Some long-period random number generators using shifts and xors*

Richard P. Brent
MSI & RSISE, ANU
Canberra, ACT 0200
CTAC06@rpbrent.com

3 July 2006

*Presented at CTAC06, Townville, 2–5 July 2006.
 CTAC06t

Introduction

Marsaglia recently proposed a class of uniform random number generators 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 $n = rw$.

Example

Let $F_2 = \mathrm{GF}(2)$ be the finite field with two elements $\{0, 1\}$. We write $\oplus$ for addition in F_2.

Here is an example (in C) of a 32-bit xorshift generator with $r = 1$, $n = w = 32$.

If $x^{(k)} \in F_2^{1\times w}$ is a row vector, and $x^{(0)} \neq 0$, the iteration is:

$$x^{(k+1)} = x^{(k)}(I \oplus L^{13})(I \oplus R^{17})(I \oplus L^{5})\,,$$

where L is the *left-shift* matrix and $R = L^T$ is the *right-shift* matrix.

In C, where initially `x` is any nonzero 32-bit integer, this is simply:

```
x ^= x << 13;  x ^= x >> 17;  x ^= x << 5;
```

Note that `x ^= x << 13` is equivalent to `x = x ^ (x << 13)`, and `^` means $\oplus$ in C.

Generators with long period – the idea

Choose parameters $r > s > 0$. We aim for a generator with period $2^{rw} - 1$. This is often (but not always) possible.

Computer words are regarded as row vectors $x^{(k)}$ over F_2. The basic recurrence is

$$x^{(k)} = x^{(k-r)}A \oplus x^{(k-s)}B\,.$$

Here A and B are fixed matrices in $F_2^{w\times w}$, chosen so that xA "mixes" the bits in x (and similarly for xB). So that the vector-matrix products are easy to compute, A and B are products of matrices such as $I \oplus L^\alpha$ and $I \oplus R^\beta$. 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 omits the factor $I + L^c$; we include it for reasons of symmetry, to increase the number of possible choices, and to improve properties related to Hamming weight.

Initialisation

Given $x^{(0)}$, …, $x^{(r-1)}$, the recurrence uniquely defines the sequence $(x^{(k)})_{k\geq 0}$.

We need a different RNG, perhaps one of Marsaglia's original xorshift generators, to initialise $x^{(0)}$, …, $x^{(r-1)}$. Any nonzero initialisation works in theory, but it is a good idea to discard at least the first $4r$ numbers to get past any initial "non-randomness" in the sequence [Gimeno].

Connection with other RNGs

Marsaglia's xorshift RNGs are a special case of the well-known linear feedback shift register (LFSR) class of RNGs.

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) and is usually an odd prime.

For example, $n = 19937$ in the case of the "Mersenne twister". The "tempering" step which transforms the output of the Mersenne twister can be omitted in the xorshift RNGs. Thus, the xorshift RNGs are simpler and potentially faster.

Why do we need a long period ?

It is desirable for RNGs to have a very long period T. Most generators fail certain statistical tests if more than about $T^{1/2}$ random numbers are used.

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.

On a parallel machine we may want to use disjoint segments of the cycle on different processors. This can be done by starting with different seeds on each processor, if the probability that two segments overlap is negligible.

For all these reasons it is important for n to be large. Our generators have n up to 4096, which is large enough, but not so large that the generators are slowed down by cache misses on memory accesses.

The companion matrix

It is well-known that we can write the recurrence

$$x^{(k)} = x^{(k-r)}A \oplus x^{(k-s)}B\,.$$

as

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

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

$$\mathcal{C} = \begin{pmatrix} 0 & 0 & A \\ I & 0 & 0 \\ 0 & I & B \end{pmatrix}\,.$$

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

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

is *primitive* over F_2.

Testing primitivity

$P(z)$ is primitive if it is irreducible (has no nontrivial factors) 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 (straightforward) *and*

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

for each prime divisor p of $2^n - 1$. This is easy using Magma, Maple or a similar package provided we know the prime factors of $2^n - 1$.

A use for Fermat factors

Marsaglia's original proposal discusses mainly the case $n \leq 64$, but an extension to larger n is suggested, We have implemented a generalisation `xorgens` with $n \leq 4096$, in particular we can choose any power of two $n = 2^k$ for $6 \leq k \leq 12$.

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. 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 (but not the complete factorisation of F_{12}).

