \frac{(m^k-1)(1-1/q_1)(1-1/q_2)\cdots(1-1/q_h)}{k}$$

choices of $(a_1,\ldots,a_k)$ that satisfy the above sufficient conditions, where $q_1,\ldots,q_h$ are the distinct prime factors of m^k-1 (Knuth (1981)).

If m is not necessarily prime ($m = p^e$ for $e \ge 1$), the maximal possible period is $(p^k-1)p^{e-1}$, except for $p = 2$ and $k = 1$, where it is 2^{e-2} (Knuth (1981)). Eichenauer-Herrmann et al. (1989) give sufficient conditions under which this period is attained and describe a simple method for constructing matrices A giving maximal period generators. (The exception $p - 1 = k = 1$ is treated in Knuth (1981).) The case where $p = 2$ can have some interest in terms of implementation, but the cost in terms of period length, for a given approximate size of m, is important. For example, for $p = 2$ and $k = 1$, the maximal period is about one-fourth of $m^k - 1$. For $m = 2^{31}$ and $k = 5$, the longest possible period is $(2^5-1)2^{31-1} = 2^{35} - 2^{30}$, while $m^k - 1 = 2^{155} - 1$ is about 2^{120} times longer ! For this reason, it is recommend to use only prime values of m.

It is possible, though, to obtain a period length of m^k by using a slightly more general class of transition functions, of the from

$$X_n := (AX_{n-1} + C) \bmod m, \qquad (3)$$

where $C \in S$ is a constant vector. Equation (3) defines a *linear congruential generator* (LCG) in matrix form.

When $C = 0$ (as above), the generator is called *multiplicative* (MLCG). The most widely used generators in practice are multiplicative with prime m. Indeed, using $C \neq 0$ is interesting only when $e > 1$ (e.g. if m is a power of 2). Maximal period conditions for the case $k = 1$ and $C \neq 0$ are given in Knuth (1981). Of course, the case $k = 1$ corresponds to the usual linear congruential generators.

3.2. Multiple recursive generators

For a given prime m, whether a MLCG has full period or not depends only on the characteristic polynomial of its matrix. Any polynomial of the form (2) has a *companion* matrix

$$A = \begin{pmatrix} 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ a_k & a_{k-1} & \dots & a_1 \end{pmatrix} \qquad (4)$$

whose $f(x)$ is the characteristic polynomial. When the matrix A has this special structure, the first $k-1$ components of X_n are obtained by shifting the last $k-1$ components of X_{n-1}, and the last component of X_n is a linear combination of the components of X_{n-1}. This can be viewed as producing a sequence of integers, each one defined as a linear combination modulo m of the k previous ones. This kind of generator is called *multiple recursive* (MRG). With a matrix of this form, and denoting

$$X_n = (x_n, \dots, x_{n+k-1})', \qquad (5)$$

equation (1) is equivalent to the recursion

$$x_n := (a_1 x_{n-1} + \dots + a_k x_{n-k}) \bmod m. \qquad (6)$$

Restricting our search to generators of this class is certainly supported by their ease of implementation. It is further reinforced by the following property (Grothe (1988), L'Ecuyer and Blouin (1988b), Niederreiter (1986)): for any generator defined by (1) with the characteristic polynomial of A defined by (2), the sequence of states obeys the recursion

$$X_n := (a_1 X_{n-1} + \dots + a_k X_{n-k}) \bmod m. \qquad (7)$$

In other words, each component of X_n evolves according to the same recursion (6), which means that we just have in fact k copies of the same MRG evolving in parallel (hopefully, with different and "far apart" seeds). This gives a good argument supporting the direct use of (6).

Another interesting special case in terms of implementation is when the characteristic polynomial $f(x)$ is only a trinomial, of the form $f(x) = x^k - a_j x^{k-j} - a_k$, for $1 \leq j < k$. Primitive trinomials of this form are easy to find (L'Ecuyer and Blouin (1988a, 1988b)). The corresponding recursion can be written

$$X_n := (a_j X_{n-j} + a_k X_{n-k}) \bmod m. \qquad (8)$$

The generator can be implemented directly in this form, but then, its state should be redefined as $s_n = (X_{n-1}, \dots, X_{n-k})$ (which is a matrix), or as the vector $(x_{n-1}, \dots, x_{n-k})$ in the case of the MRG.

