Computing (2010) 89:27–43
DOI 10.1007/s00607-010-0097-9

64-Bit and 128-bit DX random number generators

Lih-Yuan Deng · Henry Horng-Shing Lu · Tai-Been Chen

Received: 23 June 2009 / Accepted: 30 May 2010 / Published online: 24 June 2010
© Springer-Verlag 2010

Abstract Extending 32-bit DX generators introduced by Deng and Xu (ACM Trans Model Comput Simul 13:299–309, 2003), we perform an extensive computer search for classes of 64-bit and 128-bit DX generators of large orders. The period lengths of these high resolution DX generators are ranging from 10^{1915} to 10^{58221}. The software implementation of these generators can be developed for 64-bit or 128-bit hardware. The great empirical performances of DX generators have been confirmed by an extensive battery of tests in the TestU01 package. These high resolution DX generators can be useful to perform large scale simulations in scientific investigations for various computer systems.

Keywords Combined generators · Empirical tests · Equidistribution · Linear congruential generator (LCG) · Multiple recursive generator (MRG) · MT19937

Mathematics Subject Classification (2000) 65C10 Random number generation

Communicated by X. Chen.

L.-Y. Deng
Department of Mathematical Sciences, The University of Memphis, Memphis, TN 38152, USA
e-mail: lihdeng@memphis.edu

H. H.-S. Lu (✉)
Institute of Statistics, National Chiao Tung University, Hsinchu 30010, Taiwan, ROC
e-mail: hslu@stat.nctu.edu.tw

T.-B. Chen
Department of Medical Imaging and Radiological Sciences, I-Shou University,
Kaohsiung 82445, Taiwan, ROC
e-mail: ctb@isu.edu.tw

1 Introduction

The quality of any simulation study depends heavily on the quality of the random number generators. Most, if not all, of algorithms in a computer simulation make an implicit assumption that one can produce a sequence of independent and identically distributed $U(0, 1)$ sequence. Because of its finite bit representation, a computer can only produce a *discrete* variate with a limited number of bits. As the computer architectures of CPUs are moving from 32 bits to 64 bits (or beyond), we should consider new random number generators that can increase the resolution of the simulated sequence. This may be an important consideration for those simulations demanding higher precision in various challenging research problems during this computation era.

In Sect. 2, we review recent random number generators proposed for 64-bit computer architectures. In Sect. 3, we review recent developments on the multiple recursive generators (MRGs) and a class of efficient DX generators of larger order which was introduced by [1]. In Sect. 4, we perform a computer search for classes of 64-bit and 128-bit DX generators of large orders. The period lengths of these DX generators are ranging from 10^{1915} to 10^{58221}. We then put these DX generators to an extensive testing and we report the results of excellent empirical performances in Sect. 5. In addition to the generating speed, we also consider various selection criteria, such as the period length, the high dimensional equidistribution property, the portability, theoretical justifications and empirical performances. Based on these criteria, these 64-bit and 128-bit DX generators turn out to be great choices for future computer simulations.

2 64-Bit random number generators

While 32-bit application programs are still the most popular, we expect the interest in 64-bit computing will be increasing and it will become dominant in the near future. Hence, we will develop good random number generators for 64-bit and 128-bit computer systems in this study.

2.1 The need for 64-bit generators

Linear congruential generator (LCG) has been proposed more than 50 years ago by [2]. LCG is based on a simple first order recurrence equation

$$X_i = BX_{i-1} \bmod p, \quad i \geq 1, \tag{1}$$

where X_0 is a non-zero integer as a staring seed. The parameters for the most well-known LCG are $p = 2^{31} - 1$ and $B = 16807$. Its period length is approximately 2.1×10^9 which is quite short by today's standard. Since then, there are numerous papers published in the literature on improving or replacing LCGs. Most of those generators proposed are 32-bit random number generators because 32-bit CPUs are still the most popular today.

There are several 32-bit generators with extremely long period length and good empirical performances proposed recently. In [3], a popular random number generator, called MT19937, was proposed. It has the property of 623-dimension equidistribution for the uniform pseudo-random number generator. In addition, its has an extremely long period of $2^{19937}-1$ which is approximately 10^{6001}. Several 32-bit DX generators of large order were proposed by [1,4,5]. In particular, the period length of DX-1597, proposed in [5], is approximately 10^{14903}. More description on DX generators will be given later.

As 64-bit CPUs/operating systems become more and more popular, there is a need to design generators with a finer resolution. Most, if not all, of algorithms to produce random variates implicitly assumed that one can generate a sequence of continuous random variates over $U(0, 1)$. While it is impossible to generate all the points in $(0, 1)$, a good random number generator should generate many different points in $(0, 1)$ to resemble a random variate with a continuous $U(0, 1)$ distribution. Therefore, it is desirable to produce a random variates with the maximum possible resolution. One way to increase the resolution is to consider 64-bit (or higher) RNGs which are discussed next.

2.2 Some 64-bit random number generators

One way to construct a 64-bit (or higher) generator from a 32-bit generator is to use the "digital (Tausworthe) method" (See, [6]). For example, for the sequence $\{X_i, i \geq 0\}$ generated by a LCG using Eq. (1) with a prime modulus $p = 2^{31}-1$, we can consider

$$u_i = X_{2i}/p + X_{2i+1}/p^2 \bmod 1, \quad i \geq 0,$$

as a 64-bit generator. In general, it is slower, its period length is reduced by half. Due to the size limitation of B for the 32-bit LCG, its lattice property is not as good as a 64-bit LCG as discussed next.

