Some long-period random number generators using shifts and xors

Richard P. Brent[1]

(Received 6 July 2006; revised 2 July 2007)

Abstract

Marsaglia recently introduced a class of 'xorshift' random number generators with periods $2^n - 1$ for $n = 32, 64, \ldots$. Here Marsaglia's xorshift generators are generalised to obtain fast and high quality random number generators with extremely long periods. Whereas random number generators based on primitive trinomials may be unsatisfactory, because a trinomial has very small weight, these new generators can be chosen so that their minimal polynomials have a large number of non-zero terms and, hence, a large weight. A computer search using Magma found good random number generators for n a power of two up to 4096. These random number generators are implemented in a free software package `xorgens`.

See http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/40 for this article, © Austral. Mathematical Soc. 2007. Published July 6, 2007. ISSN 1446-8735

Contents

1 Introduction

Marsaglia [11] proposed a class of uniform random number generators (RNGs) called 'xorshift RNGs'. Their implementation requires only a small number of left shifts, right shifts and 'exclusive or' operations per pseudo-random number.

Assume that the computer wordlength is w bits (typically $w = 32$ or 64). Marsaglia's xorshift RNGs have period $2^n - 1$, where n is a small multiple of w, say $n = rw$.

I showed that Marsaglia's xorshift RNGs are a special case of the well-known linear feedback shift register (LFSR) class of RNGs [3]. This was also observed by Panneton and L'Ecuyer [14]. However, the xorshift RNGs have implementation advantages because n (the number of state bits) is a multiple of the wordlength w. In contrast, for RNGs based on primitive trinomials, the corresponding parameter n can not be a multiple of eight (due to Swan's theorem [13, 15]) and is usually an odd prime. For example, $n = 19937$ in the case of the 'Mersenne twister' [12]. The 'tempering' step of the Mersenne twister can be omitted in the xorshift RNGs. Thus, the xorshift RNGs are simpler and potentially faster.

Any RNG based on a finite state must eventually cycle, but it is desirable for RNGs to have a very long period T (the cycle length). Most generators fail certain statistical tests if more than about $T^{1/2}$ random numbers are used [14], or perhaps even about $T^{1/3}$ numbers for the 'birthday spacings' test [9]. For generators satisfying linear recurrences such as the LFSR generators with period $2^n - 1$, there is a linear relationship between blocks of $n+1$ consecutive bits, so the generator may fail statistical tests that detect this linear relationship. Also, on a parallel machine we may want to use disjoint segments of the cycle on different processors, and if this is done by starting with different seeds on each processor, we want the probability that two segments overlap to be negligible. For all these reasons it is important for n to be large. The generators that we describe below have n as large as 4096 which is enough, but not so large that the generators are slowed down by memory accesses. This is possible if the computer's memory cache is smaller than n bits.

Marsaglia's original proposal [11] discusses mainly the case $n \le 64$, but an extension to larger n is suggested, I implemented a generalisation `xorgens` [4] with $n \le 4096$, in particular we can choose any power of two $n = 2^k$ for $6 \le k \le 12$. The problem in going to larger n is that we need to know the complete prime factorisation of $2^n - 1$ in order to be sure that the generator's period is maximal. These factorisations are known for all multiples of 32 up to 1632 and for certain larger n [16]. If we restrict n to powers of two then it is sufficient to know the factorisations of certain Fermat numbers $F_k = 2^{2^k} + 1$, since, for example

$$2^{4096} - 1 = (2^{2048} + 1)(2^{2048} - 1) = F_{11}(2^{2048} - 1) = \cdots = F_{11}F_{10}F_9 \cdots F_1F_0 \,.$$

The factorisations of the Fermat numbers $F_0, \ldots, F_{11}$ are known [2].

Panneton and L'Ecuyer [14] tested Marsaglia's xorshift RNGs and found certain deficiencies, but they did not find any significant problems with our xorgens generators for $n \geq 128$.

In Section 2 we introduce some notation, summarise the relevant theory, and describe the class of RNGs implemented in our xorgens package. In Section 3 we discuss criteria for the selection of 'optimal' generators in the class, and give specific examples of optimal generators for various $n \leq 4096$ and $w = 32$ and 64. Finally, Section 4 discusses a known weakness of xorshift RNGs and mentions some possible improvements.