Still more interesting is when m is a "representable" integer on the target computer and $a_i(m \bmod a_i) < m$ for each i, because then, there is an efficient and easily implementable way to compute $a_i x \bmod m$ (Bratley et al. (1987), L'Ecuyer (1988), Park and Miller (1988)). Generators of this form are proposed by L'Ecuyer and Blouin (1988a, 1988b). Techniques for computing $a_i x \bmod m$ in a high level language for the more general case are studied in L'Ecuyer and Côté (1987).

Jumping ahead in the sequence of a MLCG can be done efficiently using

$$X_{n+j} = (A^j X_n) \bmod m = (A^j \bmod m) X_n \bmod m.$$

The matrix $(A^j \bmod m)$ can be precomputed using the divide-to-conquer algorithm:

$$A^j \bmod m = \begin{cases} A & \text{if } j = 1; \\ A \times A^{j-1} \bmod m & \text{if } j > 2,\ j \text{ odd}; \\ A^{j/2} \times A^{j/2} \bmod m & \text{if } j > 1,\ j \text{ even}. \end{cases}$$

Such "jumping ahead" facilities are useful for some simulation applications (Bratley et al. (1987), L'Ecuyer and Côté (1987)). For the maximal period to be attained, A must be nonsingular in arithmetic modulo m, since otherwise $AX \bmod m = 0$ for some vector $X \neq 0$. Then, if A^{-1} denotes its inverse, we have $X_{n-1} = A^{-1} X_n \bmod m$, so that the sequence can be generated in reverse order.

3.3. Tausworthe, GFSR, lagged-Fibonacci

When $a_j = |a_k| = 1$, the recursion (8) is a special case of the so-called *lagged-Fibonacci* generator (LFG). A LFG is defined by

$$X_n := (X_{n-j} \diamond X_{n-k}) \bmod m \qquad (9)$$

where $\diamond$ is any componentwise binary operation (sum, product, subtraction, etc.) and X_n is a vector of any size, with components in $GF(m)$. These generators are analysed in Marsaglia (1985) and Marsaglia and Tsay (1985) for different operators $\diamond$ and $m = 2^e$. For such values of m, their maximal period lengths are typically much smaller than $m^k - 1$.

Increasing the period of a LCG can be done by taking a larger m or a larger k. Typical MLCGs use m near 2^{31} and small k. At the other extreme lies the case $m = 2$ (with large k). In this case, X_n is a vector of k bits. For $b \leq k$, one can interpret, say, the last b bits of X_n as a b-bit integer. The generator thus obtained is called *Generalized Feedback Shift Register* (GFSR) (Fushimi (1988), Fushimi and Tezuka (1983), Lewis and Payne (1973), Marsaglia and Tsay (1985)). For the "efficient" special case (8), with $a_j = a_k = 1$, it becomes a special kind of lagged-Fibonacci generator, with operator $\diamond$ denoting the bitwise XOR. Since the first $k - b$ bits (if any) are unused, X_n can be viewed as a b-bit vector. The generator's state s_n is a $b \times k$ matrix of bits. Examples of suggested values in practice are say $b = 31$ and k somewhere between 64 and 512.

For $b = 1$, one gets a MRG called a *Tausworthe* (or simple shift register) generator, which produces a sequence of bits (also called a M-sequence, for maximal period generators). Tausworthe (1965) suggested regrouping blocks of successive bits to form integers of reals. These generators are rather slow and are almost not used any more in practice. GFSR generators are faster but use more memory. Since a GFSR generator corresponds in fact to b copies of the same Tausworthe generator evolving in parallel, efficient algorithms are needed to compute an initial matrix of bits so that these b bit-generators have their seeds far enough apart (Arvillias and Maritsas (1978), Collings and Hembree (1986)). Marsaglia and Tsay (1985) and Marsaglia (1985) argue strongly against the use of GFSR generators, for efficiency reasons and because thay fail miserably some statistical tests.