It is straightforward to consider a 64-bit LCG in (1) by choosing a prime modulus $p = 2^{63} - c$ or $p = 2^{64} - c$ and a multiplier B which will yield a maximum period of $p - 1$. Using spectral test and Beyer ratios as selection criteria, L'Ecuyer et al. [7] found some good parameters for 64-bit LCGs:

1. $B = 2307085864$, $p = 2^{63} - 25$, and
2. $B = 13891176665706064842$, $p = 2^{64} - 59$.

These LCGs are used mainly as a comparison with other generators proposed in that paper.

In addition to the problem of having a relatively short period, LCG is known to have a problem with a poor high dimensional distribution/lattice structure, see [8]. In addition, Hörmann [9] showed that LCGs are not recommended to generate random variates by the ratio of uniforms method, proposed by [10]. This problem is not limited by LCG and it can be alleviated by taking a larger modulus or other class of 64-bit random number generators.

To the best of our knowledge, there are only few other 64-bit random number generators proposed in the literature. Marsaglia and Tsang [11] proposed a 64-bit

"universal" random number generator which is a 64-bit extension of 32-bit generators proposed in [12]. It is based on a combined generator of a lagged Fibonancci sequence with another simple arithmetic sequence. Its overall period length is shown to be approximately 10^{202}.

Extending the procedure for 32-bit MT19937, Nishimura [13] listed several new parameters for 64-bit CPUs. While the period length is still $2^{19937} - 1 \approx 10^{6001}$, the dimension of equidistribution property is reduced from 623 (for 32-bit MT19937) to 311 (with 64-bit word). While the dimension is reduced by half, the number of bits considered for each number is doubled. Hence, the total number of bits considered remains the same.

For the 64-bit "universal" generator and MT19937 (either 32-bit or 64-bit), there is a slight probability that a zero will appear. A precautionary test may be required to exclude the possibility of generating zero. Doornik [14] proposed an efficient way to avoid generating zero by "type casting" unsigned random integers from MT19937 to signed integers without additional computing cost. For details, see [14].

L'Ecuyer [15] proposed another 64-bit generator, called *MRG63k3a*, by combining the following two generators:

$$X_i = 1754669720X_{i-2} - 3182104042X_{i-3} \bmod (2^{63} - 6645), \tag{2}$$

$$Y_i = 31367477935Y_{i-1} - 6199136374Y_{i-3} \bmod (2^{63} - 21129), \tag{3}$$

$$Z_i = (X_i - Y_i) \bmod (2^{63} - 6645). \tag{4}$$

The corresponding uniform (0,1) generator is

$$U_i = Z_i^*/(m_1 + 1), \quad m_1 = 2^{63} - 6645, \tag{5}$$

where $Z_i^* = Z_i$, if $Z_i > 0$ and $Z_i^* = m_1$, if $Z_i = 0$. The period length is about $10^{113.5}$.

The two generators corresponding to the sequences $\{X_i, i \geq 0\}$ in (2) and $\{Y_i, i \geq 0\}$ in (3) belong to a general class of generators, called multiple recursive generators, which we discuss next.

3 Multiple recursive generators

Multiple recursive generators (MRGs) have become widely used RNGs. They are based on the k-th order linear recurrence

$$X_i = (\alpha_1 X_{i-1} + \cdots + \alpha_k X_{i-k}) \bmod p, \quad i \geq k \tag{6}$$

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 $U_i = X_i/p$. To avoid the possibility of obtaining 0 or 1, one can use $U_i = (X_i + 0.5)/p$.

The maximum period of an MRG of order k as in (6) is known to be $p^k - 1$. See, for example, Knuth [16]. Maximal period MRG is equidistributed up to k dimensions as stated in [6]. That is, every m-tuple $(1 \leq m \leq k)$ of integers between 0 and $p - 1$

appears exactly the same number of times (p^{k-m}) over its entire period $p^k - 1$, with the exception of the all-zero tuple which appears one time less ($p^{k-m} - 1$).

The spectral test, as discussed in [16], has been the generally accepted standard to evaluate the global performance of LCGs or MRGs with small order. It is a useful criterion because it is measuring the minimum distance between successive parallel hyperplanes over a higher dimension. The equidistribution property of a maximal period MRG of order k as mentioned earlier is clearly a stronger condition than the spectral test. Therefore, it is important to find MRGs of large orders.

The sequences $\{X_i, i \geq 0\}$ in (2) and $\{Y_i, i \geq 0\}$ in (3) are examples of MRGs with order $k = 3$. L'Ecuyer [17] showed that the combined MRGs in (4) is a close approximation to another MRG with large multipliers and a large composite modulus. For two MRGs X_i and Y_i with corresponding modulus m_1 and m_2, L'Ecuyer [17] showed another type of combination

$$U_i = X_i/m_1 + Y_i/m_2 \bmod 1 \tag{7}$$

is exactly equivalent to another MRG with large multipliers and a large composite modulus.

Using the spectral test on the MRG, L'Ecuyer [17] suggested "Good Figure of Merit" as a selection criterion. It is measuring the degree of even distribution over a high dimensional unit hypercube. L'Ecuyer [18] recommended additional 64-bit combined MRGs with good parameters. In addition to the MRG63k3a, we list some of them below:

RNG name	k	m_1	m_2	m_3	Nonzero terms	Period
MRG63k3a	3	$2^{63} - 6645$	$2^{63} - 21129$	–	4	$10^{113.5}$
	5	$2^{63} - 15981$	$2^{63} - 594981$	–	6	$10^{189.4}$
	7	$2^{63} - 52425$	$2^{63} - 92181$	$2^{63} - 152541$	9	$10^{397.7}$

In general, one can improve the period length and the empirical performance of the combined generators by increasing the order (k) of the recurrence, number of nonzero terms, and number of components in a combined MRG. However, the generating efficiency tends to decrease accordingly. For example, the MRG63k3a in (5) can increase the period length from $m_1^3 - 1$ ($\approx 10^{56.9}$) to $10^{113.5}$. However, the MRG63k3a needs four multiplications, three modulus operations, one floating-point division, and several addition/subtraction operations. All the operations mentioned here are 64-bit operations. In terms of computation cost, it is generally true that a floating-point division operation is most expensive, followed by an integer modulus or a multiplication operation, then a subtraction or an addition operation. The magnitude of their relative costs is highly hardware dependent.