2 Notation and theory

Let $F_2 = \mathrm{GF}(2)$ be the finite field with two elements $\{0, 1\}$. We usually write addition in F_2 as $+$, but we use $\oplus$ when it is necessary to distinguish it from normal integer addition. If 0 is interpreted as 'false' and 1 as 'true', then the field operations are 'exclusive or' (xor or $\oplus$) and 'and' ($\wedge$). A computer word of w bits can be regarded as a vector x of length w over F_2. We shall identify a bit-vector x with the corresponding integer (and vice-versa) when necessary.

Our RNGs generate pseudo-random bit-vectors x, but these easily give pseudo-random unsigned integers $x \in [0, 2^w)$, pseudo-random signed integers $x - 2^{w-1} \in [-2^{w-1}, 2^{w-1})$, or (by a linear transformation) pseudo-random real numbers in $(0, 1)$.

Unfortunately there are two conventions for bit-vectors x: Marsaglia [11] uses *row* vectors $x \in F_2^{1\times w}$, but Panneton and L'Ecuyer [14] use *column* vectors $x \in F_2^{w\times 1}$. We follow Marsaglia and take x as a row vector. (To convert to column vector notation, transpose all equations involving vectors and matrices.)

Fix parameters $r > s > 0$ (the choice of these will be discussed below), and consider the linear recurrence

$$x^{(k)} = x^{(k-r)}A + x^{(k-s)}B\,, \qquad (1)$$

where $x^{(k)} \in F_2^{1\times w}$. Here A and B are fixed matrices in $F_2^{w\times w}$. Given $x^{(0)}, \ldots, x^{(r-1)}$, the recurrence (1) uniquely defines the sequence $(x^{(k)})_{k\ge 0}$.

Let $L \in F_2^{w\times w}$ be the *left shift* matrix

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix}$$

such that

$$(x_1, \ldots, x_w)L = (x_2, \ldots, x_w, 0)\,.$$

Similarly, let $R = L^T$ be the *right shift* matrix such that

$$(x_1, \ldots, x_w)R = (0, x_1, \ldots, x_{w-1})\,.$$

Marsaglia's idea is to take A and B as products of a small number of terms such as $(I + L^a)$ and $(I + R^b)$. Specifically, let us take

$$A = (I + L^a)(I + R^b)$$

and

$$B = (I + L^c)(I + R^d)$$

for small positive integer parameters a, b, c, d. Marsaglia [11, §3.1] omits the factor $I + L^c$; we include it for reasons of symmetry, to increase the number of possible choices (see §3), and to improve properties related to Hamming weight (see §4).

With our choice of A and B, the recurrence (1) becomes

$$x^{(k)} = x^{(k-r)}(I + L^a)(I + R^b) + x^{(k-s)}(I + L^c)(I + R^d)\,. \qquad (2)$$

Note that if x is a bit-vector of length w, then xL^a is just x shifted left a places ($xL^a = 0$ if $a \geq w$), and $x(I + L^a)$ is the xor of x and xL^a. The operation

$$x \leftarrow x(I + L^a)$$

can be written in C as

```
x = x ^ (x << a)
```

or more succinctly as

```
x ^= x << a
```

(here x is represented in a computer word `x` which C treats as an unsigned integer). Similarly $x \leftarrow x(I + R^b)$ can be written in C as `x ^= x >> b`, and

$$x \leftarrow x(I + L^a)(I + R^b)$$

can be written in C as

```
x ^= x << a ;   x ^= x >> b .
```

The recurrence (1) is best implemented using a 'circular array'; that is, an array where the indices are computed mod r (see [7, §3.2.2]), unless r is very small.

It is well known that we can write the recurrence (1) as

$$(x^{(k-r+1)}|x^{(k-r+2)}|\cdots|x^{(k)}) = (x^{(k-r)}|x^{(k-r+1)}|\cdots|x^{(k-1)})\mathcal{C}\,, \tag{3}$$

where the *companion matrix* $\mathcal{C} \in F_2^{n\times n}$ can be regarded as an $r \times r$ matrix of $w \times w$ blocks (recall that $n = rw$). For example, if $r = 3$ and $s = 1$, then

$$(x^{(k-2)}|x^{(k-1)}|x^{(k)}) = (x^{(k-3)}|x^{(k-2)}|x^{(k-1)}) \begin{pmatrix} 0 & 0 & A \\ I & 0 & 0 \\ 0 & I & B \end{pmatrix} .$$

The period of the recurrence (3) is $2^n - 1$ if the characteristic polynomial

$$P(z) = \det\left(\mathcal{C} - zI\right)$$

is primitive over F_2. $P(z)$ is primitive if it is irreducible and the powers $z, z^2, z^3, \ldots, z^{2^n-1}$ are distinct mod $P(z)$. To verify this, without checking $2^n - 1$ cases, it is sufficient to show that $P(z)$ is irreducible and

$$z^{(2^n-1)/p} \neq 1 \bmod P(z)$$

for each prime divisor p of $2^n - 1$: see Lidl [8] or Menezes [13, §4.5].

