Version: August 27, 1996
To appear in: *INFORMS Journal on Computing*

An Implementation of the Lattice and Spectral Tests for Multiple Recursive Linear Random Number Generators

PIERRE L'ECUYER AND RAYMOND COUTURE /
Département d'Informatique et de Recherche Opérationnelle (IRO), Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada;
e-mail: lecuyer@iro.umontreal.ca; couture@iro.umontreal.ca

We discuss the implementation of theoretical tests to assess the structural properties of simple or combined linear congruential and multiple recursive random number generators. In particular, we describe a package implementing the so-called spectral and lattice tests for such generators. Our programs analyze the lattices generated by vectors of successive or non-successive values produced by the generator, analyze the behavior of generators in high dimensions, and deal with moduli of practically unlimited sizes. We give numerical illustrations. We also explain how to build lattice bases in several different cases, e.g., for vectors of far-apart non-successive values, or for sublattices generated by the set of periodic states or by a subcycle of a generator, and, for all these cases, how to increase the dimension of a (perhaps partially reduced) basis.

Subject classifications: random number generation; linear congruential generators; lattice structure; spectral test

The aim of this paper is to explain the implementation of a software package for analyzing the lattice structure of linear congruential or multiple recursive random number generators. Such generators are based on linear recurrences of the form

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

It is well-known that the set of all vectors of successive values of the form $(x_n, \ldots, x_{n+t-1})$, obeying this recurrence, is the intersection of a t-dimensional integer lattice L_t with the hypercube $[0, m)^t$. A *lattice* is the set of all integer linear combinations of a familly of t linearly independent vectors. These vectors form a *basis* of this lattice. Among all possible bases of a given lattice are the Minkowski-reduced bases. They are comprised of vectors "as short as possible" in a specific sense. We give further details in Section 1.2. The Beyer quotient of a Minkowski-reduced basis is defined as the ratio of the lengths of the shortest to that of the longest vector in this basis. The upper bound q_t of the Beyer quotients of all Minkowski-reduced bases of a lattice gives an indication of the "quality" of this lattice: a q_t close to 1 means that its points are more evenly distributed [2, 3, 11].

The lattice structure implies that all the points $(x_n, \ldots, x_{n+t-1})$ lie on a relatively small family of equidistant parallel hyperplanes, especially for large values of t [2, 15, 17, 21, 18, 27, 28, 29]. The more distant are contiguous hyperperplane in such a familly, the more this conflicts with the idea that the sequence $\{x_n\}$ should imitate a sample of independent values generated from the uniform distribution. As a result, the maximum d_t of the distances associated with all such families of hyperplanes is widely adopted as perhaps the most significant figure of merit for ranking these generators [15, 17, 18]. If we define the dual of a lattice as the lattice with basis equal to the dual of a basis of the given lattice, then, computing d_t is equivalent to finding the length of the shortest non-zero vector in the lattice dual to L_t [7, 12]. References [2, 5, 8] address the problems of computing the shortest vector in a lattice and of finding a Minkowski-reduced basis.

In the next section, we recall some basic facts about linear congruential and multiple recursive generators and their lattice structure. We outline an easy way to construct a lattice basis and its dual for full period generators. We also comment on the relative pertinence of d_t and q_t. In Section 2, we explain how to constuct a basis for the t-dimensional lattice associated with a given generator by using any basis for the correspondig $t-1$-dimensional lattice. Section 3 deals with lacunary (or "leap frog") indices, i.e., with analyzing the lattice spanned by the vectors of the form $(x_{n+i_1}, x_{n+i_2}, \ldots, x_{n+i_t})$, for $n \geq 0$, where $i_1, \ldots, i_t$ are any fixed positive integers. Analyzing the lattice structure of vectors formed by such non-successive values, which are some fixed distance apart in the sequence, is useful for studying certain (long range) correlations between disjoint segments of the sequence. This is particularly important when the generator's sequence is split into several subsequences in order to obtain many virtual (or parallel) generators [17, 22]. In Section 4, we consider generators with period length much smaller than the maximum achievable (that is, much smaller than m^k) and with transient states. Generators with these properties are often used because of their higher speed and ease of implementation. It may happen that the lattice generated by the set of periodic states of such a generator is a strict sublattice of that generated by all possible initial states. We show how to construct a basis for that sublattice, for either successive or lacunary indices, and give examples. Considering such sublattices is appropriate, in particular, if we want to analyze certain classes of combined generators [25, 20]. In Section 5, we analyze generators of order one (ordinary linear congruential ones) whose modulus is a non-trivial power of a prime. In this case, the generator has different subcycles and the lattice generated by the vectors visited over a given subcycle is often a strict sublattice of L_t, for which we show how to construct a basis. This includes as a special case the moduli that are powers of two. Section 6 gives a succint overview of the software package `LatMRG` [23], while Section 7 gives concrete examples of results. These results show that the package can analyze the lattice structure of a generator in large dimensions in reasonable time. It can deal with arbitrary large moduli and multipliers. It can also perform computer searches for good generators according to figures of merit based on d_t or q_t. A preliminary version of this package was used, for instance, to obtain the results given in [21]. The package is available via ftp (contact the first author).

1. Lattice structure of linear congruential generators

1.1 Multiple recursive generators

Consider the linear recurrence (1), where m and k are positive integers and each a_i belongs to the ring $\mathbf{Z}_m = \mathbf{Z}/m\mathbf{Z}$, whose elements are identified with $\{0, 1, \ldots, m-1\}$. Let S be the set of k-dimensional vectors with coordinates in $\mathbf{Z}_m$. For $n \geq 0$, $s_n = (x_n, \ldots, x_{n+k-1}) \in S$ is the *state* at step n. The initial state s_0 is called the *seed*. Let $u_n = x_n/m \in [0, 1)$ be the *output* at step n. The sequence of output values u_n is often used to imitate a sequence of independent random variables uniformly distributed over the interval $[0, 1)$. This kind of random number generator is called a *multiple recursive generator* (MRG). When $k = 1$, it becomes the well-known multiplicative linear congruential generator (MLCG). MLCG's in *matrix form* can also be expressed as many copies of the same MRG running in parallel. For more details, see [11, 17, 21, 18, 27, 28].

The maximal possible period for the s_n's is $\rho = m^k - 1$, attained if and only if m is prime and the characteristic polynomial of (1), given by

$$f(x) = x^k - \sum_{i=1}^{k} a_i x^{k-i}, \tag{2}$$

is primitive modulo m. Knuth [15] states necessary and sufficient conditions for $f(x)$ to be primitive modulo a prime m. These conditions are given in the form of an algorithm, which is implemented in our package. If $k = 1$ and $m = p^e$, with $e > 1$, then the maximal possible period is 2^{e-2} for $p = 2$ and $(p-1)p^{e-1}$ for $p > 2$ [15].