3.1 DX-k–s generators

Deng and Lin [19] proposed a fast MRG (FMRG) which is a maximal period MRG with minimal number terms of nonzero coefficient. FMRG is almost as efficient as the classical LCG. Only lower order k (up to 4) with relative small coefficients of FMRGs were reported in [19]. L'Ecuyer and Touzin [20] reported that such FMRGs failed

drastically for certain tests. Deng and Xu [1] proposed DX generators as a system of portable, efficient, and maximal period MRGs where coefficients of the nonzero multipliers are the same:

1. DX-k-1 [FMRG] ($\alpha_1 = 1, \alpha_k = B$).

$$X_i = X_{i-1} + BX_{i-k} \bmod p, \quad i \geq k. \tag{8}$$

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

$$X_i = B(X_{i-1} + X_{i-k}) \bmod p, \quad i \geq k. \tag{9}$$

3. DX-k-3 ($\alpha_1 = \alpha_{\lceil k/2 \rceil} = \alpha_k = B$).

$$X_i = B(X_{i-1} + X_{i-\lceil k/2 \rceil} + X_{i-k}) \bmod p, \quad i \geq k. \tag{10}$$

4. DX-k-4 ($\alpha_1 = \alpha_{\lceil k/3 \rceil} = \alpha_{\lceil 2k/3 \rceil} = \alpha_k = B$).

$$X_i = B(X_{i-1} + X_{i-\lceil k/3 \rceil} + X_{i-\lceil 2k/3 \rceil} + X_{i-k}) \bmod p, \quad i \geq k. \tag{11}$$

Here, $\lceil x \rceil$ is the ceiling function of x which is the smallest integer $\geq x$. For the class of DX-k–s, s is the number of terms for the coefficient B. For simplicity, we use DX for a class of such DX-k–s generators. Several 32-bit DX generators of a large order k with prime modulus $p = 2^{31} - c$ for some suitably chosen c have been found and listed in [4,5].

Since MRG and DX generators do not depend on the size of the prime modulus, it is straightforward to consider 64-bit (or higher) DX generators. However, the time and effort to find such generators increase dramatically. Some discussion on the parameters selection is given next.

3.2 Parameters selection for DX generators

For the sake of fast implementation, the prime modulus is selected slightly smaller than $p = 2^d$ where the choice of d depends on the word size (say, w) of a CPU. In general, we can either choose $d = w - 1$ for a signed integer and $d = w$ for an unsigned integer in a computer word of size w. In addition to the currently most popular $w = 32$, we consider two other computer word sizes: $w = 64$ and $w = 128$. Specifically, we choose the prime modulus p to be of the form $p = 2^{63} - c$ and $p = 2^{64} - c$ for 64-bit CPUs; $p = 2^{127} - c$ and $p = 2^{128} - c$ for 128-bit CPUs. The number of bits in p determines the "resolution" of the uniform random variate over its range (0, 1). To differentiate various resolution among DX generators, we will refer to them as DX(d) generators for $p = 2^d - c$. More restriction on p will be discussed next.

Following [4], we choose the smallest prime k for each interval of one hundred ranging from 101 up to 1511. For each value of k and d, we then find the smallest c for a prime $p = 2^d - c$ such that $R(k, p) = (p^k - 1)/(p - 1)$ is a probable-prime number.

We then verify the primality of $R(k, p)$ using probabilistic tests via some commercial packages such as *Maple* and *MATHEMATICA*. We further perform *industrial prime test* as proposed in [21]. The probability of making false positive error can be made to be much smaller than 10^{-200} with several independent probabilistic tests. This error probability is much smaller than the computer software error or the hardware error. According to [22], it can be safely accepted as a "prime" in all but the most sensitive practical applications. Throughout this paper, we are following this procedure to find prime modulus p as described here.

In addition, we require that both p and $Q = (p-1)/2$ are prime numbers. Here, Q is commonly called a Sophie–Germain prime number. The search of p for a (probable) prime of $(p^k - 1)/(p - 1)$ is motivated by the well-known fact that a primality check of a huge number is easier than its factorization. There have been several studies in the field of number theory on the primes of the form $(a^k - 1)/(a - 1)$ especially when a is a small number (from 2 up to 12) and it may not be a prime. In particular, when $a = 2$, it is known as a Mersenne number; when $a = 10$, it is decimal integer with all 1's and it is called a repunit number. See, for example, [23,24]. Finding a large prime a for applications in random number generation was first considered in [7] for $k \leq 7$ and later in [18] for $k \leq 13$. Using this idea and an efficient search algorithm, [4] found some DX(31)-k generators for order k up to 1511.

To design efficient and portable DX generators, it is common to impose a limit on the multiplier B for the DX generators. In general, it is better to have a large B but it is harder to have a portable and efficient implementation when B is large. In this paper, we choose $B < 2^b$ for DX(d) generators where $b = \lfloor d/2 \rfloor$. Here, $\lfloor x \rfloor$ is the floor function of x which is the largest integer $\leq x$. Imposing such an upper bound on B is a common technique to avoid the need for higher precision arithmetic when $B \leq \sqrt{p}$. See, for example, [25,26].

