Title

Design, Search and Implementation of High-dimension, Efficient, Long-cycle and Portable Uniform Random Variate Generator

Authors

Deng, Lih-Yuan
Xu, Hong Q

Publication Date

2005

A System of High-dimensional, Efficient, Long-cycle and Portable Uniform Random Number Generators

LIH-YUAN DENG
The University of Memphis
and
HONGQUAN XU
University of California, Los Angeles

We propose a system of multiple recursive generators of modulus p and order k where all nonzero coefficients of the recurrence are equal. The advantage of this property is that a single multiplication is needed to compute the recurrence, so the generator would run faster than the general case. For $p = 2^{31} - 1$, the most popular modulus used, we provide tables of specific parameter values yielding maximum period for recurrence of order $k = 102$ and 120. For $p = 2^{31} - 55719$ and $k = 1511$, we have found generators with a period length approximately $10^{14100.5}$.

Categories and Subject Descriptors: F.2.1 [**Theory of Computation**]: Numerical Algorithms and Problems– *computations in finite fields*; G.3 [**Probability and Statistics**]: Random Number Generation

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: DX-k generator, FMRG-k generator, linear congruential generator, multiple recursive generator, MT19937, portable and efficient generator, primitive polynomial.

1. INTRODUCTION

Linear Congruential Generators (LCGs), proposed by Lehmer [1951], are known to have several defects such as a relatively short cycle by today's standard, and questionable empirical performances. In Section 2, we review Multiple Recursive Generators (MRGs), which are natural extensions of LCGs in that the next value is computed iteratively from the previous k values. MRGs with maximum period are known to have a much longer period than LCGs and have an equidistribution property over a higher dimension. However, they may be less efficient than LCGs because several multiplications are needed. To improve the speed of generation, Grube [1973], L'Ecuyer and Blouin [1988], L'Ecuyer, Blouin and Couture [1993] considered MRGs with two non-zero terms. Recently, Deng and Lin [2000] proposed a special form of MRGs, called FMRG (Fast MRGs), which require only a single

Authors' addresses: Lih-Yuan Deng, Department of Mathematical Sciences, The University of Memphis, Memphis, TN 38152-3240, email: lihdeng@memphis.edu. Hongquan Xu, Department of Statistics, University of California, Los Angeles, CA 90095-1554, email: hqxu@stat.ucla.edu.

multiplication and are almost as efficient as LCGs.

In Section 3, we propose a system of special MRGs, called DX-k generators, where all nonzero coefficients of the recurrence are equal and k is the order of recurrence. This is motivated by the fact that computer multiplication is more time consuming than computer addition/subtraction. By requiring a common coefficient for a DX-k generator, we achieve high efficiency with a single multiplication.

In Section 4, we discuss the bottleneck of the commonly used criteria given in Knuth [1998, page 30]: It requires the complete factorization of $p^k - 1$, which can be difficult when p and k are large. Using some powerful computer factorization programs recently developed, we find all factors of $p^k - 1$ for several (but not all) values $k \leq 120$. In particular, we report a complete factorization of $p^k - 1$, with $p = 2^{31} - 1$, for $k = 102$ and $k = 120$. Subsequently, we discuss the effort and result of finding DX-102 and DX-120 generators. For example, we have found many DX-120 generators, each with the maximum period of $p^{120} - 1 \approx 0.679 \cdot 10^{1120}$.

In Section 5, we describe a C implementation of DX-k random number generators. Their key component code and some C-directives are given. The general implementation does not depend on a specific choice of k, modulus p, or the multiplier. We also discuss the issue of efficiency improvement by exploring a special form of the multiplier and the modulus $p = 2^{31} - 1$. Finally, in Section 6, we briefly discuss the possibility of extending the limit of k by removing the constraint on p. The main idea is to avoid the problem of factoring $p^k - 1$ by selecting p so that $(p^k - 1)/(p - 1)$ is a prime number. Checking whether a huge number is a prime is known to be much easier than finding its factorization (see, for example, Crandall and Pomerance [2000]). For $k = 1511$ and $p = 2^{31} - 55719$, we have indeed found a DX-1511 random number generator with a period length approximately $10^{14100.5}$.