Combining two or more MLCGs or MRGs with distinct and relatively prime moduli provides an efficient way of implementing a linear recurrence based on a large (non-prime) modulus m. The combined generator is equivalent, or can be approximated (depending on the combination approach), by an associated MLCG or MRG whose modulus is the product of the individual moduli of its components [20, 25]. The package `LatMRG` permits one to analyze combined generators by specifying only their components, and search for "good" combined generators within specified classes. Since these combined generators have composite moduli, their period falls well short of $m^k - 1$. In many cases, they also possess transient states. Then, one can decide to analyze either the lattice associated with all possible initial states, or that generated by the set of recurrent states (see Section 5).

1.2 Lattices

Let t be any positive integer. 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 basis of $\mathbf{R}^t$. This basis is said to be a *lattice basis* of L. If m is any positive real number, we will say that the basis $\{W_1, \ldots, W_t\}$ satisfying $V_i' W_j = \delta_{ij} m$ for all i, j (δ_{ij} is the Kroenecker's delta), is the m-dual of the basis $\{V_1, \ldots, V_t\}$, and that the lattice generated by this m-dual basis, is the m-dual to L. This extension of the usual notion of dual basis, and dual lattice, will allow us, by a suitable choice of m (in our context it will be the modulus in (1)), to deal uniquely with integer coordinate vectors, which can then be represented exactly on the computer.

Consider the sequence x_n produced by the recurrence (1), and the set

$$T_t = \{(x_n, \ldots, x_{n+t-1}) \mid n \geq 0,\ s_0 = (x_0, \ldots, x_{n+k-1}) \in \mathbf{Z}_m^k\} \tag{3}$$

of all *overlapping* t-tuples of successive values, starting from all possible initial seeds. Then, the periodic continuation of T_t with period m,

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

forms a lattice with unit cell volume equal to $\max(1, m^{t-k})$ (see [7, 11, 12, 15, 18, 21, 26]).

Bases for L_t and its dual can be constructed as follows. For $t \leq k$, L_t comprises all t-dimensional integer vectors. For $t > k$, a vector of L_t can have arbitrary first k coordinates; but then, the remaining coordinates are determined (up to a multiple of m) by the recurrence. Let $e_{i(j)}$ denote the i-th unit vector in dimension j. For $i \leq k$, let $V_i = (x_{i,1}, \ldots, x_{i,t})$ be the t-dimensional vector whose first k coordinates are $(x_{i,1}, \ldots, x_{i,k}) = e_{i(k)}$, while the remaining ones are determined by:

$$x_{i,j} \ := \ (a_1 x_{i,j-1} + \cdots + a_k x_{i,j-k}) \bmod m \tag{4}$$

for $j > k$. All the vectors of T_t can be expressed as integer linear combinations of $V_1, \ldots, V_k$, together with "modulo m" operations. From that observation, it follows that $\{V_1, \ldots, V_t\}$ forms a basis for L_t, where

$$V_i = \begin{cases} (x_{i,1}, \ldots, x_{i,t}) & \text{for } i \leq k, \\ m\ e_{i(t)} & \text{for } i > k. \end{cases} \tag{5}$$

The corresponding m-dual basis is $\{W_1, \ldots, W_t\}$, where

$$W_i = \begin{cases} m\ e_{i(t)} & \text{for } i \leq k, \\ e_{i(t)} \ - \ (x_{1,i}, \ldots, x_{k,i}, 0, \ldots, 0) & \text{for } i > k. \end{cases} \tag{6}$$

If one adds a constant b on the right-hand-side of (1), before applying the modulo operation, then the vectors of successive values will all belong to L'_t, where $L'_t = L_t + V_{0,t}$ is a shift of L_t by some constant $V_{0,t} \in \mathbf{Z}_m^t$. Such a "shifted lattice" is called a *grid*. Since L'_t and L_t have the same structural properties, `LatMRG` does not consider the possible presence of a constant b in (1). In other words, we consider only the MLCGs and MRGs.

When m is prime and the MRG has full period $m^k - 1$, then T_t is the set of all t-tuples produced by the generator over its main cycle, plus the zero vector. Otherwise, if the generator does not have full period, then the set of t-dimensional vectors produced over any given (sub)cycle (plus the zero vector and plus $m\mathbf{Z}^t$) is a strict subset of L_t which may not form a lattice. In that case, `LatMRG` can still analyze the set of all t-tuples produced over the *union* of all subcycles, but can also analyze some strict sublattices (if desired), as described in Sections 5 and 6.

A lattice basis for which the vectors are "reduced" in the following sense is called a *Minkowski-reduced lattice basis* (MRLB) [1, 2, 11]: (i) the first vector V_1 of the basis is a shortest vector in the lattice; (ii) for each $i < t$, given the first i vectors $V_1, \ldots, V_i$, the $(i+1)$th vector V_{i+1} of the basis is a shortest vector among those vectors V such that the set $\{V_1, \ldots, V_i, V\}$ can be extended into a lattice basis of the t-dimensional lattice. Geometrically, a MRLB is a basis for which the vectors are in some sense the most orthogonal (see [3, 11]). The Beyer quotient of a Minkowski reduced basis is defined as the ratio of the lengths of the shortest to that of the longest vector in this basis. In dimensions $t > 6$, there may exist two MRLB with unequal Beyer quotients [30]. We denote by q_t the upper bound of the Beyer quotients of all MRLB of L_t. Values of q_t close to one are preferable. A figure of merit can be $Q_T = \min_{k<t\leq T} q_t$ for some large enough T. Our package permits one to compute q_t. We note that defining q_t via a MRLB is somewhat arbitrary. There are other definitions of a reduced basis that one could use instead, e.g., a Hermite-reduced basis [13].

The lattice structure implies that all points of T_t lie in a relatively small 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. This distance is in fact equal to one over the length of the shortest vector in the *dual* lattice to L_t. Our package computes that shortest vector through a branch-and-bound algorithm, as described in [5, 8]. For large dimensions t, this algorithm is much faster than the one given by Dieter [7] and Knuth [15]. If we rescale the hypercube $[0, m)^t$ that contains T_t to $[0, 1)^t$, the distance between hyperplanes becomes $d_t = D_t/m$. There is a theoretical lower bound d_t^* on d_t [15, 10] and one can define the figures of merit $S_t = d_t^*/d_t$ and $M_T = \min_{k\leq t\leq T} S_t$, which lie between 0 and 1. Again, one seeks values near one. Note that d_t^* is known only for $t \leq 8$.

The measure d_t is easier to interpret and justify than q_t. It is also much faster to compute. Our package deals with q_t mainly for historical reasons. One advantage of the Beyer quotients is that they are all normalized (between 0 and 1) and that Q_T is defined for all positive T, in contrast to M_T. One may then compare (and rank) generators of the same size using the figure of merit Q_T for a large T. To do the same with M_T, for $T > 8$, one could compute (e.g., by simulation) approximations of the d_t^* (or find tight bounds) and use these approximations to define a figure of merit approximately equivalent to M_T. This would require more work and we do not pursue it in the present paper.

2. Going up one dimension

Suppose one wants to apply the spectral or lattice test in successive dimensions $t = k+1, k+2, \ldots, T$. When going up from dimension $t-1$ to dimension t, a straightforward approach uses equations (5–6) to rebuild from scratch bases for the lattice and its dual in dimension t. Alternatively, one can try to "extend" the current basis for dimension $t-1$ into a basis for dimension t. This is done by adding a zero coordinate to each vector W_i in the dual basis, yielding (if one also adds an extra vector W_t described below) a basis of the dual lattice for dimension t. The rationale is that since the basis in dimension $t-1$ has already been reduced, the vectors in the dual basis should be small and, since extending them does not increase their length, this must provide a much better starting point for the next round than rebuilding the basis from scratch. This turns out to be an efficient heuristic in practice.