Throughout the remaining of this paper, we use DX-k to denote the general class of DX generators of order k. To differentiate various DX-k generators with a specific size of the prime modulus p, we refer to them as DX(d)-k generators.

4 Tables of DX generators

Using a similar procedure and an efficient search algorithm proposed in [4], we can find DX(d)-k–s generators with four prime number sizes ($d = 63$, 64, 127 and 128), fifteen different orders k up to 1511, and four values of s (1, 2, 3, 4). There are 60 generators found in each of the four classes of DX(d) generators.

4.1 DX(63) generators

For DX(63) generators, for each prime order k, we first choose a prime modulus of the form $p = 2^{63} - c$ so that both $R(k, p) = (p^k - 1)/(p - 1)$ is (probable) prime number and $(p - 1)/2$ is a prime. Once k and p have been selected, we then use the search algorithm proposed by [4] to search for the multiplier B starting from the upper bound of $2^{31} - 1$ downward. We found various DX(63)-k–s generators for $s = 1, 2, 3, 4$ with

Table 1 List of k, c and $B < 2^{31}$ for DX(63)-k–s generators with $p = 2^{63} - c$

k	c	$\log_{10}(p^k - 1)$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
101	2941809	1915	2147483368	2147483606	2147483358	2147483434
211	969741	4002	2147483129	2147483390	2147483346	2147483557
307	3400329	5822	2147483549	2147483577	2147483009	2147483004
401	402105	7605	2147482138	2147481939	2147483261	2147482844
503	8175705	9539	2147483268	2147482176	2147479944	2147483019
601	3997821	11398	2147483420	2147483197	2147483049	2147482652
701	1137009	13294	2147482313	2147483513	2147481463	2147483063
809	6373005	15343	2147482662	2147483487	2147480247	2147482951
907	7416321	17201	2147482851	2147482426	2147483367	2147482515
1009	6182529	19136	2147483149	2147480890	2147474619	2147482952
1103	30158505	20918	2147481846	2147483393	2147480008	2147482724
1201	6186009	22777	2147473205	2147482568	2147483174	2147482893
1301	3241965	24673	2147482301	2147474911	2147483192	2147482137
1409	11522061	26722	2147482492	2147482526	2147481028	2147481062
1511	26619045	28656	2147483328	2147482443	2147471141	2147479114

the maximum period ranging from 10^{1915} to 10^{28656}. In total, there are 60 DX(63)-k–s generators listed in Table 1.

4.2 DX(64), DX(127) and DX(128) generators

Similarly, for DX(64) generators, we choose the prime modulus of the form $p = 2^{64} - c$ for each prime order k. For each k and p selected, we then search for the multiplier B starting from the upper bound of $2^{32} - 1$ (more precisely, $\sqrt{p}$) downward. In order to apply the technique proposed by [25,26], we need to make sure $B < \sqrt{p}$ which is most likely to hold because $\sqrt{p} \approx 2^{32}$. We find 60 DX(64)-k–s generators for $s = 1, 2, 3, 4$ with the maximum period ranging from 10^{1946} to 10^{29111} as listed in Table 2.

Following exactly the same procedure as in DX(63) and DX(64), we search for 128-bit generators with the prime modulus of the form $p = 2^{127} - c$ and $p = 2^{128} - c$. We first find 60 DX(127)-k–s generators for $s = 1, 2, 3, 4$ with $B < 2^{63}$ with the maximum period length ranging from 10^{3861} to 10^{57767} as listed in Table 3. We then find 60 DX(128)-k–s generators for $s = 1, 2, 3, 4$ with $B < 2^{64}$ with the maximum period length ranging from 10^{3891} to 10^{58221} as listed in Table 4.

Among the DX generators listed in Tables 1–4, the generators with the shortest period length are DX(63)-101 generators on the first row of Table 1 with the period length approximately 10^{1915}. The generators with the longest period length are DX(128)-1511 generators on the last row of Table 4. In particular, DX(128)-1511 generators have the length of 10^{58221} and they have the property of equidistribution over dimensions up to 1511. In contrast, the popular generator *MRG63k3a* in (4)

Table 2 List of k, c and $B < 2^{32}$ for DX(64)-k–s generators with $p = 2^{64} - c$

k	c	$\log_{10}(p^k - 1)$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
101	103709	1946	4294967293	4294966629	4294967266	4294966829
211	2323877	4065	4294967052	4294966680	4294966998	4294966783
307	9123149	5915	4294967295	4294966991	4294964840	4294967229
401	5109569	7726	4294967137	4294966905	4294967061	4294967162
503	610553	9691	4294966514	4294965530	4294967140	4294966521
601	1178813	11579	4294966786	4294967135	4294967290	4294965884
701	3863129	13505	4294965635	4294966321	4294964482	4294964225
809	17589113	15586	4294965606	4294964532	4294966247	4294967220
907	2012513	17474	4294966905	4294967254	4294959750	4294966316
1009	21298889	19439	4294960490	4294963149	4294965726	4294966465
1103	7366769	21250	4294961971	4294965233	4294965873	4294965920
1201	8355149	23138	4294966708	4294966586	4294966096	4294966963
1301	9528257	25065	4294966815	4294962681	4294961896	4294962561
1409	3454937	27146	4294964133	4294965185	4294965171	4294959534
1511	16445057	29111	4294966976	4294966049	4294955652	4294965548