2. RECENT RANDOM NUMBER GENERATORS

2.1 Multiple Recursive Generators

A *multiple recursive generator* (MRG), is defined as

$$X_i = (\alpha_1 X_{i-1} + \cdots + \alpha_k X_{i-k}) \bmod p, \quad i \geq k \tag{1}$$

for any initial seeds $(X_0, \ldots, X_{k-1})$, not all of them being zero. Here the modulus p is a large prime number and X_i can be transformed using $U_i = X_i/p$. Other transformations will be discussed later. It is well-known that the maximum period of an MRG is $p^k - 1$, reached if the polynomial

$$f(x) = x^k - \alpha_1 x^{k-1} - \cdots - \alpha_k, \tag{2}$$

is a primitive polynomial. See, for example, Knuth [1998], Zierler [1959], Golomb [1967], Grube [1973], L'Ecuyer [1990], and Lidl and Niederreiter [1986] for more details about primitive polynomials and MRGs.

It is well-known that a maximum period MRG has the property of equidistribution up to k dimensions (see, for example, Lidl and Niederreiter [1986, page 240]).

2.2 Fast Multiple Recursive Generators

While the maximum period is increased to $p^k - 1$, an MRG may be less efficient because it needs several multiplications whereas an LCG needs only one multiplica-

tion. To improve the efficiency of MRGs, Grube [1973], L'Ecuyer and Blouin [1988], L'Ecuyer, Blouin and Couture [1993] considered only two nonzero coefficients α_j and α_k $(1 \le j < k)$ of the MRG in (1) and provided portable implementations of MRGs satisfying these conditions. Deng and Lin [2000] proposed to set as many coefficients α_i in an MRG to be 0 and/or ± 1 as possible. In particular, they proposed a Fast MRG (FMRG) which is a special MRG with maximum period $p^k - 1$ of the form

$$X_i = (BX_{i-k} \pm X_{i-1}) \bmod p, \quad i \ge k. \tag{3}$$

An FMRG is fast because it requires one multiplication and one addition/subtraction. In addition, they proposed to restrict the multiplier B in (3) (say $B \le \sqrt{p}$) for efficiency and portability.

According to L'Ecuyer [1997], a necessary (but not sufficient) condition for an MRG to have a good lattice structure is that the sum of squares of coefficients, $\sum_{i=1}^{k} \alpha_i^2$, is large. This implies that an FMRG generator does not have a desirable lattice structure because it has only two nonzero coefficients B and ± 1. Therefore, it is of interest to extend the limit of B to a much larger value and/or to add more nonzero terms while maintaining efficiency and portability.

3. DESIGN OF DX RANDOM NUMBER GENERATORS

Similar to FMRGs, we consider a special efficient MRG with maximum period $p^k - 1$. Ideally, we would like to have many nonzero terms without significantly increasing its computing time. A simple way is to choose all nonzero coefficients in (1) to be equal so that only one multiplication is required. This leads us to a general class of efficient MRGs as described next.

3.1 DX random number generators

Formally, a DX-k-s generator is a special MRG in (1) that has s nonzero coefficients, all of them being equal. To pinpoint the nonzero coefficients, we require that their indices be about $k/(s-1)$ apart. Therefore, coefficients in a DX-k-s generator can be written as

$$\alpha_1 = \alpha_{\lfloor k/(s-1) \rfloor} = \alpha_{\lfloor 2k/(s-1) \rfloor} = \cdots = \alpha_{\lfloor (s-2)k/(s-1) \rfloor} = \alpha_k = B,$$

where $\lfloor x \rfloor$ denotes a floor function, where x is truncated to an integer. In this paper, we consider only $s \le 4$ with the following four special cases:

(1) FMRG-k $(\alpha_1 = 1, \alpha_k = B)$

$$X_i = BX_{i-k} + X_{i-1} \bmod p, \quad i \ge k. \tag{4}$$

(2) DX-k-2 $(\alpha_1 = \alpha_k = B)$

$$X_i = B(X_{i-k} + X_{i-1}) \bmod p, \quad i \ge k. \tag{5}$$

(3) DX-k-3 $(\alpha_1 = \alpha_{\lfloor k/2 \rfloor} = \alpha_k = B)$

$$X_i = B(X_{i-k} + X_{i-\lfloor k/2 \rfloor} + X_{i-1}) \bmod p, \quad i \ge k. \tag{6}$$

(4) DX-k-4 $(\alpha_1 = \alpha_{\lfloor k/3 \rfloor} = \alpha_{\lfloor 2k/3 \rfloor} = \alpha_k = B)$

$$X_i = B(X_{i-k} + X_{i-\lfloor 2k/3 \rfloor} + X_{i-\lfloor k/3 \rfloor} + X_{i-1}) \bmod p, \quad i \ge k. \tag{7}$$

While FMRG-k does not formally belong to the DX-k-s class, we still refer it as DX-k-1 for convenience. The family of generators is referred as DX-k, when the value of k is specified. We refer to the collection of DX-k generators as a system of DX random number generators.