Suppose that

$$P(z) = \sum_{j=0}^{n} c_j z^j \,.$$

From the Cayley–Hamilton theorem, $P(\mathcal{C}) = 0$, so

$$\sum_{j=0}^{n} c_j \, \mathcal{C}^j = 0 \,.$$

It follows from (3) that

$$(x^{(j)}|x^{(j+1)}|\cdots|x^{(j+r-1)}) = (x^{(0)}|x^{(1)}|\cdots|x^{(r-1)})\mathcal{C}^j \,,$$

so

$$\sum_{j=0}^{n} c_j x^{(k+j)} = 0 \,.$$

This shows that the pseudo-random sequence $x^{(k)}$ satisfies a linear recurrence over F_2. For a good random number generator it is important that the *weight* $W(P(z))$ of the polynomial $P(z)$, that is, the number of nonzero coefficients c_j, is not too small [3, 5, 14].

3 Optimal generators

Suppose the wordlength w and a parameter $r \geq 2$ are given, so $n = rw$ is defined. We want to choose positive parameters (s, a, b, c, d) such that $s < r$

and the RNG obtained from the recurrence (2) has full period $2^n - 1$. Of the many possible choices of (s, a, b, c, d), which is best? We give a rationale for making the 'best' choice (or at least a reasonably good one, since often many choices are about equally good).

1. Each bit in $x(I + L^a)(I + R^b)$ should depend on at least two bits in x, that is each column of the matrix $(I + L^a)(I + R^b)$ should have weight (number of nonzeros) at least two. A necessary condition for this is that $a + b \leq w$. Similarly, we require that $c + d \leq w$.

2. Repeated applications of the transformation $x \leftarrow x(I + L^a)(I + R^b)$ should mix all the bits of the initial x (that is, after a large number of iterations each output bit should depend on each of the input bits). A necessary condition for this is that $\text{GCD}(a, b) = 1$. Similarly, we require that $\text{GCD}(c, d) = 1$.

3. If (s, a, b, c, d) is one set of parameters, then (s, b, a, d, c) is associated with the same characteristic polynomial. We can assume that $a \geq b$, as otherwise we could interchange $a \leftrightarrow b$, $c \leftrightarrow d$ to obtain an equivalent RNG.

4. So that the left shift parameters (a and c) are not both greater than the right shift parameters (b and d) we also assume that $c \leq d$.

5. In order that the bits in $x(I + L^a)(I + R^b)$ depend on bits as far away as possible (to both left and right) in x, we want to maximise $\min(a, b)$. Similarly, we want to maximise $\min(c, d)$. Thus, we try to maximise $\delta = \min(a, b, c, d)$.

6. As already discussed, once (a, b, c, d) are fixed, we want to choose $s < r$ so that the generator has full period $2^n - 1$.

7. Finally, in case of a tie (two or more sets of parameters satisfying the above conditions with the same value of δ), we choose the set whose characteristic polynomial has maximum weight W.

There might still be a tie, that is two sets of parameters satisfying the above conditions, with the same δ and W values. However, because the weights W are quite large (see Tables 1 and 2), this is unlikely and has not been observed.

Criteria 1 and 5 lead to a simple search strategy. From criterion 1 we see that $\delta \le w/2$, but criterion 5 is to maximise δ. Thus, we start from $\delta = \lfloor w/2 \rfloor$ and decrease δ by 1 until we find a quadruple of parameters (a, b, c, d) satisfying criteria 1–4. This involves checking $O((w/2-\delta)^4)$ possibilities since $(a, b, c, d) \in [\delta, w-\delta]^4$. We then search for s satisfying criterion 6 (this is the most time-consuming step). There are $r-1$ possibilities to check for each quadruple (a, b, c, d). If no s satisfying criterion 6 is found, we decrement δ and repeat the process. Once one satisfactory quintuple (s, a, b, c, d) has been found, we need only check other quintuples (s', a', b', c', d') with the same δ, and choose the best according to criteria 6 and 7. We need only consider s such that $\mathrm{GCD}(r, s) = 1$, since this is a necessary (but not sufficient) condition for the characteristic polynomial to be irreducible.