Table 3 List of k, c and $B < 2^{63}$ for DX(127)-k–s generators with $p = 2^{127} - c$, where x = 184467440737095

k	c	$\log_{10}(p^k - 1)$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
101	8023365	3861	x5754	x5734	x5500	x5180
211	11501829	8067	x5540	x5744	x5774	x5101
307	12818949	11737	x4844	x5596	x4819	x5329
401	10064781	15331	x5653	x5476	x4767	x4845
503	154659081	19230	x5721	x4786	x5196	x5144
601	142397385	22977	x3801	x1385	x5168	x5792
701	31187169	26800	x4833	x4652	x4702	x3564
809	81710265	30929	x5068	x2890	x4963	x4053
907	26968581	34675	x4754	x5689	x4714	x4495
1009	2451789	38575	x5062	x5467	x4828	x5460
1103	253989021	42169	x2352	x4786	x4774	x0535
1201	29068281	45915	x0911	x1926	x5307	x6488
1301	79595481	49738	x5043	x5685	x1461	x2427
1409	21557661	53867	x5189	x2502	x4760	x4987
1511	185276181	57767	x5454	x0506	x5101	x4080

has a (relatively short) period length of $10^{113.5}$, it is not very efficient, and it has "reasonable" (but not exact) equidistribution property over (relatively) low dimensional space. In terms of the property of equidistribution over a high dimensional space or the period length, we believe that DX generators listed in Tables 1–4 are

Table 4 List of k, c and $B < 2^{64}$ for DX(128)-k–s generators with $p = 2^{128} - c$, where x = 184467440737095

k	c	$\log_{10}(p^k - 1)$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
101	781733	3891	x51370	x51579	x51485	x51432
211	14363333	8130	x50727	x50662	x51593	x50957
307	67573457	11829	x51010	x51560	x51314	x51127
401	9780293	15451	x51413	x50844	x50503	x51137
503	25760477	19381	x50953	x51566	x50913	x51548
601	29337077	23157	x51119	x51166	x51595	x50695
701	49288097	27010	x51584	x48278	x50028	x49708
809	440234213	31172	x50326	x51459	x51538	x50634
907	31065533	34948	x46471	x48411	x51610	x47425
1009	170478209	38878	x51101	x51425	x48893	x50881
1103	181533689	42500	x49650	x50537	x48697	x50453
1201	81181637	46276	x49772	x45824	x49236	x46375
1301	283176053	50129	x51436	x49542	x50139	x50009
1409	587284637	54291	x51417	x51201	x49479	x49996
1511	83322269	58221	x49588	x50441	x50388	x51364

preferred over the "universal" 64-bit generator proposed by [11] and the 64-bit MT19937 proposed by [13].

4.3 Combined DX RNGs

Similar to combined MRGs as proposed in [17], it is straightforward to consider the combination of two or more DX(63), DX(64), DX(127), and DX(128) generators. Taking DX(63) as an example, we can choose two k's ($\binom{15}{2} = 105$ choices) and sixteen compositions of s ($4 \times 4 = 16$ choices) from Table 1. Consequently, the total number combined generators can be constructed is 1680. The period length of the combined generator [similar to arguments for (5) or (7)] is the least common multiple of the two period lengths of the corresponding two DX(63) generators. Comparing with the MRG63k3a, the combined DX generators have much larger period length and they are also faster and easier to implement.

4.4 Selection of DX generators

If we have to choose between DX generators and their combined generators, we would recommend DX(d)-k–s generators, with a variety selection of d, k and s. DX generator is preferred mainly because of its already extremely long period, its generating efficiency, its high-dimensional equidistribution property, and its ability not to generate 0 or 1. It is straightforward to to avoid 0 or 1 for combined MRGs. See [17].

Generally speaking, increasing the prime modulus p which is of size 2^d, recurrence order k, and number of nonzero terms in a DX(d)-k–s generator tends to yield a generator with a better theoretical and/or empirical property. For example, it is clear that choosing a larger k will yield a much longer period length and the dimension of equidistribution is also larger. Increase the value of s tends to yield a better generator. According to [15], a necessary (but not sufficient) condition for a "good" MRG (better lattice structure over dimension larger than k) is that the sum of squares of coefficients, $\sum_{i=1}^{k} \alpha_i^2$, should be large. For DX generators, they all have an equidistribution property over dimension up to k. Larger value of s and B tends to yield a better lattice structure only when we consider lattice structure over the dimension larger than k. From a statistical justification viewpoint, Deng et al. [27,28] showed that a good MRG should have many terms with large coefficients. Finally, increasing the prime modulus p of size 2^d can greatly increase the "density" of possible generated points over any small interval. For example, we can increase more than 10^9 folds of density when we compare DX(32) generators and DX(64) generators. Therefore, larger value of d in the DX(d)-k–s generators is preferred.

There are some (mostly minor) negative aspects on increasing the values of d, k and s among the class of DX(d)-k–s generators. It tends to increase the difficulty of portable implementation, or increase the memory requirement, or decrease (somewhat) the generating efficiency.

5 Implementation and empirical evaluations

While 64-bit CPUs are becoming more and more popular, there is no mainstream 128-bit CPU available at the present time. However, we can use popular multi-precision software packages to implement DX(d) generators for any large d at the expense of a slower generating speed. In general, there are many folds difference of generating times between the hardware and software implementations.

5.1 Initialization

As explained in [1], one can choose any k initial values, not all are zeros, as a starting seeds for a DX(d)-k generator. To save time, we can use LCG with multiplier to be the same as the coefficient B found in the DX(d) generator to produce the required initial seeds. Based on [29], it is possible that initializing the MRG with an LCG is a bad idea. The structure of the LCG may easily show up in some initial segment of the output. To avoid some potential problems of using LCG to generate initial seeds, one can use a lower order MRG to generate the required initial seeds. In our opinion, a good RNG should not depend on a particular choice of initial seeds. Indeed, DX generators passed extensive empirical tests in TestU01, to be discussed next, even with various "bad" initial seed vectors.