3.2 Limits on B for DX-k-s

We now turn our attention to the limit on B. According to the ANSI/IEEE Standard 754-1985, the IEEE double precision floating point standard representation requires a 64-bit word, which are numbered from 0 to 63, left to right. The first bit is the sign bit, the next 11 bits are the exponent bits, and the final 52 bits are the fraction (mantissa). Note that the multiplication of two 31-bit numbers may yield a number 62-bit long, which cannot be stored exactly in double-precision.

It is straightforward to see that if $p < 2^{31}$ and B satisfies the following condition

$$B < 2^e, \quad \text{where } e = 20, \text{ when } s = 1, 2; \quad e = 19, \text{ when } s = 3, 4, \tag{8}$$

then the double precision implementation of the DX-k-s will keep precise results in the 52-bit mantissa. Another possibility is to consider the special form

$$B = \pm 2^r \pm 2^w. \tag{9}$$

A multiplier of this form was first suggested by Wu [1997] for LCGs. It can result in a fast computation by using only shift and addition operations. However, L'Ecuyer and Simard [1999] correctly pointed out that the successive values in the output of LCGs show strong dependence on corresponding Hamming weights. It would be interesting to know whether the successive values in the output of the DX-k-s generators have such weakness.

3.3 $U(0, 1)$ Generators

To generate a uniform variate, we need to scale X_i to a number U_i between 0 and 1. Since $0 \le X_i \le p - 1$, there are three reasonable choices:

$$(a)U_i = X_i/p, \quad (b)U_i = X_i/(p-1), \quad (c)U_i = (X_i + 0.5)/p = X_i/p + H, \tag{10}$$

where $H = 1/(2p)$. Here, (a), (b), (c) are referred as $U[0, 1)$, $U[0, 1]$, and $U(0, 1)$ generators, respectively.

We recommend a $U(0, 1)$ generator as in (10)(c) because it has three major advantages: (i) there is no possibility of obtaining $U_i = 0$ or $U_i = 1$, (ii) the average of U_i over its entire period is closer to $1/2$, and (iii) the range of U_i, $1/(2p) = H \le U_i \le 1 - H$, is symmetric around $1/2$. Generating either 0 or 1 may cause problems in certain applications. For example, it is common to use $X = -\ln(U)$ to generate a random variable with a standard exponential distribution. To generate a random variable with a logistic distribution, one usually uses $Y = \ln(U) - \ln(1 - U)$. In both cases, a $U[0, 1)$ or $U[0, 1]$ generator will fail when $U = 0$ is generated.

3.4 Statistical Justification

Regardless of how we scale the X_i into U_i, using any of equation (10), one has

$$U_i = (\alpha_1 U_{i-1} + \cdots + \alpha_k U_{i-k} + d) \bmod 1, \tag{11}$$

where d is simply a constant. Treating U_i as a continuous random variable, Deng and George [1990] and Deng, Lin, Wang and Yuan [1997] provided some theoretical and heuristic justification of the asymptotic uniformity and independence of the generator in (11). Their study shows that a good MRG should have many terms with large coefficients. L'Ecuyer [1997] further pointed out that large multiplier is only a necessary condition and other additional conditions are required.

Consequently, in theory, one should consider an MRG that has as many terms as possible with large coefficients. However, the more terms an MRG has, the less efficient it is, and the harder to design a portable program. In practice, one needs to balance the tradeoff among various criteria such as "portability", "efficiency", "empirical", and "theoretical" properties. As evidenced by popular generators proposed by L'Ecuyer [1999], it is usually sufficient to use an MRG of two or three terms. We believe that the restriction, $s \leq 4$, imposed on DX-k-s generators is reasonable.

4. SEARCH FOR DX RANDOM NUMBER GENERATORS

4.1 Algorithm to search for DX random number generators

To search for DX random number generators, we need to check whether $f(x)$ in (2) is a primitive polynomial. A search algorithm usually verifies a set of conditions (Alanen and Knuth [1964] and Knuth [1998, page 30]):

(i) $(-1)^{k-1}\alpha_k$ must be a primitive root mod p.