3.4. Transforming into $U(0,1)$ variates

Coming back to the general MLCG (1), there are different ways of using the state vector $X_n = (x_{n1}, \ldots, x_{nk})'$ to produce real values between 0 and 1 (i.e. to define U and g). When m is large, a component x_{ni} can be simply divided by m, yielding a result in $[0, 1)$. But it is often necessary to make sure that the result lies *strictly* between 0 and 1. This can be accomplished by dividing instead by $m + 1$, replacing first x_{ni} by m when $x_{ni} = 0$. Other slightly more involved techniques are proposed in Marse and Roberts (1983) and Monahan (1985).

Afflerbach and Grothe (1988) and Grothe (1987, 1988) use *all* the components of X_n to obtain k $U(0,1)$ variates at each iteration. L'Ecuyer and Blouin (1988b) use only x_{nk} (the last component), which is equivalent to using a MRG. The former could eventually generate variates at a faster rate using parallel processors. Another approach in that case could also be to run *different* MRGs in parallel.

When dividing x_{ni} by $(m + 1)$, the *mesh size* (or "granularity") of the output is $1/(m + 1)$. For some applications, a smaller mesh size might be necessary (see e.g. the example in Eichenauer and Lehn (1986)). One can then use a *digital method*, in which a value $u_n \in (0, 1)$ is produced by

$$u_n = p^{-t} + \left(\left(\sum_{i=1}^{t} p^{-i} x_{tn+i-1} \right) \bmod 1 \right),$$

where $p \leq m + 1$ and $t \geq 1$ are integers (p could be for instance a power of two), and $\{x_i,\ i \geq 0\}$ is the sequence of all used vectors components (or a sequence produced by (6)). Other variants are discussed in Niederreiter (1986, 1988). Tausworthe and GFSR generators use a similar technique with $p = 2$. In the MRG case, the period of the u_n's always divides $m^k - 1$. When $m^k - 1$ and t are relatively prime, it is almost always $m^k - 1$.

3.5. A class of generators by inversion

Eichenauer and Lehn (1986) and Eichenauer et al. (1987) introduced a class of "non-linear" generators in which the sequence $\{x_n,\ n \geq 0\}$ obeys (6). Let $\tilde{x}_i$ be the i-th non-zero value x_n in that sequence. Define $z_n = (\tilde{x}_{n+1}\tilde{x}_n^{-1}) \bmod m$, where $\tilde{x}_n^{-1}$ denotes the inverse element of $\tilde{x}_n$ in $GF(m)$. The z_n's can be used to produce the u_n's exactly as the x_n's were in the previous subsection. A version of Euclid's algorithm, whose average running time is approximately $12(\ln 2)(\ln m)/\pi$ ((Knuth (1981)), can be used to compute the inverse $\tilde{x}_n^{-1}$.

For prime m, the maximal possible period length for the z_n's is m^{k-1}. Eichenauer et al. (1987) give sufficient conditions for it to be attained. Maximal period generators are easy to find. For $k = 2$ or 3, one can write a recursion directly for the z_n's. For $k = 2$, it is

$$z_n = \begin{cases} (a_1 + a_2 z_{n-1}^{-1}) \bmod m & \text{if } z_{n-1} \neq 0; \\ a_1 & \text{if } z_{n-1} = 0. \end{cases}$$

The main motivation behind these generators is that the sequence they produce does not share the lattice property of the usual LCGs (next subsection).