5.2 Generation

It is straightforward to implement DX(63) or DX(64) generators for 64-bit CPUs. Because of the limit that we imposed on $B \leq \sqrt{p}$, we can also implement these

generators for 32-bit CPUs using a (popular but non-standard) 64-bit integer type which is available in `gcc` as type `long long` or in MS-C as type `__int64`. To implement 64-bit or 128-bit generators for 32-bit CPUs, we can use some popular general multi-precision packages such as NTL (http://www.shoup.net/) or GMP (http://www.swox.com/gmp/). NTL provided some user-defined multi-precision integer classes in C++ with numerous operator overloads so that one can use essentially the same code developed for 32-bit DX generator, available in http://www.cs.memphis.edu/~dengl/dx-rng/, without much modification.

5.3 Software development for DX generators

As correctly pointed out by the referee, it is important to develop a program module and make it available in one of the open-source software packages which are used by scientists all over the world. Based on study reports on the DX and related generators, we expect a much wider interest to adopt a program module for portable and efficient MRGs for 32-bit, 64-bit, and 128-bit computing platforms. In addition to developing program modules available for open-source software packages, we plan to maintain a web-site so that the users can download the source codes in various programming languages of DX generators. Information about the latest developments will be available at http://www.cs.memphis.edu/~dengl/dx-rng/.

5.4 Empirical evaluations

We use the comprehensive, predefined test module in TestU01 along with the general multi-precision package of GMP (http://www.swox.com/gmp/) to evaluate the empirical performance in our study. TestU01 was developed by Professor L'Ecuyer with the source code and user's manual available from http://www.iro.umontreal.ca/~lecuyer/. It is by far the most comprehensive test suite available today. We evaluate our DX generators running on the high performance computers with Pentium 4 running at 3.2 Ghz. The actual run time for the tests mentioned above depends on the computing hardware and software used. There are three predefined test modules in TestU01. The test module of *Small crush* produces 15 p-values and it takes only 1–2 min of computing time in the test PC. If the test module of Small crush is passed, we perform the larger test module of *Crush* which generates 144 p-values. Finally, we can perform the largest test module of *Big crush* which is most comprehensive with 160 p-values.

There are 60 generators for each of DX(63), DX(64), DX(127) and DX(128) generators found in Tables 1–4. For each generators tested, we apply Crush test module in TestU01 with two different starting seeds of 123 and 12345. Therefore, we obtain 17280 ($= 60 \times 144 \times 2$) p-values in Tables 1–4. The number of tests with p-values outside the range [0.001, 0.999] are tabulated in Table 5.

As one can see from the result of Table 5, DX generators have excellent empirical performances. None of these tests produce p-values that are too close to 0 or 1. Specifically, none of the 4×17280 p-values are smaller than 10^{-5} and only one p-value in the interval $(1-10^{-5}, 1-10^{-15})$. Additional interesting observations from Table 5 are given below:

Table 5 Results of Crush test module on DX(63), DX(64), DX(127) and DX(128) generators

p value	$>1-10^{-15}$	$>1-10^{-5}$	$>1-10^{-4}$	$>1-10^{-3}$	$<10^{-3}$	$<10^{-4}$	$<10^{-5}$
DX(63), (17280 p-values)							
Counts	0	1	3	16	14	4	0
Proportion	0	0.00006	0.00017	0.00093	0.00081	0.00023	0
DX(64), (17280 p-values)							
Counts	0	0	3	16	22	3	0
Proportion	0	0	0.00017	0.00093	0.00127	0.00017	0
DX(127), (17280 p-values)							
Counts	0	0	0	9	16	2	0
Proportion	0	0	0	0.00052	0.00093	0.00012	0
DX(128), (17280 p-values)							
Counts	0	0	4	17	23	2	0
Proportion	0	0	0.00023	0.00098	0.00133	0.00012	0
64-bit DX: DX(63)+DX(64), (17280 $\times$ 2 p-values)							
Count	0	1	6	32	36	7	0
Proportion	0	0.00003	0.00017	0.00093	0.00104	0.00020	0
128-bit DX: DX(127)+DX(128), (17280 $\times$ 2 p-values)							
Count	0	0	4	26	39	4	0
Proportion	0	0	0.00012	0.00075	0.00113	0.00012	0
64-bit and 128-bit DX, (17280 $\times$ 4 p-values)							
Count	0	1	10	58	75	11	0
Proportion	0	0.00001	0.00014	0.00084	0.00109	0.00016	0

1. The Proportion of p-values which are below 10^{-3} for DX(63), DX(64), DX(127), and DX(128) are 0.00081, 0.00127, 0.00093, and 0.00133, respectively. The overall Proportion is 0.00109 which is extremely close to its nominal value of 0.001.
2. Comparing 64-bit versus 128-bit DX generators, there appears no statistically significant difference between them. Counting the Proportion of p-values below 10^{-4}, we observed 0.00020 and 0.00012, respectively, for the 64-bit DX generators and 128-bit DX generators. It appears that the Proportion of p-values below 10^{-4} for 128-bit DX generators is closer to the corresponding nominal value of 10^{-4}.

No DX(63), DX(64), DX(127) or DX(128) generators were found to be clear failures by the TestU01 battery of tests. All DX generators listed in Tables 1–4 have passed the extensive battery of tests in the TestU01 package. As already reported in [5,30], DX(31) generators have also passed a battery of tests in the TestU01 package.