(ii) $x^R = (-1)^{k-1}\alpha_k \mod (f(x), p)$, where $R = (p^k - 1)/(p - 1)$.

(iii) For each prime factor q of R, the degree of $x^{R/q} \mod (f(x), p)$ is positive.

The major bottleneck in the above criteria is to find a complete factorization of $p^k - 1$. That may be very time consuming when k and p are large. While p can be any prime number, it is common to choose $p = 2^{31}-1$, the largest prime number that can be stored as a signed integer in a 32-bit computer word. Deng and Rousseau [1991] gave a complete listing of $p^k - 1$ with the ten largest prime numbers below 2^{31} and $k \leq 6$. As k becomes larger, finding the complete factorization becomes harder.

There are various powerful factorization programs available on different web sites. With their help, we find a complete factorization of $p^k - 1$, $p = 2^{31} - 1$, for many (but not all) values of $k \leq 120$. We report only the results for $k = 102$ and $k = 120$ because they are the largest values of k found so far.

4.2 Factoring $p^{102} - 1$ and $p^{120} - 1$ with $p = 2^{31} - 1$

Due to the space limitation, we will not tabulate the complete factorizations of $p^{102} - 1$ and $p^{120} - 1$ with $p = 2^{31} - 1$. They can be found at authors' web sites: `http://www.cs.memphis.edu/~dengl/` and `http://www.stat.ucla.edu/~hqxu/`.

For $p^{102}-1$, we label its four largest prime factors with 62, 131, 295 and 297 digits as $p62$, $p131$, $p295$ and $p297$, respectively. Note that $p^{102} - 1 \approx 0.719 \cdot 10^{952}$ has 952 digits. Generally speaking, factoring such a huge number is almost impossible with current computer technology. Fortunately, the four largest prime factors can be found separately in a simple polynomial factorization of $p^{102} - 1$. In particular, $p62$ was found in $p^{17} - 1$, $p131$ in $p^{17} + 1$, $p297$ in $(p^{51} - 1)/(p^{17} - 1)$, and $p295$ in

Table 1 Listing of various types of B for DX-k-s random number generators

k	s	(a) $\min B$	(b) $\max B < \sqrt{p}$	(c) $\max B < 2^e$	(d) (r,w)	(r,w)	(r,w)
102	1	820	46329	1048554	(20,9)	(29,9)	(30,25)
102	2	23	45787	1047849	(20,16)	(30,22)	
102	3	358	45537	523905	(29,24)		
102	4	721	46299	524076	(18,14)	(29,11)	
120	1	335	44771	1047690	(26,-22)	(28,-16)	
120	2	33	46213	1048555	(20,9)	(28,-16)	
120	3	392	46116	522630	(21,-8)		
120	4	1441	45546	521673	(21,17)		

$(p^{51}+1)/(p^{17}+1)$. Then, a complete factorization of $p^{102}-1$ with $p=2^{31}-1$ was found.

For $p^{120}-1$ with $p=2^{31}-1$, let $p39$, $p51$, $p65$, $p112$ and $p278$ be the five largest prime factors with 39, 51, 65, 112 and 278 digits, respectively. Again, these large prime factors are contained in some simple polynomial factors of $p^{120}-1$. In particular, $p51$ and $p278$ were found in the first and the second polynomial factor of $(p^{60}+1)/(p^{20}+1) = (p^8-p^4+1)\times(p^{32}+p^{28}-p^{20}-p^{16}-p^{12}+p^4+1)$; $p112$ and $p39$ in $(p^{30}+1)/(p^{10}+1)$; and $p65$ in $(p^{30}-1)/(p^{10}-1)$. The hardest part of the factorization is finding $p112$ and $p39$ because they are the only two prime factors contained in the second polynomial factor of

$$(p^{30}+1)/(p^{10}+1) = (p^4-p^2+1) * (p^{16}+p^{14}-p^{10}-p^8-p^6+p^2+1).$$

It took more than one week for 35 PCs (700 MHz Pentium III) running concurrently to find them. The remaining prime factors are less than 35 digits and were not difficult (relatively speaking, of course) to find.