The basis to be used with the extended dual basis above is obtained as follows. We add one (new) coordinate to each vector V_i for $i < t$. The value of that new coordinate, say $x_{i,t}$, is easily determined by (4), since the previous coordinates are known. We also add the extra vector $V_t = m\, e_{t(t)}$. To complete the update of the m-dual basis, let

$$W_t = e_{t(t)} - \frac{1}{m}\sum_{i=1}^{t-1} x_{i,t} W_i. \qquad (7)$$

It is easily verified that $V_i \cdot W_j = m\delta_{ij}$ for all i, j. Note that in general, the $x_{i,t}$'s and W_t would not be the same as when the basis is constructed from scratch, because the first $t-1$ coordinates of the (original) vectors $V_1, \ldots, V_{t-1}$ have been transformed linearly and we must apply the same transformations to their last coordinates.

3. Lacunary indices

Instead of forming vectors with successive values as in the definition of T_t given in (3), one can form vectors with values that are some distance apart in the sequence (so-called "leapfrog" values, or *lacunary indices*). Let $I = \{i_1, i_2, \cdots, i_t\}$ be a set of fixed integers (not necessarily ordered or distinct). Redefine

$$T_t = \{(x_{i_1+n}, \ldots, x_{i_t+n}) \mid n \geq 0,\ s_0 = (x_0, \ldots, x_{k-1}) \in \mathbf{Z}_m^k\}$$

and consider the lattice $L_t = T_t + m\mathbf{Z}^t$. The previous definition of L_t was a special case of this one, with $(i_1, \ldots, i_t) = (0, \ldots, t-1)$.

To construct a basis for this L_t, we need to be able to compute x_{i_j} from any s_0. To see how this can be done efficiently even when i_j is large, we will use the polynomial representation of $\mathbf{Z}_m^k$. We consider the set of polynomials $P(x) = \sum_{j=0}^{k-1} b_{0,j} x^j$ with coefficients in $\mathbf{Z}_m$. To the polynomial $P(x)$ corresponds the sequence of polynomials $P_n(x) = ((x^n P(x)) \bmod f(x)) \bmod m = \sum_{j=0}^{k-1} b_{n,j} x^j$ for all $n \geq 0$ and the state $s_0 = (b_{0,k-1}, \ldots, b_{k-1,k-1})$. The sequence $\{b_{n,k-1},\ n \geq 0\}$ follows the recurrence (1) for all $n \geq k$, starting from the seed s_0. This gives an efficient way for computing x_n for large n from s_0 or from the corresponding polynomial $P(x)$: compute $P_n(x)$ by a standard divide-to-conquer algorithm (see [15]); then $x_n = b_{n,k-1}$.

Let $U_i \in \mathbf{Z}_m^t$ denote the vector $(x_{i_1}, \ldots, x_{i_t})$ obtained when $P(x) = x^{i-1}$, $i = 1 \ldots, k$. It is shown in [6] how to construct a system $\{\bar{V}_1, \ldots, \bar{V}_t\}$ generating the same subgroup of $\mathbf{Z}_m^t$ as $(U_1, \ldots, U_k)$ and with the following properties (i–iv), where $\mathrm{ind}(V)$ denotes the index of the first non-zero coordinate of $V \in \mathbf{Z}_m^t$:

(i) $\mathrm{ind}(\bar{V}_i) \geq i$;

(ii) If $\mathrm{ind}(\bar{V}_i) = i$, then the i-th coordinate of $\bar{V}_i$ divides m;

(iii) $\mathrm{ind}(\bar{V}_i) > i$ implies $\bar{V}_i = 0$;

(iv) If $V \in S'$ and $\mathrm{ind}(V) \geq j$, then V is in the subgroup of $\mathbf{Z}^t$ generated by$\{\bar{V}_j, \ldots, \bar{V}_t\}$.

This system is a lattice basis for L_t after one has replaced zero vectors $\bar{V}_i$ by $me_{i(t)}$.

Let L_t and L_{t-1} be the lattices corresponding to the index sets $I = \{i_1, \ldots, i_t\}$ and $I' = \{i_1, \ldots, i_{t-1}\}$. As in the case of consecutive indices, one can extend a current (reduced) basis $V_1, \ldots, V_{t-1}$for L_{t-1} to a basis of L_t in such a way that the dual vectors W_i, $i < t$ in dimension t are obtained by adding a zero coordinate to the dual vectors of the previous dimension.

This can be done using the basis for L_t given by $\bar{V}_1, \ldots, \bar{V}_t$. Observe that L_{t-1} admits the basis formed by the vectors obtained from $\bar{V}_i$, $i < t$ by omitting the last coordinate. One can therefore add a coordinate to each vector V_i, $i < t$ so that

$$\begin{pmatrix} V_1 \\ \vdots \\ V_{t-1} \end{pmatrix} = M \begin{pmatrix} \bar{V}_1 \\ \vdots \\ \bar{V}_{t-1} \end{pmatrix}, \tag{8}$$

where M is the (unique) unimodular matrix of order $t-1$ which makes the two sides agree in the first $t-1$ columns. One obtains a basis for L_t after adding the vector $V_t = \bar{V}_t$.

Clearly, the columns of M are easily determined from left to right using (8) since the matrix that multiplies M on the right side of (8) is upper triangular. Once a basis is available, the distances between hyperplanes, Beyer quotients, and so on, can then be computed as usual.

4. Generators with transient states and lattices generated by the recurrent states

If, in the recurrence (1), a_k is not invertible modulo m, then the mapping $s_n \mapsto s_{n+1}$ defined by (1) is not invertible. As a result, certain (transient) states cannot be visited more than once by the generator. One can then consider the subset S_r of *recurrent* (i.e., non-transient) states, which form a subgroup of $S = \mathbf{Z}_m^k$. If one uses the polynomial representation of the state space, this subgroup is generated by the set $(x^{\eta+i-1} \bmod f(x)) \bmod m$, $i = 1, \ldots, k$, if $2^\eta > m^k$ (see [6]).

Now, for a given set of (possibly lacunary) indices $I = \{i_1, i_2, \cdots, i_t\}$, define

$$T_{r,t} = \{(x_{i_1+n}, \ldots, x_{i_t+n}) \mid n \geq 0,\ s_0 = (x_0, \ldots, x_{k-1}) \in S_r\}$$

and let $L_{r,t}$ be the lattice generated by $T_{r,t}$. When $S_r \neq S$, $L_{r,t}$ is in general a strict sublattice of L_t. Bases for $L_{r,t}$ and its dual can be constructed in the same way as those constructed in Section 4, but with x^{i-1} replaced by $(x^{\eta+i-1} \bmod f(x)) \bmod m$.