5.5 Testing MT19937

To compare the empirical performances of the DX generators, we apply Crush test module on MT19937 with various starting seeds. There are two versions of MT19937,

with different initialization routines, defined in TestU01 package. It was first reported in [30] that MT19937 and its related family of generators failed linear complexity tests in TestU01, with p-values larger than $1 - 10^{-15}$. Regardless of different starting seed and different initialization routine, MT19937 cannot pass the linear complexity test in TestU01. L'Ecuyer and Simard [30] also reported that any linear generator fails in linear complexity test. Thus, not only Mersenne Twister, generators such as GFSR (proposed by [31]) and WELL (proposed in [32]), they all fail in F_2 linear complexity test. However, it is believed that it is not a serious issue for the Monte Carlo simulation.

One may reasonably argue that whether DX will also fail in the linear complexity test modulo p. However, the p is not fixed for all DX generators and the nonzero coefficients for DX generators are large. Indeed, as we have shown earlier, all DX generators have passed battery of tests in TestU01.

Using the method of "combined generators" as considered in [26,33,34], one can consider combining MT19937 with other RNG to improve the empirical performance of MT19937 (or related family of generators). Taking the efficiency advantage of MT19937, we propose a simple and efficient method to improve the $U(0, 1)$ sequence $\{u_i, i \geq 0\}$ generated by MT19937:

$$w_i = u_{2i} + u_{2i+1} \bmod 1, \quad i = 0, 1, 2, \cdots$$

Usually, a combination generator is to combine two different generators so that it has a better properties such as increased period and a better lattice structure as in [17]. Since MT19937 has already a huge period length and a good lattice structure, it is reasonable to consider a simple combined generator with itself. The period length is not increased and we believe its lattice structure or its equidistribution property is not greatly affected. A statistical justification of the combined generators can be found in [27,28]. Most importantly, the "combined MT19937" (CMT19937, for short) generator indeed passed the battery of tests including the linear complexity test in TestU01. We describe the details of the empirical study on the CMT19937 next.

To evaluate the empirical performances of the CMT19937, we have applied Crush test module with 120 different starting seeds for 32-bit CMT19937 and 64-bit CMT19937, respectively. We chose 120 various seeds for each type of CMT19937 so that we can match the same number of p-values produced as in Table 5 for testing DX generators. There are $120 \times 144 = 17280$ p-values produced for each type of generators. The empirical results are summarized in Table 6.

Table 6 shows that the CMT19937 (32-bit or 64-bit) have pass the Crush test module including Linear Complexity tests in TestU01. We have also applied the big Crush test module on both 32-bit and 64-bit CMT19937. They have passed these tests.

5.6 Time comparisons

We compare the generation time for DX(63), CMT19937-64 and MT19937-64 generators on two platforms: (a) AMD Althlon 64 X2 dual CPU and DDR II 667 GHz 2 GB RAM in WinXP (b) HPC in Memphis, each CPU is 3.2 GHz Intel Xeon processors

Table 6 Results of Crush test on 32-bit/64-bit combined MT19937 with various starting seeds

p value	$>1-10^{-15}$	$>1-10^{-5}$	$>1-10^{-4}$	$>1-10^{-3}$	$<10^{-3}$	$<10^{-4}$	$<10^{-5}$
Combined 32-bit MT19937, (17280 p-values)							
Counts	0	0	2	18	17	2	0
Proportion	0	0	0.00012	0.00104	0.00098	0.00012	0
Combined 64-bit MT19937, (17280 p values)							
Counts	0	0	3	17	18	1	0
Proportion	0	0	0.00017	0.00098	0.00104	0.00006	0
32-bit and 64-bit combined MT19937, (17280 $\times$ 2 p values)							
Counts	0	0	5	35	35	3	0
Proportion	0	0	0.00014	0.00101	0.00101	0.00009	0

Table 7 Time comparisons on DX(63), CMT19937-64 and MT19937-64 generators on two platforms

No. of runs	MT19937-64		DX(63)		CMT19937-64	
	(a)	(b)	(a)	(b)	(a)	(b)
10^6	0.080	0.020	0.094	0.030	0.125	0.080
10^7	0.700	0.220	0.936	0.290	1.110	0.810
10^8	7.280	2.120	9.453	2.940	11.060	8.190

running in Linux. We run the test on generating 10^6, 10^7 and 10^8 random variates. The timing comparisons are tabulated in Table 7.

As we can see that the timing results are highly hardware dependent. The generating speed for DX(63) is somewhere between that of MT19937-64 and CMT19937-64. In platform (a) AMD Althlon 64 X2 dual CPU, the total time (in seconds) for generating 10^8 random variate are 7.280, 9.453, and 11.060 for MT19937-64, DX(63), and CMT19937-64, respectively. Running time under platform (b) appears to be much faster for both DX(63) and MT19937-64 but the relative ordering of the generating efficiency remains the same.

6 Conclusion and discussion

We extend the 32-bit DX generators pioneered by [1] to perform an extensive computer search for classes of 64-bit and 128-bit DX generators of large orders. The period lengths of these DX generators are ranging from 10^{1915} to 10^{58221}. Compared to generators like the MRG63k3a, DX generators are clearly more efficient by utilizing special and simpler forms of MRGs. Like any maximal period MRGs, DX generators have the nice property of equidistribution over high dimensions. If the lattice property over dimension higher than k is important, then, if possible, we need a proper selection of B for a better lattice structure. Alternatively, we can search for a generator with a larger k. The dimension of equidistribution for high precision DX generators found

is up to 1511. Finally, the great empirical performances of DX generators have been confirmed by an extensive battery of tests in the TestU01 package.