4.3 Listing of DX-102 and DX-120 random number generators

We have performed a complete search for DX-102-s and DX-120-s generators with $s=1,2,3,4$ for $0<B<\sqrt{p}$. We have found a total of 377 and 239 values of $B<\sqrt{p}$ within DX-102 and DX-120, respectively. To save space, we will only report $\min B$ and $\max B$ in each of DX-k-s, $s=1,2,3,4$. A complete table is available from the authors' web sites.

In addition, we search for $\max B$ below the upper limit $B<2^e$ as in (8). Finally, we also explore the possibility of $B=2^r\pm 2^w$ as in (9) within each DX-k-s, $s=1,2,3,4$.

Table 1 lists values of B with (a) $\min B$, (b) $\max B<\sqrt{p}$, (c) $\max B<2^e$ in (8), and (d) (r,w). Here, we use (r,w) for $B=2^r+2^w$, and $(r,-w)$ for $B=2^r-2^w$.

It should be clear that we do not recommend the general use for the values under "$\min B$" in column (a). However, they are useful to determine the "power" of empirical tests. The values under "$\max B<\sqrt{p}$" in column (b) are useful if one wishes to use only integer operation with the techniques by Payne, Rabung and Bogyo [1969] and L'Ecuyer [1988]. The values under "$\max B<2^e$" in column (c) are recommended for general purpose generators. As described next, they are quite simple to implement. Finally, $B=2^r\pm 2^w$ in column (d) are recommended for efficiency and portability. However, we need to code each generator individually and it only works for $p=2^{31}-1$.

In Deng and Lin [2000], only FMRG-k with $k\le 4$ were listed with maximum

period length up to $p^4 - 1 \approx 0.212 \cdot 10^{38}$. With DX-120, we extend the maximum period length to $p^{120}-1 \approx 0.679 \cdot 10^{1120}$. The DX-120-$s$ is a major improvement with extremely long cycle and the property of equidistribution up to 120 dimensions.

5. IMPLEMENTATION OF DX RANDOM NUMBER GENERATORS

Here we describe the main components of the C code implementing DX-120. In the authors' web sites, we have a list of complete C implementations for DX-k-s with various k and $s \leq 4$. The programs are carefully designed so that they can be easily modified to a new DX-k generator for any k or p.

5.1 C directives for various implementations

There are two ways to implement DX-k generators: (a) use the function `fmod(n,p)` in `<math.h>`, if n and p are stored as `double`, or (b) use the operation `n % p` if n and p are stored as `_int64` or `int64_t`, a data type of 64-bit integers. The former is the default version because it is available in any C/C++ compiler, whereas the latter is system-dependent but more efficient. In particular, `_int64` is available in MS C/C++ compiler, and `int64_t` is available in GNU C/C++ compiler. We include a C-directive so that the program can automatically choose an appropriate version and be portable across all computer systems with a C compiler that follows the IEEE-754 standard.

```
/*_int64/int64_t : machine dependent, see sys/types.h   */
/*_int64 used in MS-C, int64_t used in GNU-C            */
#if defined(_WIN32)
typedef _int64 XXTYPE;
#define DMOD(n, p) ((n) % (p))
#elif defined(__GNUC__)
typedef int64_t XXTYPE;
#define DMOD(n, p) ((n) % (p))
#else
#include <math.h>
typedef double XXTYPE;
#define DMOD(n, p) fmod((n), (p))
#endif
```

5.2 Initialization

Any 120 integer numbers, not all zero, can be used as initial seeds for DX-120. Here we provide an initialization routine that requires a single 32-bit number from the user to produce the required 120 initial seeds. To generate these numbers quickly, we use an LCG with the most popular multiplier 16807 to generate 119 remaining numbers.

```
#define KK 120          /* can be changed              */
#define PP 2147483647   /* 2^31-1,  can be changed     */
#define HH 1/(2.0*PP)
#define B_LCG 16807     /* for LCG, can be changed     */

/* internal buffer and status, initialized in su_dx() */
```

```
static XXTYPE XX[KK];           /* buffer */
static int II;                  /* index */
static int K12;                 /* used by u_dx3 */
static int K13, K23;            /* used by u_dx4 */

/* su_dx : Initialization of dx-k-s, using an LCG */
void su_dx(unsigned int seed)
{
    int i;
    if(seed==0) seed = 12345;           /* seed can not be zero */
    XX[0] = seed;
    for(i=1; i<KK; i++) XX[i] = DMOD( B_LCG *  XX[i-1],  PP);

    II = KK-1;                          /* running index */
    K12 = KK/2-1;                       /* used by u_dx3 */
    K13 = KK/3-1;    K23 = 2*KK/3-1;    /* used by u_dx4 */
}
```

Some workspace, of size 121 to 123, is needed to store the generated sequence and its running index. Using this LCG, none of 120 initial seeds produced will be zero. This should not present any problem because the empirical performances of a good random number generator should not be affected by the initial seeds. To explore other specific seeds for DX-120, the user needs to provide an initialization routine.