In practice, situations where m is composite and a_k not invertible modulo m arise when we combine two or more MRGs [6, 20]. For example, suppose that we combine two MRGs, the first one having modulus m_1, order 2 and multipliers a_{11} and a_{12}, and the second one having modulus m_2, order 1, and multiplier a_{21}. Suppose that m_1 and m_2 are relatively prime. Then, the combined generator is "equivalent" (see [20]) to an MRG with modulus $m = m_1 m_2$, order 2, and multipliers a_1, a_2, such that $a_2 = a_{12}(m_2^{-1} \bmod m_1) m_2 \bmod m$ is a multiple of m_2 and so is not invertible modulo m. See also Section 8.4 for a numerical example.

5. Sublattices generated by subcycles in a MLCG when m is a power of a prime

Linear congruential generators whose modulus is a power of two have been quite popular in the past and are still widely used, mainly because the modulo operation when m is a power of two is easily implemented and very fast on binary computers (for example, take $m = 2^{32}$ on a 32-bit computer). Analyzing the lattice structure of such generators requires special care, as we now explain. Consider a MLCG ($k = 1$) with modulus m and multiplier $a = a_1$, where m is a power of a prime p, say $m = p^e$ for $e > 1$ ($p = 2$ is a special case). Generally, such a generator has several distinct subcycles and the vectors of successive values over any given subcycle form a proper subset of T_t. We now explain how to take care of this situation by analyzing the appropriate associated lattice. We restrict our attention to the (most interesting) case where the seed x_0 is prime to p.

Let $p = 2$. If $a \equiv 1 \pmod 4$ and $\nu = \max\{n > 1 \mid a \equiv 1 \pmod{2^n}\}$, then the period length is $m/2^\nu$ and the set of points visited over that subcycle is the intersection of a translate of a lattice $\tilde{L}_t$ with $[0, m)^t$. It also turns out that bases for this lattice and its dual can be constructed as usual, after simply replacing m by $m/2^\nu$ [9, 15]. For instance, if $a \equiv 5 \pmod 8$, then $\nu = 2$. In the case where $a \equiv 3 \pmod 4$, the visited points form the intersection of *two* translates of a same lattice $\tilde{L}_t$ with the unit cube. The same as above concerning the period length and basis construction applies to $\tilde{L}_t$, provided that we now define $\nu = \max\{n > 1 \mid a^2 \equiv 1 \pmod{2^n}\}$.

For $p > 2$ we have similar phenomena. We distinguish the two cases $a \equiv 1 \pmod p$ and $a \not\equiv 1 \pmod p$. In the first case we take $\nu = \max\{n > 1 \mid a \equiv 1 \pmod{p^n}\}$ while in the second case, $\nu = \max\{n > 1 \mid a^{p-1} \equiv 1 \pmod{p^n}\}$. Lattice bases can be constructed similarly as for the case where $p = 2$, except that one now divides m by p^ν and that in the second case, the points produced over one subcycle form the intersection of $p-1$ translates of the same lattice. Note that if p is large, it is not clear whether analyzing the structure of only one of those translates is appropriate. `LatMRG` also allows one to analyze the lattice L_t associated with the union of all subcycles.

6. Overview of the software package

The package `LatMRG` is large software system implemented as a library of modules written in the Modula-2 language. It provides different tools for examining the theoretical properties of generators based on linear recurrences in modular arithmetic. It offers facilities for checking if a generator has maximal period or not, to apply the lattice and spectral tests, and to perform computer searches for "good" generators according to different criteria. We now give a brief overview of the package structure and functionality.

We may classify the modules of `LatMRG` in three groups: (a) low-level, (b) intermediate-level, and (c) high-level. Higher-level modules import facilities from lower-level ones. The high-level modules (c) are in fact programs in executable form which read their data in files or work interactively with the user. They can either analyze a given generator or seek "good" generators according to different criteria. Examples of what they can do are given in the next section. The intermediate-level modules (b) provide data types and procedures to construct lattice bases for different classes of generators (simple or combined MRGs, lacunary indices, etc.), manipulate such bases, find a shortest vector in a lattice, reduce a basis in the sense of Minkowski, and so on. These tools are used by the upper-level modules (c), but can also be used directly to make programs different than those already provided at level (c), offering thus more flexibility. The lower-level modules (a) implement basic operations on scalars, vectors, matrices, polynomials, and so on. They allow different possible representations for these objects, according, for example, to the size of the modulus m and to the precision we want. To deal with large integers, they use other modules from the package SENTIERS [24]. Several procedures make conversions between the different representations. These lower-level tools are used by the modules of levels (b) and (c).

We now discuss a little more each of the three levels. We do not explain here each module and procedure (there are more than 30 modules and several hundred procedures). For more details, see the user's guide [23].

6.1 Level a: large numbers and basic tools

`LatMRG` can deal with large moduli and multipliers. Theoretically, there is no limit on the size other than the size of the computer's memory and the cpu time. Generators with moduli of several hundred bits can be analyzed. Operations on large integers are performed using the package SENTIERS [24], also written in Modula-2. Of course, these operations are performed in software and are significantly slower than the standard operations supported by hardware. For that reason, most of the basic (low-level) operations required by our programs have been implemented in two versions.

For example, when building a basis or checking maximal period conditions, the modulus and multipliers can be represented either as `LONGINT` (regular 32-bit integers) or

SuperInteger (arbitrary large integers, from the package SENTIERS). When working on a lattice basis (finding shortest vector, Minkowski reduction, etc.), the vector coordinates can be represented either as LONGREAL (64-bit floating-point numbers) or SuperInteger. The low-level modules support these different representations, and provide basic facilities for the other modules. To change the representation in a higher-level module or program, the latter must be recompiled and relinked with the appropriate lower-level modules. Software tools at the "command" level have been implemented to facilitate these operations.

6.2 Level b: basis construction and reduction

The intermediate-level modules construct and manipulate bases using the techniques we saw in the previous sections. They can also perform different types of basis reduction and compute the shortest vector in the lattice or its dual.

To compute a shortest non-zero vector, we use a version of a branch-and-bound algorithm described in [5, 8]. That algorithm constructs the shortest vector coordinate by coordinate, relatively to the current lattice basis, and needs a Choleski decomposition of the matrix of scalar products of all pairs of vectors in this basis. When this is implemented in 64-bit floating-point arithmetic, numerical roundoff errors may occur and affect the results. In fact, in high dimensions (say, 25 or more), we have observed in some examples significant numerical errors in the standard Choleski decomposition, especially when the basis includes both short and long vectors. For that reason, we have implemented versions of the algorithms which take into account all sources of numerical error during the computations, and compute error bounds on them, thereby yielding "guaranteed error-free" results (see [5] for further details). Of course, computing such bounds entails overhead. For this reason we also have versions which do not take roundoff error into account (i.e., do not compute error bounds). When performing a search for good generators, for instance, one can first perform all the "screening" computations (involving many generators) without computing the error bounds, and then recompute (verify) with the error bounds only for the retained generator(s). We also implemented another approach for "guaranteed error-free" results, which bypasses all floating-point calculations (all quantities are large integers which can be represented exactly), and which is sometimes slower but sometimes much faster than the "error-bound" approach (see [5] for more details).