3.6. Lattice structure and spectral test

Consider a maximal period MRG, of the form (6), and let

$$T_t = \{(x_n, \ldots, x_{n+t-1}),\ n \geq 0\} \cup \{0\}$$

be the set of all *overlapping* t-tuples of successive values, plus the zero vector. It is well known (Grube (1973), Knuth (1981), L'Ecuyer and Blouin (1988b), Marsaglia (1968)) that the periodic continuation of T_t with period m,

$$L_t = T_t + m\mathbb{Z}^t,$$

forms a lattice with unit cell volume of $\max(1, m^{k-t})$. Recall that a t-dimensional lattice is a set of the form

$$L = \left\{ \sum_{i=1}^{t} z_i V_i, \quad \text{each } z_i \text{ integer} \right\}$$

where $V_1, \ldots, V_t$ is a set of linearly independent vectors called a basis. A set of vectors $W_1, \ldots, W_t$ such that the scalar products obey $V_i \cdot W_j = \delta_{ij}$ form a basis of the *dual* lattice.

For $t \leq k$, the lattice contains all possible integer vectors and the unit cell volume is one. For $t = k$, each vector except the zero vector occurs once and only once over the period. For $t > k$, the unit cell volume can be huge compared to 1, which is the value that one would expect from truly random integer vectors. This can be viewed as a strong limitation of simple LCGs (with $k = 1$) and suggests using larger values of k. A unit cell of the lattice is determined by the vectors of a *Minkowski-reduced lattice base* (MRLB) (Afflerbach and Grothe (1985, 1988), Grothe (1988)). It is traditionally accepted that "better" generators are obtained when the unit cells of the lattice are more "cubic-like", i.e. when the vectors of the MRLB have about the same size. The ratio of the sizes of the shortest and longest vectors of the MRLB is called the *Beyer-quotient*. It can be used to assess the quality of the lattice. Values near one are said to be more desirable. Note however that reducing the unit cell volume (by increasing m, or k, or both) can be much more effective in improving the quality than getting a larger Beyer-quotient with fixed m and k. Afflerbach and Grothe (1985) and Grothe (1988) give efficient algorithms to compute the MRLB and Beyer-quotient of a given lattice.

The lattice structure also means that all points of T_t lie in a family of equidistant parallel hyperplanes. Among all such families of hyperplanes that cover all the points, choose the one for which the successive hyperplanes are farthest apart, and let d_t be the distance between them. The smaller that distance, the better, since this implies thinner empty "slices" in the lattice. Dieter (1975) and Knuth (1981) give an algorithm to compute d_t, which is in fact equal to the length of the shortest vector in the *dual* lattice to L_t (it is also the shortest vector in the MRLB of that dual lattice). For given m and k, the number of hyperplanes in the chosen family cannot exceed $(t!(m^k - 1))^{1/t}$ and there is also a theoretical lower bound d_t^* on d_t. One can define the figures of merit $S_t = d_t^*/d_t$ and $M_t = \min_{k \leq i \leq t} S_i$, which lie between 0 and 1. For $k = 1$ and using M_6 as a criterion, computer searchs to find good generators have been done by Fishman and Moore (1986) (for $m = 2^{31} - 1$), L'Ecuyer (1988) (for different values of m near 2^{31} and $a_1^2 < m$), and by Park and Miller (for $m = 2^{31} - 1$ and $a_1(m \bmod a_1) < m$). More extensive searchs using M_8 as a criterion, for $1 \leq k \leq 7$ and different values of m, up to near 2^{127}, have been performed by L'Ecuyer and Blouin (1988a, 1988b).

A similar lattice structure appears when all components of X_n are used at each iteration (Afflerbach and Grothe (1988) and Grothe (1987, 1988)). It can be analyzed in a similar way. When the generator is *not* multiplicative ($C \neq 0$), the lattice is shifted by a constant vector, yielding what is called a *grid* (which can be analyzed in the same way). When T_t is replaced by the set of *non-overlapping* t-tuples, L_t does not form a lattice in general (Afflerbach (1986)).

LCGs can be generalized to *quadratic* generators of the form

$$X_n := (X'_{n-1} A X_{n-1} + B X_{n-1} + C) \bmod m,$$

where A and B are $k \times k$ matrices, or more generally to

$$X_n := P(X_{n-1}) \bmod m$$

where P is some multivariate polynomial. For $k = 1$, quadratic generators are analyzed in Knuth (1981) and Eichenauer and Lehn (1987). The latter authors show that for maximal period generators ($\tau = m^k$), the *non-overlapping* t-tuples determine a union of grids.