5.3 Generation

Once a generator is selected and initialized, we simply call the generator function provided whenever a uniform random variate is needed. The user is free to choose different types of DX-120-s as well as different values of B. We recommend the selection of $s = 4$ with the largest value of B given.

```
#define BB4  521673
double u_dx4(void)
{
    int II0 = II;
    if(++II >= KK)   II = 0;    /*wrap around running index */
    if(++K13 >= KK) K13 = 0;    /*wrap around K13*/
    if(++K23 >= KK) K23 = 0;    /*wrap around K23*/
    XX[II] = DMOD(BB4 * (XX[II]+XX[K13]+XX[K23]+XX[II0]), PP);
    return ((double) XX[II] /PP) + HH;
}
```

Clearly, DX-120-4 generators are slightly less efficient than DX-120-1 generators. The actual difference in computing times is system-dependent. On our PCs, the difference is in the range of 10% to 15%. If computing efficiency is the major concern, we recommend using $s = 2$ and/or some special efficient generators as described next.

5.4 More efficient generators

The code discussed previously can be easily changed to any B satisfying the condition in (8), any degree k, and any prime number $p < 2^{31}$. Next, we discuss how to improve the computing efficiency by exploring the special feature of $p = 2^{31} - 1$ and $B = 2^r \pm 2^w$. Here we describe a DX-120-2 generator with $B = 2^{20} + 2^9$.

```
static unsigned long  XX[KK];  /* buffer for u_dx2d */

unsigned long MODP(unsigned long z) {return (((z)&PP)+((z)>>31));}

#define MUL20(x) ( ((x)>>11) + ((x)<<20)&PP )
#define MUL9(x)  ( ((x)>>22) + ((x)<<9) &PP )
double u_dx2d()
{
    int II0 = II; unsigned long S ;
    if(++II >= KK)  II = 0;
    S = MODP(XX[II] + XX[II0]);
    XX[II] = MODP( MUL20(S) + MUL9(S));
    return ((double) XX[II] /PP) + HH;
}
```

This special form of $B = 2^r + 2^w$ and its implementation is borrowed from Wu [1997] and L'Ecuyer and Simard [1999]. There are several advantages for a generator of this kind: (a) it is efficient because neither multiplication nor modulo operation is needed, (b) the multiplier B can be very large, and (c) it remains portable with improved efficiency across different platforms because it uses data type `unsigned long`, not a system-dependent 64-bit integer type or a slower `double` type. The only drawback is that we need to code a specific generator separately and it works only for $p = 2^{31} - 1$.

6. EXTENDING DX RANDOM NUMBER GENERATORS

As noted previously, while it is common to choose $p = 2^{31} - 1$, we can select any prime number as p. In particular, we can choose p so that $(p^k - 1)/(p - 1)$ is a prime number. This implies, of course, that k must be a prime number. Using such p and k, we can avoid the problem of factoring $p^k - 1$. This idea was first considered in L'Ecuyer, Blouin, and Couture [1993] for $k \leq 7$, and later in L'Ecuyer [1999] for $k \leq 13$. Here, we are mainly interested in a much larger value of k. For each prime number k, we first find the largest prime $p < 2^{31}$ such that $(p^k - 1)/(p - 1)$ is also a prime number. As pointed out earlier, a primality check of a huge number is easier than its factorization. After k and p are fixed, we then search for a multiplier B in DX-k-s. We have found many generators in DX-k-s, for $k > 100$. Only $k = 1511$ is reported here because it is the largest value found so far. More comprehensive results will be reported elsewhere. For $k = 1511$, we find $p = 2^{31} - 55719 = 2147427929$ and $B = 521816$ in DX-1511-4:

$$X_i = 521816(X_{i-1511} + X_{i-1007} + X_{i-503} + X_{i-1}) \bmod 2147427929, \quad i \geq 1511.$$

The generator defined above has a period of $p^{1511} - 1 \approx 10^{14100.5}$ and equidistribution property up to 1511 dimensions. For comparison, MT19937 proposed by

Matumoto and Nishimura [1998] has a period of $2^{19937} - 1 \approx 10^{6001.6}$ and equidistribution property up to 623 dimensions.