To construct a MRLB, we use an algorithm similar to that given in [1], as explained in [5]. This is an iterative algorithm which at each step computes a shortest nonzero vector among those vectors V such that the set of already selected vectors, plus V, can be extended into a lattice basis. That shortest vector is computed via a branch-and-bound algorithm similar to the one mentioned above, but with additional constraints. In case of a tie between two or more distinct V, each such V can lead to one (or more) distinct MRLB. The algorithm then finds all MRLBs and selects the one whose longest vector is shortest.

6.3 Level c: programs in executable form

At the high-level end, LatMRG provides programs in executable form to (i) verify the maximal period conditions for a MRG, (ii) analyze the lattice structure of a simple or combined MRG, and (iii) perform a search for good generators. The programs for (ii) and (iii) come in different versions, with different computer representations for the multipliers and bases. Some allow larger numbers while others are faster.

In the data file, the user specifies the parameters of each MRG (each component, in the case of a combined generator), which information to compute (q_t, d_t, or both), in which dimensions, which lattice to analyze (the one generated by all states, or only one subcycle, or only the recurrent states), whether lacunary indices are used and which ones, in which form the results should be given (only on the terminal screen, or in a text file, or in a specially formatted file which can be read back by the program for further analysis in a later run), and a few other parameters of the basis reduction of search algorithms. The list of generators to analyze (or to search from) can also be taken from a separate file previously produced by one of the programs in (ii) or (iii).

The search programs (iii) can perform a search for the N_g "best" generators of a given form, based on either Q_T or M_T, for given values of T and N_g. One must specify, for each MRG component, how the search is performed (exhaustive or random, see below), in which area of the space of multipliers, whether maximal period is required or not, and if we want some further conditions on the multipliers (for ease of implementation). Besides what we mentioned in the previous paragraph, the data file should also say how many generators to retain (N_g), what is the search criterion (Q_T or M_T), what is the minimal acceptable figure of merit, and some further parameters such as a cpu time limit. The search programs produce a report listing the retained generators, their properties, and various statistics on the search.

We now look more closely at how the searches are performed. For a given modulus m and order k, the search for good vectors of multipliers is made inside a region bounded by specified vectors $b = (b_1, \ldots, b_k)$ and $c = (c_1, \ldots, c_k)$ such that $-m < b_i \le c_i < m$ for each i. The search can be exhaustive in that region, or random. One can search only among maximal period generators, or not consider the period and examine only the lattice structure. Since the list of retained generators can be stored in a file (in a special format) and read back by the program in a later run, one can easily perform multipass searches. For example, one may first perform a screening over a large region, based on a criterion that does not require expensive computations, then do a second pass over the retained generators, based on a more stringent criterion, such as looking at the lattice structure in higher dimensions, and finally verifying the results by performing all computations using error bounds (see the previous subsection).

For an exhaustive search, all vectors of multipliers of the form $a = (a_1, \ldots, a_k)$ such that $b_i \le a_i \le c_i$ for $i = 1, \ldots, k$ are examined, for a total of $\prod_{i=1}^{k}(c_i - b_i + 1)$ vectors (generators). For a random search, we fix a number of subregions (clusters) that we want to examine, and the size h_i of each subregion in dimension i, for $i = 1, \ldots, k$. The program will examine a total of $n \prod_{i=1}^{k} h_i$ generators by repeating n times the following: For $i = 1, \ldots, k$, generate α_i randomly, uniformly over the set $\{b_i, \ldots, c_i - h_i + 1\}$; then, examine all the vectors $a = (a_1, \ldots, a_k)$ such that $\alpha_i \le a_i \le \alpha_i + h_i - 1$ for each i.

When examining a vector a of multipliers, the program first checks if the maximal period conditions are satisfied, if this is required. For prime modulus m, one of these conditions asks for $(-1)^{k-1}a_k$ to be primitive modulo m (see condition (i) in Knuth [15, p.29]). That condition is verified only once for each distinct value of a_k (which corresponds to $\prod_{i=1}^{k-1} h_i$

different generators). To verify the maximal period conditions, the factorizations of $m-1$ and $r = (m^k - 1)/(m-1)$ are required. They can be found by the program, if desired, or provided by the user in a file (factorizing r often takes *huge* amounts of time).

If a is not rejected by the maximal period test, the values of d_t and/or q_t are then computed for dimensions $k+1, \ldots, T$. The program always keeps a lower bound σ on the figure of merit (M_T or Q_T) for the generator to be worth considering. The initial value of the lower bound is given by the user (it can be 0.0, which means no initial lower bound). During execution, whenever the figure of merit x of the N_gth best generator becomes larger than the lower bound σ, then σ is increased to x. If, in a dimension $t < T$, the program already sees that the considered generator will have a figure of merit smaller than σ, the generator is rejected right away and no calculations in higher dimensions are performed for this generator. If M_T is used as a criterion for $T > 8$, the distances between hyperplanes are computed for dimensions up to T, but the selection of generators is based only on M_8, because the theoretical lower bound d_t^* is known only for $t \leq 8$.

7. Examples

We now give some numerical examples. Our aim here is *not* to compare different generators or to recommend any particular generator, but rather to illustrate what the programs can do and give a rough idea of how much cpu time they take. The timings are from runs on a SUN SparcStation 20, under SunOS 5.4, using version 4.5.1 of the MCS Modula-2 compiler from ModulaWare [14].

7.1 Example 1: A MLCG with modulus 2^{32}

Table 1 gives the results of the spectral test for the MLCG with modulus $m = 2^{32}$ and multiplier $a = 1099087573$ in dimensions up to 30. This multiplier was among the best found by Fishman [9] for the modulus 2^{32}, based on the lattice structure in dimensions 2 to 6. The lattice that is analyzed here is the sublattice $\tilde{L}_t$ discussed in Section 6. The successive columns give the dimension t, the distance d_t between hyperplanes, the figure of merit S_t, and the cumulative cpu time to compute and print the values in Table 1. For instance, the total cpu time to compute d_t in dimensions 2 to 20 was approximately 0.25 seconds, and that for dimensions 2 to 35 was approximately 6.5 seconds. Computing q_t is much more expensive than computing d_t; to give an idea, to compute q_t on the same machine, it took 2.2 seconds for dimensions 2 to 20 and 253 seconds for dimensions 2 to 30 (not shown in the table). These cpu times are representative of what happens in general for examples of this type. The smallest q_t for $t \leq 30$ is $q_7 = 0.5486$.

7.2 Example 2: A MLCG with lacunary indices

We consider the MLCG with $m = 2^{31} - 1$ and multiplier $a = 16807$, with the set I of lacunary indices formed by 10 triplets of three successive indices, taken $d = 131,072 = 2^{17}$ values apart; that is, $t = 30$ and $I = \{0, 1, 2, d, d+1, d+2, 2d, 2d+1, 2d+2, \ldots, 10d, 10d+1, 10d+2\}$.