4. COMBINED GENERATORS

To get rid of the regular patterns displayed by LCGs, it has often been suggested to combine different generators to produce a "hybrid" one, with the aim of getting better "scrambling" (Collings (1987), Haas (1987), Knuth (1981), L'Ecuyer (1988), Marsaglia (1985), Nance and Overstreet (1978), Wichmann and Hill (1982, 1987)).

Such combination is often viewed as completely heuristic and sometimes discouraged. Ripley (1988), for instance, associates it to a "better the unknown than the devil we know" attitude. But the fact is that besides being strongly supported by many empirical investigations, combination *has* theoretical support. Firstly, in most cases, the period of the hybrid is much longer than that of each of its components, and can be computed. Secondly, there are theoretical results suggesting that some forms of combined generators generally have better statistical behavior. For instance, suppose two random sequences $\{x_n, n \geq 0\}$ and $\{y_n, n \geq 0\}$ are combined elementwise to form a third sequence $\{z_n, n \geq 0\}$, where $z_n = x_n \diamond y_n$ and $\diamond$ denotes some binary operator. Assume that the three sequences are defined over the same finite set. Then, under fairly reasonable conditions, the t-tuples of successive values are "more" (or at least as much) uniformly distributed in some sense for the third sequence than for any of its two constituents. See Marsaglia (1985).

An efficient and portable combined generator is proposed in L'Ecuyer (1988). No structure is apparent in its geometric behavior even if the lattice structure of its individual MLCG components can be made visible. Marsaglia (1985) suggests alternative ones. They should be easy to code in some low level languages, but not so easy in high level languages with strong typing. Collings (1987) and Wichman and Hill (1982) suggest others. All these generators were submitted to heavy statistical testing.

Other combination approaches are based on ***shuffling*** (Bratley et al. (1987), Knuth (1981), Law and Kelton (1982), Nance and Overstreet (1978)). In one of the variants, two simple generators are used, one to fill the cells of a buffer and the other to select which cell the next output value will be taken from. At each step, the second generator selects a cell, outputs its content, then the first generator fills it back. Shuffling is not so well understood and have some practical drawbacks (Bratley et al. (1987)). For instance, there is no obvious efficient way to jump ahead in the sequence.

5. STATISTICAL TESTING

Knuth (1981) describes a set of empirical statistical tests, usually viewed as the "standard" ones. Many of them are included in the package of Dudewicz and Ralley (1981). Marsaglia (1985) describes supposedly more powerful ones. Statistical tests are rather easy to design: any function of a finite set of i.i.d. uniform random variables can be used as a statistic to define a test, if its distribution is known. To gain power, the test can be repeated N times, and the empirical distribution of the values of the statistic can be compared to its theoretical distribution, using say the Kolmogorov-Smirnov test (Dudewicz and Ralley (1981), L'Ecuyer (1988)).

Besides empirical tests, some theoretical tests can give information about the statistical behavior of certain generators, often over the full period but sometimes also for just part of the period. Examining the lattice structure of LCGs yields such tests. Other tests are based on the notion of *discrepancy*. Informally, the discrepancy $D_N^{(t)}$ in t dimensions is the absolute difference between the expected number and actual number of vectors $(x_n, \ldots, x_{n+t-1})$, $0 \leq n < N$, falling into a hyper-rectangular region, maximized over all such regions (or in some definitions, over those regions with a corner at the origin). For many different classes of generators, Niederreiter (1978, 1986, 1987, 1988, 1989), Mullen and Niederreiter (1987) and Tezuka (1988) provide bounds on $D_N^{(t)}$. A low upper bound can give some sort of "protection". But only in rare cases, exact values can be computed. Also, as pointed out by Knuth (1981), the discrepancy is very sensitive to rotations of the axis, in contrast to the Beyer-quotient or spectral test. Furthermore, generators that are good at the latter always have low discrepancy $D_N^{(t)}$ for large enough N. This suggests that rating generators on the basis of their discrepancy bounds is not necessarily the best idea.