There might not be a solution satisfying all the conditions 1–7. The number of candidates (s, a, b, c, d) is or order rw^4, that is nw^3 since $n = rw$. The probability that a randomly chosen polynomial of degree n over F_2 is primitive is between $1/(n \log n)$ and $1/n$, apart from constant factors [8, 13]. Thus, if our characteristic polynomials behave like random polynomials of the same degree, we expect at least of order $w^3/\log n$ solutions. For $w \ge 32$ we have always been able to find a solution with $w/2 - \delta \le 9$. If w is small, there may be no solution, for example there is no solution for $w = 8$, $r = 6$.

The parameters for 'optimal' random number generators with n a power of two (up to $n = 4096$) are given in Tables 1 and 2. Parameters when n is not a power of two are available from my web site [4]. The computations were performed using Magma [1].

We do not recommend the RNGs with $n \le 128$ since they may fail the matrix-rank test in the Crush testing package [6, 14]. However, no problems

Table 1: 32-bit generators.

n	r	s	a	b	c	d	δ	W
64	2	1	17	14	12	19	12	31
128	4	3	15	14	12	17	12	55
256	8	3	18	13	14	15	13	109
512	16	1	17	15	13	14	13	185
1024	32	15	19	11	13	16	11	225
2048	64	59	19	12	14	15	12	213
4096	128	95	17	12	13	15	12	251

Table 2: 64-bit generators.

n	r	s	a	b	c	d	δ	W
128	2	1	33	31	28	29	28	65
256	4	3	37	27	29	33	27	127
512	8	1	37	26	29	34	26	231
1024	16	7	34	29	25	31	25	439
2048	32	1	35	27	26	37	26	745
4096	64	53	33	26	27	29	26	961

have been observed while testing the RNGs with $n \geq 256$.

4 Problems and improvements

The **`xorgens`** class of RNGs are easy to implement since only simple operations (left and right shifts and xors) on full words are required. Unlike RNGs based on primitive trinomials, their characteristic polynomials have high weight (see column 'W' of Tables 1 and 2). Provided $n \geq 256$, they appear to pass all common empirical tests for randomness [6, 10, 14].

However, the **`xorgens`** class, like Marsaglia's xorshift class, does have an obvious theoretical weakness. For $x \in F_2^{1\times w}$, define $\|x\|$ to be the *Hamming weight* of x, that is the number of nonzero components of x. Then $\|x-y\|$ is the usual *Hamming distance* between vectors x and y. For random vectors $x \in F_2^{1\times w}$, $\|x\|$ has a binomial distribution with mean $w/2$ and variance $w/4$.

Because the matrices $(I+L^a)$ and $(I+R^b)$ are sparse, they map vectors with low Hamming weight into vectors with low Hamming weight, in fact $\|x(I+L^a)\| \le 2\|x\|$, $\|x(I+R^b)\| \le 2\|x\|$, and consequently

$$\|x(I+L^a)(I+R^b)\| \le 4\|x\| .$$

It follows that a sequence $(x^{(k)})$ generated using the recurrence (2) satisfies

$$\|x^{(k)}\| \le 4\left(\|x^{(k-r)}\| + \|x^{(k-s)}\|\right) .$$

Thus, the occurrence of a vector $x^{(k)}$ with low Hamming weight is correlated with the occurrence of low Hamming weights further back in the sequence (with lags r and s). A statistical test could be devised to detect this behaviour in a sufficiently large sample. It is a more serious problem for the 32-bit generators than for the 64-bit generators, since the probability that a w-bit vector x has Hamming weight $\|x\| \le w/8$ is 1.0×10^{-5} for $w=32$, but only 2.8×10^{-10} for $w=64$.

One solution, recommended by Panneton and L'Ecuyer [14], is to include more left and right shifts in the recurrence (2). This slows the RNG down, but not by much, since most of the time is taken by loads, stores, and other overheads. Another solution, which we prefer, is to combine the output of the xorshift generator with the output of a generator in a different class, for example a *Weyl generator* which has the simple form

$$w^{(k)} = w^{(k-1)} + \omega \bmod 2^w .$$

Here '+' means integer addition (mod 2^w) and ω is some odd constant (a good choice is an odd integer close to $2^{w-1}(\sqrt{5}-1)$). The generators in our `xorgens` package return

$$w^{(k)}(I \oplus R^\gamma) + x^{(k)} \bmod 2^w$$

instead of simply $x^{(k)}$. Here $\gamma \approx w/2$ is a constant. This is better than returning $w^{(k)} + x^{(k)} \bmod 2^w$ (as was done in an earlier version of `xorgens`) because the least significant bit of $w^{(k)}$ has period 2, but all bits of $w^{(k)}(I \oplus R^\gamma)$ have a longer period (about $2^{w/2}$), and this period is relatively prime to the period $2^n - 1$ of $x^{(k)}$. Thus each bit in the output should have high linear complexity [13].