Among the DX-k generators, the advantages of choosing $p = 2^{31} - 1$ are: (i) more efficient implementations are available and (ii) it is also the largest (signed) integer that can be stored in a 32-bit computer word. Thus, $p = 2^{31} - 1$ is the most popular choice. The main disadvantage is that it appears to be hard to extend k much further. Removing the limitation on p, however, we can find a much larger k for DX-k generators.

In summary, we propose a system of random number generators with the following desired properties: high dimensional uniformity, efficiency, long cycle, and portability. Clearly, in addition to the DX-k generators, other generators may also have these desired properties. The high-dimensional uniformity of a DX-k generator is achieved by using a large k, say > 100. A DX-k generator is almost as efficient as an LCG because only a single multiplication is required. It has long period because it is a maximum period MRG with a large k. Finally, it is portable so that one can produce the same sequence on various platforms.

ACKNOWLEDGMENT

The authors are very grateful to Pierre L'Ecuyer for very helpful discussion and correspondence. The comments and suggestions by two anonymous referees have also made a significant contribution to the improvement of this paper.

REFERENCES

Alanen, J. D., and Knuth, D. E. 1964. Tables of finite fields. *Sankhyā, Series A* 26, 305–328.

Crandall, R., and Pomerance, C. 2000. *Prime Numbers - A Computational Perspective.* Springer-Verlag, New York, NY.

Deng, L. Y., and George, E. O. 1990. Generation of uniform variates from several nearly uniformly distributed variables. *Communications in Statistics B* 19, 145–154.

Deng, L. Y., and Lin, D. K. J. 2000. Random number generation for the new century. *American Statistician* 54, 145–150.

Deng, L. Y., Lin, D. K. J., Wang, J., and Yuan, Y. 1997. Statistical justification of combination generators. *Statistica Sinica* 7, 993–1003.

Deng, L. Y., and Rousseau, C. 1991. Recent development in random number generation. *Proceedings of the 29th ACM Annual Southeast Regional Conference*, April 1991, 89–94.

Golomb, S. W. 1967. *Shift Register Sequence.* Holden-Day, San Francisco, CA.

Grube, A. 1973. Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen. *Z. fr angewandte Math. und Mechanik* 53, 223–225.

Knuth, D. E. 1998. *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.* 3rd ed. Addison-Wesley, Reading, MA.

L'Ecuyer, P. 1988. Efficient and portable combined random number generators. *Commun. ACM* 31, 742–748, 774.

L'Ecuyer, P. 1990. Random numbers for simulation. *Commun. ACM* 33, 85–97.

L'Ecuyer, P. 1997. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing* 9, 57–60.

L'Ecuyer, P. 1999. Good parameter sets for combined multiple recursive random number generators. *Operations Research* 47, 159–164.

L'Ecuyer, P., and Blouin, F. 1988. Linear congruential generators of order $k > 1$. *1988 Winter Simulation Conference Proceedings*, 432–439.

L'Ecuyer, P., Blouin, F., and Couture, R. 1993. A search for good multiple recursive linear random number generators. *ACM Trans. Math. Softw.* 3, 87–98.

L'Ecuyer, P., and Simard, R. 1999. Beware of linear congruential generators with multipliers of the form $a = \pm 2^q \pm 2^r$. *ACM Trans. Math. Softw.* 25, 367–374.

Lehmer, D. H. 1951. Mathematical methods in large-scale computing units. *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, Harvard University Press, Cambridge, MA, 141–146.

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

Matumoto, M., and Nishimura, T. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* 8, 3–20.

Payne, W. H., Rabung, J. R., and Bogyo, T. 1969. Coding the Lehmer pseudo number generator. *Commun. ACM* 12, 85–86.

Wu, P. C. 1997. Multiplicative, congruential random-number generators with multiplier $\pm 2^{k1} \pm 2^{k2}$ and modulus $2^p - 1$. *ACM Trans. Model. Comput. Simul.* 23, 255–265.

Zierler, N. 1959. Linear recurring sequences. *J. SIAM* 7, 31–48.

Received September, 2002; revised March, 2003; accepted ??, 2003.