6. CONCLUSION

A lot has been written on uniform variate generation, but certainly, the last word has not been said. MRGs with a trinomial characteristic function, large m and say $k \geq 5$, appear to be an excellent choice in terms of efficiency and statistical quality. The unit cell volume of the associated t-dimensional lattice, for $t > k$, can be reduced by increasing k. The mesh size can be reduced without increasing m by using the digital method. Combination can also be used for further "scrambling". Note that the digital method can be implemented using *different* MRGs (evolving in parallel) for different digits.

PT-perfect generators offer a good stimulus for further research. As computing power is getting cheaper and cheaper, applications will require increasingly robust generators. They should be submitted to (new) more powerful statistical tests.

ACKNOWLEDGEMENTS

This work has been supported by NSERC-Canada grant # A5463 and FCAR-Quebec grant # EQ2831. It was performed while the author was visiting the Laboratoire d'Automatique de Nantes (U.A.CNRS no. 823), ENSM, Nantes, France.

REFERENCES

Afflerbach, L. (1986). The Sub-Lattice Structure of Linear Congruential Random Number Generators. *Manuscripta Math.*, **55**, 455–465.

Afflerbach, L. and Grothe, H. (1985). Calculation of Minkowski-Reduced Lattice Bases. *Computing*, **35**, 269–276.

Afflerbach, L. and Grothe, H. (1988). The Lattice Structure of Pseudo-Random Vectors Generated by Matrix Generators. *J. of Computational and Applied Math.*, **23**, 127–131.

Arvillias, A. C. and Maritsas, D. G. (1978). Partitioning the Period of a Class of m-Sequences and Application to Pseudorandom Number Generation. *J. of the ACM*, **25**, 4, 675–686.

Bratley, P., Fox, B. L. and Schrage, L. E. (1987). *A Guide to Simulation*, second edition. Springer-Verlag, New York.

Chor, B. and Goldreich, O. (1988). Unbiased Bits From Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM J. on Computation*, **17**, 2, 230–261.

Collings, B. J. (1987). Compound Random Number Generators. *J. of the American Statistical Association*, **82**, 398, 525–527.

Collings, B. J. and Hembree, G. B. (1986). Initializing Generalized Feedback Shift Register Pseudorandom Number Generators. *J. of the ACM*, **33**, 706–711.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

Dieter, U. (1975). How to Calculate Shortest Vectors in a Lattice. *Math. of Computation*, **29**, 131, 827–833.

Dudewicz, E. J. and Ralley, T. G. (1981). *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code.* American Sciences Press, Columbus, Ohio.

Eichenauer, J. and Lehn, J. (1986). A Nonlinear Congruential Pseudorandom Number Generator. *Statistische Hefte*, **27**, 315–326.

Eichenauer, J. and Lehn, J. (1987). On the Structure of Quadratic Congruential Sequences. *Manuscripta Math.*, **58**, 129–140.

Eichenauer, J., Lehn, J. and Topuzoğlu, A. (1988). A Nonlinear Congruential Pseudorandom Number Generator with Power of Two Modulus. *Math. of Computation*, **51**, 184, 757–759.

Eichenauer, J., Grothe, H., Lehn, J. and Topuzoğlu, A. (1987). A Multiple Recursive Nonlinear Congruential Pseudorandom Number Generator. *Manuscripta Math.*, **59**, 331–346.

Eichenauer, J. and Niederreiter, H. (1988). On Marsaglia's Lattice Test for Pseudorandom Numbers. *Manuscripta Math.*, **62**, 245–248.

Eichenauer-Herrmann, J., Grothe, H. and Lehn, J. (1989). On the Period Length of Pseudorandom Vector Sequences Generated by Matrix Generators. *Math. of Computation*, **52**, 185, 145–148.

Fishman, G. S. and Moore III, L. S. (1986). An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus $2^{31} - 1$. *SIAM J. on Scientific and Statistical Computing* **7**, 1, 24–45.