This (well-known) generator was suggested in [4], where seeds spaced 2^{17} values apart were also given (in Section 6.8.3) to generate disjoint subsequences. So, the spectral test with the set I of lacunary indices analyzes to some extent the correlation between the corresponding values of the disjoint subsequences. The distances between hyperplanes and Beyer quotients are given in Table 2. For dimensions $t < 30$, the results correspond to the set I' formed by the first t values of I. The computations took a little over 5 minutes. For this example, computing only the d_t and S_t for the same dimensions and on the same machine takes approximately 4 seconds. On the other hand computing all the q_t using bounds on the numerical errors for formal verification takes approximately 12.5 minutes.

7.3 Example 3: A combined MLCG with lacunary indices

In this example, we analyze the combined generator of L'Ecuyer [16] with lacunary indices. That generator was shown to be "approximately" equivalent to a MLCG with modulus $m = 4611685301167870637$ and multiplier $a = 1968402271571654650$. We consider a set of lacunary indices similar to that of Example 2, formed by 10 triplets of three successive indices, which are now taken $d = 2^{30}$ values apart. Table 3 shows the results. Note that the distances between hyperplanes here are significantly smaller than those of Tables 1 and 2.

7.4 Example 4: A MRG of order 2 combined with a MLCG

Consider a combined generator whose first component is a MRG of order $k_1 = 2$, modulus $m_1 = 32749$, and multipliers $a_{11} = 180$ and $a_{12} = -175$, while the second component is a MLCG with modulus $m_2 = 32363$ and multiplier $a_{22} = 157$. Suppose that at step n, we divide the "state" x_{1n} of the first recurrence by m_1, that of the second recurrence by m_2, and add them up modulo 1 to produce the output. Using Proposition 1 in [20], one can verify that this yields a generator which is equivalent to a MRG of order 2 with modulus $m = m_1 m_2 = 1059855887$ and multipliers $a_1 = 919821343$ and $a_2 = 650755204$. Here, m is not prime and that generator has transient states, as explained in Section 5. If we analyze the lattice $L_{p,t}$ determined by the set S_r of recurrent states, we obtain the results of Table 4, while if we consider the lattice L_t determined by all possible states (both recurrent and transient), we obtain the results of Table 5. Here, $L_{r,t}$ is a strict sublattice of L_t and that translates into larger distances between hyperplanes. In practice, when using a combined generator, we will typically start from a recurrent state and never visit any transient state. Therefore, it is more appropriate to analyze $L_{r,t}$ than L_t.

7.5 Example 5: Exhaustive searches for MLCGs with specific properties

Consider the set of MLCGs with prime modulus $m_1 = 2^{31} - 1$, period length $m_1 - 1$, and whose multiplier a_1 satifies the condition $m_1 \bmod a_1 < m_1/a_1$ (for ease of implementation). Within this set, we find the generator with the largest value of $M_8 = \min_{t \le 8} S_t$. These requirements are given as data to a (level c) search program. The program then performs an exhaustive search for the best generator(s) and prints the results of Table 6. Actually, the program prints more information than shown, but we removed some for space reduction.

The best multiplier found is $a_1 = 45991$, with $M_8 = 0.6984$. The exhaustive search took a little more than 4 minutes. The program examined 52679 values of a_1, found that 13182 of them were primitive modulo m_1 (giving maximal period), and retained the 10 best among these 13182, based on M_8. The "`NoVerify`" option in the data means that the computations related to the Choleski decomposition are performed in 64-bit floating point without taking the roundoff errors into account.

Consider now the class of MLCGs with prime modulus $m_2 = 2^{31} - 105$, period length $m_2 - 1$, and with multiplier a_2 satisfying $m_2 \bmod a_2 < m_2/a_2$. Combining a MLCG of this class with the first one we found above (with $(m_1, a_1) = (2^{31} - 1, 45991)$), using one of the combination methods of [25], we obtain a combined generator equivalent to a MLCG with modulus $m_1 m_2$ and period length $m_1 m_2/2$. We seek the mutiplier a_2 such that the combined generator has maximal value of M_8. Again, with the appropriate data file, the program makes an exhaustive search and produces a results file similar to that summarized in Table 6. The best multiplier found is $a_2 = 207707$ and the corresponding combined generator has $M_8 = 0.7001$. One could go on and find a third component such that the combined generator with three components has the largest M_8, and so on.

7.6 Example 6: A search for an MRG with large modulus

We now give an example of a search for an MRG of order $k = 3$, with modulus $m = 2^{63} - 2247$ (the largest prime m below 2^{63} such that $(m^3 - 1)/(m - 1)$ is also prime), for which $a_2 = 0$, a_i (m mod a_i) $< m$ for $i = 1$ and 3, and with maximal period $\rho = (2^{63} - 2247)^3 - 1$. We examine 1000 subregions of dimension $(4 \times 1 \times 4)$, i.e., a total of 16000 generators. The results appear (partially) in Table 7. In (roughly) one hour of random search, the program found 1720 full period generators of the form specified, but all with a very low figure of merit. The best one has $S_4 \approx 4.9 \times 10^{-7}$. In fact, all MRGs of the specified form have a very low S_4, due to the constraint $a_2 = 0$ (see [19] for more explanations about this). For this reason, it is recommended to avoid zero multipliers or (perhaps better) to combine MRGs as done in [20] and in the next example. On the other hand, the distance d_t is much smaller here, for all t, than the one in each of the examples 1–5. So, despite a small S_4, this MRG of order 3 dominates the previous ones. The intuitive reason for the smaller S_4 (in comparison with the previous examples) is that since its period length is much longer, it has more points at its disposal, so the lower bound d_t^* on d_t is much smaller.

7.7 Example 7: A search for a combined MRG with large modulus

Let us now search for a "good" combined MRG with two components of order $k = 3$, for 64-bit computers. The two moduli chosen are $m_1 = 2^{63} - 2247$ and $m_2 = 2^{63} - 9609$, respectively (these are the two largest primes m_j below 2^{63} such that $(m_j^3 - 1)/(m_j - 1)$ is also prime). We impose the following conditions on the coefficients a_i of the recurrence. For both components, we require $|a_i|$ (m mod $|a_i|$) $< m$ for all i. We also require $2^{25} < a_1 < 2^{35}$, $a_2 = 0$, and $-2^{35} < a_3 < -2^{25}$ for the first component, and $a_1 = 0$, $2^{25} < a_2 < 2^{32}$, and $-2^{32} < a_3 < -2^{25}$ for the second component. Finally, the coefficients must be such that the combined generator has maximal period $\rho = ((2^{63} - 2247)^3 - 1)((2^{63} - 2247)^3 - 1)/2 \approx 2^{377}$. Among these generators, we look for one with a large figure of merit M_8.

The search for good coefficients a_i was random and examined 100 subregions of dimension $(8 \times 1 \times 4)$ for the first component and 100 subregions of dimension $(1 \times 8 \times 4)$ for the second one. We found 394 primitive polynomials for the first component and 564 for the second one, which gives 222216 full period combined generators satisfying the conditions. After nearly 3 hours of computations, the program came up with a list of best generators. The two best are given in Tables 8 and 9. Their figures of merit are $M_8 = S_4 = 0.73595$ for the first generator and $M_8 = S_8 = 0.73436$ for the second one.