Note that addition mod 2^w is not a linear operation on vectors over F_2, so we are mixing operations in two algebraic structures. This is generally a good idea because it avoids regularities associated with linearity. For example, suppose we use one of Marsaglia's xorshift generators to initialise our state vector, and we do it three times with seeds s, s', s'' satisfying $s = s' \oplus s''$; then by linearity over F_2 our three sequences x, x', x'' satisfy $x = x' \oplus x''$, which is clearly undesirable. This problem vanishes if the xorshift RNG used for initialisation is modified by addition (mod 2^w) of a Weyl generator, as is done in the `xorgens` package.

5 Conclusions

We have shown how Marsaglia's xorshift RNGs can be generalised to give high-quality RNGs with extremely long periods (greater than 10^{1232}). The RNGs are fast and easy to implement because only word-aligned operations are used and no 'tempering' step [12] is required. We discussed a potential problem related to correlation of outputs with low Hamming weights, and showed that this can be overcome by the simple expedient of combining the output of a generalised xorshift RNG with the output of a Weyl generator. An implementation of the resulting RNG is available in a free software package `xorgens` [4].

References

[1] W. W. Bosma and J. J. Cannon. *Handbook of Magma Functions*. School of Mathematics and Statistics, University of Sydney, 1997. http://magma.maths.usyd.edu.au/magma/. C196

[2] R. P. Brent. Factorization of the tenth Fermat number. *Mathematics of Computation*, 68, 429–451, 1999. http://wwwmaths.anu.edu.au/~brent/pub/pub161.html. C190

[3] R. P. Brent, Note on Marsaglia's xorshift random number generators. *Journal of Statistical Software*, 11, 5:1–4, 2004. http://www.jstatsoft.org/. C189, C194

[4] R. P. Brent, *Some uniform and normal random number generators:* `xorgens` version 3.04, 28 June 2006. http://wwwmaths.anu.edu.au/~brent/random.html. C190, C196, C199

[5] P. L'Ecuyer. Random number generation. *Handbook of Computational Statistics* (J. E. Gentle, W. Haerdle and Y. Mori, eds.), Ch. 2. Springer–Verlag, 2004, 35–70. http://www.iro.umontreal.ca/~lecuyer/papers.html. C194

[6] P. L'Ecuyer and R. Simard. *Testu01. A software library in ANSI C for empirical testing of random number generators*. University of Montreal, Canada, 2005. http://www.iro.umontreal.ca/~simardr/testu01/tu01.html. C196, C197

[7] D. E. Knuth. *The Art of Computer Programming*, Vol. 2: Seminumerical Algorithms, third edition. Addison-Wesley, Reading, Massachusetts, 1997. http://www-cs-faculty.stanford.edu/~uno/taocp.html. C193

[8] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications*, second edition. Cambridge Univ. Press, Cambridge, 1994. C194, C196

[9] G. Marsaglia. A current view of random number generators. *Computer Science and Statistics: The Interface* (edited by L. Billard). Elsevier Science Publishers B. V., 1985, 3–10. C190

[10] G. Marsaglia, *Diehard*, 1995. http://stat.fsu.edu/~geo/. C197

[11] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8, 14:1–9, 2003. http://www.jstatsoft.org/. C189, C190, C191, C192

[12] M. Matsumoto and T. Mishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Computer Simulation*, 8, 1:3–30, 1998. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. C189, C199

[13] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, New York, 1997. http://cacr.math.uwaterloo.ca/hac/. C189, C194, C196, C199

[14] F. Panneton and P. L'Ecuyer. On the xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation*, 15, 4:346–361, 2005. http://www.iro.umontreal.ca/~lecuyer/papers.html. C189, C190, C191, C194, C196, C197, C198

[15] R. G. Swan, Factorization of polynomials over finite fields, *Pacific J. Mathematics*, 12:1099–1106, 1962. C189

[16] S. Wagstaff et al. The Cunningham Project. http://homes.cerias.purdue.edu/~ssw/cun/index.html (last modified 31 March 2006). C190

Author address

1. **Richard P. Brent**, MSI, ANU, Canberra, ACT 0200, Australia.
   mailto:xorgens@rpbrent.com