Fushimi, M. (1988). Designing a Uniform Random Number Generator Whose Subsequences Are k-Distributed. *SIAM J. on Computing*, **17**, 1, 89–99.

Fushimi, M. and Tezuka, S. (1983). The k-Distribution of Generalized Feedback Shift Register Pseudorandom Numbers. *Communications of the ACM*, **26**, 7, 516–523.

Goldreich, O., Goldwasser, S. and Micali, S. (1986). How to Construct Random Functions. *J. of the ACM*, **33**, 4, 792–807.

Grothe, H. (1987). Matrix Generators for Pseudo-Random Vectors Generation. *Statist. Hefte*, **28**, 233–238.

Grothe, H. (1988). Matrixgeneratoren zur Erzeugung gleichverteilter Pseudozufallsvektoren (in german). Dissertation (thesis), Tech. Hochschule Darmstadt, Germany.

Grube, A. (1973). Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen (in german). *Zeitschrift für angewandte Math. und Mechanik* **53**, T223–T225.

Haas, A. (1987). The Multiple Prime Random Number Generator. *ACM Trans. on Math. Software*, **13**, 4, 368–381.

Kannan, R., Lenstra, A. K. and Lovász, L. (1988). Polynomial Factorization and Nonrandomness of Bits of Algebraic and Some Transcendental Numbers. *Math. of Computation*, **50**, 181, 235–250.

Knuth, D. E. (1981). *The Art of Computer Programming : Seminumerical Algorithms*, vol. 2, second edition. Addison-Wesley.

Law, A. M. and Kelton, W. D. (1982). *Simulation Modeling and Analysis*, McGraw-Hill.

L'Ecuyer, P. (1987). A Portable Random Number Generator for 16-Bit Computers. *Modeling and Simulation on Microcomputers 1987*. The Society for Computer Simulation, 45–49.

L'Ecuyer, P. (1988). Efficient and Portable Combined Random Number Generators. *Communications of the ACM*, **31**, 6, 742–749 and 774.

L'Ecuyer, P. and Blouin, F. (1988a). Linear Congruential Generators of Order $k > 1$. *Proceedings of the 1988 Winter Simulation Conference*, 432–439.

L'Ecuyer, P. and Blouin, F. (1988b). Generalized Linear Congruential Generators. Report no. DIUL-RR-8814, Computer Science Dept., Laval University. (Also presented at ORSA/TIMS St-Louis, 1987).

L'Ecuyer, P. and Côté, S. (1987). A Random Number Package with Splitting Facilities. Report no. DIUL-RR-8705, dépt. d'informatique, Univ. Laval (submitted to *ACM Trans. on Math. Software*).

L'Ecuyer, P. and Proulx, R. (1989). About Polynomial-Time "Unpredictable" Generators. In these *Proceedings*.

Lewis, T. G. and Payne, W. H. (1973). Generalized Feedback Shift Register Pseudorandom Number Algorithm. *J. of the ACM*, **20**, 3, 456–468.

Lidl, R. and Niederreiter, H. (1986). *Introduction to Finite Fields and Their Applications*. Cambridge University Press, Cambridge.

Marsaglia, G. (1968). Random Numbers Fall Mainly in the Planes. *Proceedings of the National Academy of Sciences of the United States of America* **60**, 25–28.

Marsaglia, G. (1985). A Current View of Random Number Generation. *Computer Science and Statistics, Proceedings of the Sixteenth Symposium on the Interface*, Atlanta, march 1984. Elsevier Science Publ. (North-Holland), 1985, 3–10.

Marsaglia, G. and Tsay, L.-H. (1985). Matrices and the Structure of Random Number Sequences. *Linear Algebra and its Applications*, **67**, 147–156.

Marse, K. and Roberts, S. D. (1983). Implementing a Portable FORTRAN Uniform (0,1) Generator. *Simulation*, **41**, 4, 135–139.

Modianos, D. T., Scott, R. C. and Cornwell, L. W. (1987). Testing Intrinsic Random Number Generators. *Byte*, **12**, 1, 175–178.