Table 1: Spectral test for the generator of Example 1.

t	d_t	S_t	cumulative cpu (sec)
2	3.184E-5	0.89204	0.01
3	1.016E-3	0.85634	0.02
4	5.399E-3	0.86035	0.03
5	0.01507	0.84205	0.04
6	0.02909	0.83254	0.05
7	0.06868	0.55466	0.06
8	0.07001	0.75065	0.07
9	0.13868		0.07
10	0.15430		0.08
11	0.17150		0.09
12	0.17678		0.10
13	0.17678		0.11
14	0.19612		0.13
15	0.23570		0.14
16	0.23570		0.16
17	0.26726		0.18
18	0.28868		0.20
19	0.28868		0.23
20	0.28868		0.25
21	0.28868		0.28
22	0.28868		0.36
23	0.28868		0.46
24	0.31623		0.55
25	0.31623		0.64
26	0.31623		0.78
27	0.31623		0.96
28	0.31623		1.24
29	0.31623		1.69
30	0.31623		2.39
31	0.35355		2.78
32	0.35355		3.19
33	0.35355		3.82
34	0.35355		4.78
35	0.35355		6.54

Table 2: Spectral and lattice tests results for Example 2.

t	d_t	q_t	S_t	cumulative cpu (sec)
2	5.950E-5	0.13151	0.33751	0.01
3	1.565E-3	0.29533	0.44118	0.02
4	4.810E-3	0.84208	0.81211	0.04
5	0.02503	0.31520	0.44139	0.05
6	0.04415	0.26198	0.48863	0.06
7	0.04603	0.76509	0.74959	0.08
8	0.07538	0.82702	0.63937	0.10
9	0.14142	0.56305		0.13
10	0.14142	0.55920		0.15
11	0.14586	0.65384		0.20
12	0.15076	0.82755		0.25
13	0.16903	0.79080		0.34
14	0.20412	0.68542		0.44
15	0.20851	0.78554		0.59
16	0.23570	0.75328		0.78
17	0.25820	0.72583		1.04
18	0.25820	0.90410		1.69
19	0.25820	0.87023		2.29
20	0.26726	0.85431		3.19
21	0.27735	0.84129		4.06
22	0.27735	0.82662		6.97
23	0.28868	0.88720		10.18
24	0.30151	0.86030		16.58
25	0.30151	0.82550		31.84
26	0.30151	0.82763		57.92
27	0.30151	0.88913		100.80
28	0.31623	0.86869		146.88
29	0.31623	0.88768		229.41
30	0.35355	0.86502		301.47

Table 3: The combined generator of L'Ecuyer, with lacunary indices.

t	d_t	S_t	cumulative cpu (sec)
2	6.502E-10	0.66650	0.01
3	7.002E-7	0.76439	0.02
4	4.552E-5	0.39867	0.04
5	3.025E-4	0.49685	0.06
6	8.949E-4	0.67113	0.08
7	2.902E-3	0.55212	0.12
8	4.560E-3	0.72029	0.16
9	8.261E-3		0.24
10	0.01416		0.31
11	0.02197		0.41
12	0.02558		0.57
13	0.03360		0.76
14	0.04096		0.98
15	0.05376		1.28
16	0.05670		1.60
17	0.06565		1.98
18	0.07906		2.43
19	0.09535		2.96
20	0.09535		3.55
21	0.10000		4.55
22	0.11111		5.72
23	0.13245		7.02
24	0.13245		8.28
25	0.13245		9.75
26	0.13868		12.03
27	0.14142		13.79
28	0.14744		16.02
29	0.16903		18.07
30	0.16903		20.51

Table 4: The combined generator of Example 4: sublattice $L_{r,t}$

t	d_t	S_t	cumulative cpu (sec)
3	0.00101	0.86558	0.01
4	0.00595	0.78239	0.01
5	0.02153	0.59097	0.02
6	0.04120	0.58897	0.03
7	0.08362	0.45639	0.04
8	0.08362	0.62950	0.05
9	0.10153		0.06
10	0.15250		0.07
11	0.17150		0.08
12	0.17150		0.09
13	0.19612		0.10
14	0.22361		0.11
15	0.23570		0.13
16	0.23570		0.14
17	0.25820		0.17
18	0.25820		0.19
19	0.25820		0.22
20	0.27735		0.25

Table 5: The combined generator of Example 4: full lattice

t	d_t	S_t	cumulative cpu (sec)
3	2.582E-6	0.33197	0.01
4	5.886E-5	0.43884	0.02
5	6.907E-4	0.28859	0.03
6	2.140E-3	0.35512	0.04
7	5.519E-3	0.35523	0.05
8	0.01123	0.34883	0.06
9	0.02174		0.07
10	0.03446		0.08
11	0.04608		0.09
12	0.06275		0.10
13	0.07019		0.11
14	0.10483		0.13
15	0.10483		0.15
16	0.10483		0.16
17	0.12039		0.19
18	0.15076		0.21
19	0.15076		0.24
20	0.15076		0.27

```
 DATA:
   Modulus  m = 2147483647 ( = 2^31 -1 )
   Bounds : a1 from : 40000
                 to : 1000000000
   Search method                         : EXHAUST
   Implem. cond.  a_i (m mod a_i) < m : YES
   Maximum period required               : YES
   Merit criterion                       : M_8
   Verify Branch-and-bound               : NoVerify
   Maximum nodes in branch-and-bound     : 1000000
   Lattice Type                          : Full

 RESULTS:
   Values of a1 tried                       :      52679
   Values of a1 primitive element           :      13182
   Nb. Generators conserved                 :         10
   Total CPU time (after setup)             : 0:04:15.55
 -----------------------------------------------------
1.
   a_1 =            45991

    t        d_t              q_t           S_t
 -----------------------------------------------------------
    2    2.17434E-005       0.90464       0.92358
    3    8.43240E-004       0.85044       0.81891
    4    4.94656E-003       0.81124       0.78969
    5         0.01536       0.53212       0.71917
    6         0.03015       0.64010       0.71552
    7         0.04531       0.83008       0.76141
    8         0.06901       0.79459       0.69840
    9         0.12403       0.60142
   10         0.14744       0.53565

   Merit =    0.69840  = S_8

2.
   a_1 =            61407

   ⋮             ⋮
```

Table 6: Seach for a good MLCG with modulus $2^{31} - 1$.

```
 DATA:
   Modulus  m = 9223372036854773561 ( = 2^63 - 2247 )
   Order    k = 3
   Bounds : a1 from : 1125899906842624
                  to : 9223372036854765808
            a2 from : 0
                  to : 0
            a3 from : -9223372036854765808
                  to : -1125899906842624
   Search method                      : RANDOM
      Sampling : Numbers of subregions = 1000,  H = 4,  Hk = 4
   Implem. cond.  a_i (m mod a_i) < m : YES
   Maximum period required            : YES
   Merit criterion                    : M_8

 RESULTS:
   Values of a3 tried                    :       4000
   Values of a3 primitive element        :       1272
   Polynomials with a3 primitive element :       5088
   Primitive polynomials                 :       1720
   Nb. Generators conserved              :          3
   Total CPU time (after setup)          : 0:58:33.34
 -----------------------------------------------------
1.
   a_1 = 1145902849652723
   a_2 = 0
   a_3 = -1184153554609676

    t        d_t          q_t         S_t
 ---------------------------------------------
    4    1.02228E-8    1.061E-11    4.915E-7
    5    1.02228E-8    1.068E-7     3.320E-4
    6    1.02228E-8    2.891E-4     0.02496
    7    1.02228E-8    0.27884      0.54151
    8    1.05850E-7    0.63196      0.51637
    9    4.69926E-7    0.71015
   10    2.01652E-6    0.63542
   11    6.51884E-6    0.62227
   12    1.78722E-5    0.84289

   Merit = 4.91481E-7  = S_4

2.
   a_1 = 1161195019118062
   a_2 = 0
   a_3 = -1243042053484470

   ⋮           ⋮
```

Table 7: Results of the search for an MRG of order 3 with $m = 2^{63} - 2247$.