The 64-bit MT19937 and 64-bit universal RNG are fast. DX generators are also fast. Any comparison of the generating speed is highly hardware-dependent and one may observe different results on various 32-bit or 64-bit computing platforms. But the time difference of 1 or 2 s for a generation of 10^8 random variates should not be a major factor for the selection of good RNGs. In addition to the generating speed, it is very important to consider other factors such as the period length, the high dimensional equidistribution property, the portability, the scalability, theoretical justifications and empirical performances. Based on these criteria, these new 64-bit and 128-bit DX generators become great choices for large scale simulations encountered in challenging researches in any scientific investigation.

Acknowledgments This research was partially supported by the National Science Council, National Center for Theoretical Sciences and Center of Mathematical Modeling and Scientific Computing (CMMSC) at the National Chiao Tung University in Taiwan, R.O.C. This work was done while the first author was visiting Institute of Statistics, National Chiao Tung University, Hsinchu, Taiwan. We also acknowledge the usage of high speed computing facility for this research in the University of Memphis at USA and the National Center for High-Performance Computing at Taiwan, ROC.

References

1. Deng LY, Xu H (2003) A system of high-dimensional, efficient, long-cycle and portable uniform random number generators. ACM Trans Model Comput Simul 13:299–309
2. Lehmer DH (1951) Mathematical methods in large-scale computing units. In: Proceedings of the second symposium on large scale digital computing machinery. Harvard University Press, Cambridge, MA, pp 141–146
3. Matumoto M, Nishimura T (1998) Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans Model Comput Simul 8:3–20
4. Deng LY (2004) Generalized Mersenne prime number and its application to random number generation. In: Niederreiter H (ed), Monte Carlo and Quasi-Monte Carlo methods. Springer-Verlag, Berlin, pp 167–180
5. Deng LY (2005) Efficient and portable multiple recursive generators of large order. ACM Trans Model Comput Simul 15:1–13
6. Lidl R, Niederreiter H (1994) Introduction to finite fields and their applications. Revised edition. Cambridge University Press, Cambridge
7. L'Ecuyer P, Blouin F, Couture R (1993) A search for good multiple recursive linear random number generators. ACM Trans Model Comput Simul 3:87–98
8. Gentle JE (2003) Random number generation and Monte Carlo methods, 2nd edn. Springer-Verlag, New York
9. Hörmann W (1994) A note on the quality of random variates generated by the ratio of uniforms method. ACM Trans Model Comput Simul 4:96–106
10. Kinderman AJ, Monahan JF (1997) Computer generation of random variables using ratio of uniform deviates. ACM Trans Math Softw 3:257–260
11. Marsaglia G, Tsang W (2004) The 64-bit universal RNG. Stat Probab Lett 66:183–187
12. Marsaglia G, Zaman A, Tsang W (1990) Toward a universal random number generator. Stat Probab Lett 8:35–39
13. Nishimura T (2000) Tables of 64-bit Mersenne twisters. ACM Trans Model Comput Simul 10:348–357
14. Doornik J (2007) Conversion of high-period random numbers to floating point. ACM Trans Model Comput Simul 17:article 3
15. L'Ecuyer P (1997) Bad lattice structures for vectors of non-successive values produced by some linear recurrences. INFORMS J Comput 9:57–60

16. Knuth DE (1998) The art of computer programming, vol 2. Seminumerical algorithms, 3rd edn. Addison-Wesley, Reading
17. L'Ecuyer P (1996) Combined multiple recursive random number generators. Oper Res 44:816–822
18. L'Ecuyer P (1999) Good parameter sets for combined multiple recursive random number generators. Oper Res 47:159–164
19. Deng LY, Lin DKJ (2000) Random number generation for the new century. Am Stat 54:145–150
20. L'Ecuyer P, Touzin R (2004) On the Deng–Lin random number generators and related methods. Stat Comput 14:5–9
21. Damgard I, Landrock P, Pomerance C (1993) Average case error estimates for the strong probable prime test. Math Comput 61:177–194
22. Crandall R, Pomerance C (2000) Prime numbers: a computational perspective. Springer-Verlag, New York
23. Williams HC, Seah E (1979) Some primes of the form $(a^n - 1)/(a - 1)$. Math Comput 33:1337–1342
24. Brillhart J, Lehmer DH, Selfridge JL, Tuckerman B, Wagstaff SS Jr (2002) Factorizations of $b^n \pm 1$, b=2,3,5,6,7,10,11,12 up to high powers, 3rd edn. American Mathematical Society
25. Payne WH, Rabung JR, Bogyo T (1969) Coding the Lehmer pseudo number generator. Commun ACM 12:85–86
26. L'Ecuyer P (1988) Efficient and portable combined random number generators. Commun ACM 31:742–748, 774
27. Deng LY, George EO (1990) Generation of uniform variates from several nearly uniformly distributed variables. Commun Stat B 19:145–154
28. Deng LY, Lin DKJ, Wang J, Yuan Y (1997) Statistical justification of combination generators. Stat Sin 7:993–1003
29. Matsumoto M, Wada I, Kuramoto A, Ashihara H (2007) Common defects in initialization of pseudorandom number generators. ACM Trans Model Comput Simul 17:Article 15
30. L'Ecuyer P, Simard R (2007) TestU01: a C library for empirical testing of random number generators. ACM Trans Math Softw 33(22):1–40
31. Lewis TG, Payne WH (1973) Generalized feedback shift register pseudorandom number algorithm. J Assoc Comput Mach 20:456–468
32. Panneton F, L'Ecuyer P, Matsumoto M (2006) Improved long-period generators based on linear recurrences modulo 2. ACM Trans Math Softw 32:1–16
33. Wichmann BA, Hill ID (1982) An efficient and portable pseudo-random number generator. Appl Stat 31:188–190
34. Marsaglia G (1985) A current view of random number generators. In: Computer science and statistics, sixteenth symposium on the interface. North-Holland, Amsterdam. Elsevier, pp 3–10