Monahan, J. F. (1985). Accuracy in Random Number Generation. *Math. of Computation*, **45**, 172, 559–568.

Morain, F. (1988). Implementation of the Atkin-Goldwasser-Kilian Primality Testing Algorithm. Rapport de recherche no. 911, INRIA, Rocquencourt, France.

Mullen, G. L. and Niederreiter, H. (1987). Optimal Characteristic Polynomials for Digital Multistep Pseudorandom Numbers. *Computing*, **39**, 155–163.

Nance, R. E. and Overstreet, C., Jr. (1978). Some Experimental Observations on the Behavior of Composite Random Number Generators. *Operations Research*, **26**, 5, 915–935.

Narkiewicz, W. (1984). *Uniform Distribution of Sequences of Integers in Residue Classes*. Lecture Notes in Mathematics, No. 1087, Springer-Verlag.

Niederreiter, H. (1978). Quasi-Monte Carlo Methods and Pseudorandom Numbers. *Bull. Amer. Math. Soc.*, **84**, 6, 957–1041.

Niederreiter, H. (1986). A Pseudorandom Vector Generator Based on Finite Field Arithmetic. *Math. Japonica*, **31**, 5, 759–774.

Niederreiter, H. (1987). A Statistical Analysis of Generalized Feedback Shift Register Pseudorandom Number Generators. *SIAM J. Sci. Stat. Comput.*, **8**, 6, 1035–1051.

Niederreiter, H. (1988). The Serial Test for Digital k-Step Pseudorandom Numbers. *Mathematical Journal of Okayama University*, **30**, 93–119.

Niederreiter, H. (1989). The Serial Test for Congruential Pseudorandom Numbers Generated by Inversions. *Math. of Computation*, **52**, 185, 135–144.

Park, S. K. and Miller, K. W. (1988). Random Number Generators: Good Ones Are Hard to Find. *Communications of the ACM*, **31**, 10, 1192–1201.

Reif, J. H. and Tygar, J. D. (1988). Efficient Parallel Pseudorandom Number Generation. *SIAM J. Computing*, **17**, 2, 404–411.

Ripley, B. D. (1988). Uses and Abuses of Statistical Simulation. *Mathematical Programming*, **42**, 53–68.

Tausworthe, R. C. (1965). Random Numbers Generated by Linear Recurrence Modulo Two. *Math. of Computation*, **19**, 201–209.

Tezuka, S. (1987). Walsh-Spectral Test for GFSR Pseudorandom Numbers. *Communications of the ACM*, **30**, 8, 731–735.

Tezuka, S. (1988). On Optimal GFSR Pseudorandom Number Generators. *Math. of Computation*, **50**, 182, 531–533.

Wichmann, B. A. and Hill, I. D. (1982). An Efficient and Portable Pseudo-random Number Generator. *Applied Statistics*, **31**, 188–190. See also corrections and remarks in the same journal by Wichmann and Hill (1984), **33**, 123; McLeod (1985), **34**, 198–200; Zeisel (1986), **35**, 89.

Wichmann, B. A. and Hill, I. D. (1987). Building a Random Number Generator. *Byte*, **12**, 3, 127–128.

AUTHOR'S BIOGRAPHY

PIERRE L'ECUYER is an associate professor in the Computer Science Department at Laval University, Ste-Foy, Québec, Canada. He received the B.Sc. degree in mathematics in 1972, and was a college teacher in mathematics from 1973 to 1978. He then received the M.Sc. degree in operations research and the Ph.D. degree in computer science, in 1980 and 1983 respectively, both from the University of Montreal. His research interests are in Markov renewal decision processes, sensitivity analysis and optimization of discrete-event stochastic systems, random number generation, and discrete-event simulation software. He is a member of ACM, IEEE, ORSA and SCS.

Pierre L'Ecuyer
Département d'informatique
Pavillon Pouliot
Université Laval
Ste-Foy, G1K 7P4
Canada
(418) 656-3226