Table 8: Best combined MRG of order 3 found for Example 7.

Component	a_1	a_2	a_3
1	3866005879	0	-3472501966
2	0	48193584	-3751984989

t	d_t	S_t
4	4.07906E-29	0.73595
5	1.63643E-23	0.86682
6	1.11424E-19	0.75401
7	5.59988E-17	0.73653
8	5.66459E-15	0.74585
9	2.42992E-13	
10	4.11144E-12	
11	5.86855E-11	
12	3.41228E-10	

Table 9: Second best combined MRG of order 3 found for Example 7.

Component	a_1	a_2	a_3
1	9793152422	0	-1205362420
2	0	1545957508	-4123666983

t	d_t	S_t
4	3.76340E-29	0.79768
5	1.89861E-23	0.74711
6	1.08442E-19	0.77475
7	5.45485E-17	0.75611
8	5.75317E-15	0.73436
9	2.45100E-13	
10	4.33655E-12	
11	4.58516E-11	
12	3.05231E-10	

Acknowledgments

This work has been supported by NSERC-Canada grant # ODGP0110050 and FCAR-Québec grant # 93ER1654 to the first author. We wish to thank L. Afflerbach and H. Grothe for helpful discussions (in 1989) and the anonymous referees for their help. François Blouin, Marco Jacques, Armand Nkendjuo, François Paradis, and Josée Turgeon have participated in the development of this software over a period of seven years.

References

[1] L. Afflerbach and H. Grothe, 1985. Calculation of Minkowski-reduced lattice bases, *Computing*, 35, 269–276.

[2] L. Afflerbach and H. Grothe, 1988. The lattice structure of pseudo-random vectors generated by matrix generators, *Journal of Computational and Applied Mathematics*, 23, 127–131.

[3] W. A. Beyer, R. B. Roof, and D. Williamson, 1971. The lattice structure of multiplicative congruential pseudo-random vectors, *Mathematics of Computation*, 25, 345–363.

[4] P. Bratley, B. L. Fox, and L. E. Schrage, 1987. A Guide to Simulation. second edition, Springer-Verlag, New York.

[5] R. Couture and P. L'Ecuyer, 1996. Computation of a shortest vector and Minkowski-reduced bases in a lattice. In preparation.

[6] R. Couture and P. L'Ecuyer, 1996. Orbits and lattices for linear random number generators with composite moduli, *Mathematics of Computation*, 65, 189–201.

[7] U. Dieter, 1975. How to calculate shortest vectors in a lattice, *Mathematics of Computation*, 29, 827–833.

[8] U. Fincke and M. Pohst, 1985. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis, *Mathematics of Computation*, 44, 463–471.

[9] G. S. Fishman, 1990. Multiplicative congruential random number generators with modulus 2^β: An exhaustive analysis for $\beta = 32$ and a partial analysis for $\beta = 48$, *Mathematics of Computation*, 54, 331–344.

[10] G. S. Fishman and L. S. Moore III, 1986. An exhaustive analysis of multiplicative congruential random number generators with modulus $2^{31} - 1$, *SIAM Journal on Scientific and Statistical Computing*, 7, 24–45.

[11] H. Grothe, 1988. Matrixgeneratoren zur Erzeugung gleichverteilter Pseudozufallsvektoren, Dissertation (thesis), Tech. Hochschule Darmstadt, Germany.

[12] A. GRUBE, 1973. Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen, *Zeitschrift für angewandte Mathematik und Mechanik*, 53, T223–T225.

[13] P. M. GRUBER AND C. G. LEKKERKERKER, 1987. Geometry of Numbers, North-Holland, Amsterdam.

[14] D. JÄGER, 1992. *MCS Modula-2 Cross System, User's Guide.* La Chanenche, France.

[15] D. E. KNUTH, 1981. The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Second edition, Addison-Wesley, Reading, Mass.

[16] P. L'ECUYER, 1988. Efficient and portable combined random number generators, *Communications of the ACM*, 31, 742–749 and 774. See also the correspondance in the same journal, 32, 8 (1989) 1019–1024.

[17] P. L'ECUYER, 1990. Random numbers for simulation, *Communications of the ACM*, 33, 85–97.

[18] P. L'ECUYER, 1994. Uniform random number generation, *Annals of Operations Research*, 53, 77–120.

[19] P. L'ECUYER, 1996. Bad lattice structures for vectors of non-successive values produced by some linear recurrences, *ORSA Journal on Computing.* To appear.

[20] P. L'ECUYER, 1996. Combined multiple recursive generators, *Operations Research.* To appear.

[21] P. L'ECUYER, F. BLOUIN, AND R. COUTURE, 1993. A search for good multiple recursive random number generators, *ACM Transactions on Modeling and Computer Simulation*, 3, 87–98.

[22] P. L'ECUYER AND S. CÔTÉ, 1991. Implementing a random number package with splitting facilities, *ACM Transactions on Mathematical Software*, 17, 98–111.

[23] P. L'ECUYER AND R. COUTURE, 1996. LatMRG user's guide, a toolkit for theoretical testing of linear congruential and multiple recursive generators, Technical report, Montreal, Canada. In preparation.

[24] P. L'ECUYER, G. PERRON, AND F. BLOUIN, 1988. Sentiers: Un logiciel modula-2 pour l'arithmétique sur les grands entiers, Technical Report DIUL-RT-8802, Compter Science Department, Laval University, Ste-Foy (Que.), Canada.

[25] P. L'ECUYER AND S. TEZUKA, 1991. Structural properties for two classes of combined random number generators, *Mathematics of Computation*, 57, 735–746.

[26] G. MARSAGLIA, 1968. Random numbers fall mainly in the planes, *Proceedings of the National Academy of Sciences of the United States of America*, 60, 25–28.

[27] H. Niederreiter, 1986. A pseudorandom vector generator based on finite field arithmetic, *Mathematica Japonica*, 31, 759–774.

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

[29] B. D. Ripley, 1983. The lattice structure of pseudo-random number generators, *Proceedings of the Royal Society of London, Series A*, 389, 197–204.

[30] S. S. Ryshkov, 1972. On hermite, Minkowski, and Venkov reduction of positive quadratic forms in n variables, *Soviet Math. Doklady*, 13, 1676–1679. (Russian Translation).