The Cayley-Hamilton theorem

Suppose that

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

From the Cayley-Hamilton theorem,

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

Since

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

it follows that

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

Thus the pseudo-random sequence $x^{(k)}$ satisfies a linear recurrence over F_2. Any $n+1$ consecutive $x^{(k)}$ are linearly dependent over F_2. For a good random number generator it is important that n is large and also that the *weight* $W(P(z))$ of the polynomial $P(z)$, i.e. the number of nonzero coefficients c_j, is not too small.

"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 has 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).

Criteria

1. Each bit in $x(I \oplus L^a)(I \oplus R^b)$ should depend on at least two bits in x, that is each column of the matrix $(I \oplus L^a)(I \oplus R^b)$ 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 $x \leftarrow x(I \oplus L^a)(I \oplus R^b)$ 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 $\mathrm{GCD}(a, b) = 1$.

Similarly, we require that $\mathrm{GCD}(c, d) = 1$.

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. Thus, we may as well assume that $a \geq b$.

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 $c \leq d$.

Criteria continued

4. In order that the bits in $x(I \oplus L^a)(I \oplus R^b)$ 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

$$\delta = \min(a, b, c, d)\,.$$

5. Once (a, b, c, d) are fixed, we want to choose s so that the generator has period $2^n - 1$.

6. 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, this is unlikely and has not been observed.

Search for good RNGs

Criteria 1 and 4 lead to a simple search strategy. From criterion 1 ($a + b \leq w$) we see that

$$\delta = \min(a, b, c, d) \leq \min(a, b) \leq w/2\,,$$

but criterion 4 is to maximise δ.

We start from $\delta = w/2$ and decrease δ by 1 until we find a quadruple of parameters (a, b, c, d) satisfying criteria 1–3. 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 5 (this is the most time-consuming step).

There are $r - 1$ candidate s for each quadruple (a, b, c, d). If no s 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 5 and 6.

Are there any solutions?

There might not be a solution satisfying all the criteria 1–6.

The number of candidates (s, a, b, c, d) is $O(rw^4)$, that is $O(nw^3)$ since $n = rw$.

The probability that a randomly chosen polynomial of degree n over F_2 is primitive is between $1/n$ and $1/(n \log n)$, apart from constant factors.

Thus, if our characteristic polynomials behave like random polynomials of the same degree, we expect at least of order $w^3/\log n$ solutions. The probability of finding no solutions for a given $n \leq 4096$ should be very small unless w is small.

For $w \geq 32$ we have always been able to find a solution with $w/2 - \delta \leq 9$. The worst case is $w = 64$, $r = 30$, $\delta = 23$, $w/2 - \delta = 9$.

If w is very small, there may be no solution. For example, there is no solution for $w = 8$, $r = 6$.

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

Results

The parameters for "optimal" random number generators with n a power of two (up to $n = 4096$) are given in Tables 1–2. Parameters when n is not a power of two are available from my web site (or send me an email). The computations were performed using Magma.

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

Which RNGs to use ?

We do not recommend the RNGs with $n \leq 128$ since they may fail the matrix-rank test in the Crush package. However, no problems have been observed while testing the RNGs with $n \geq 256$.

The good news

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.

Provided $n \geq 256$, they appear to pass all common empirical tests for randomness.

The bad news

The `xorgens` class, like Marsaglia's xorshift class, has 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 $||x \oplus y||$ 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 $(I \oplus L^a)$ and $(I \oplus R^b)$ are sparse, they map vectors with low Hamming weight into vectors with low Hamming weight, in fact

$$||x(I \oplus L^a)|| \le 2||x||\,, \quad ||x(I \oplus R^b)|| \le 2||x||\,,$$

and consequently

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

It follows that our sequence $(x^{(k)})$ satisfies

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

Bad news continued

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.

This is a more serious problem for the 32-bit generators than for the 64-bit generators, since the probability p that a w-bit vector x has Hamming weight $||x|| \leq w/8$ is 1.0×10^{-5} for $w = 32$, but only 2.8×10^{-10} for $w = 64$. The sample size required to detect the low Hamming weight correlation is roughly of order $1/p^2$.

The cure

One solution, recommended by Panneton and L'Ecuyer, is to include more left and right shifts in the recurrence. 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 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

$$w^{(k)} + x^{(k)} \bmod 2^w$$

instead of simply $x^{(k)}$. Note: we are implicitly converting bit-vectors into integers; this is a "free" operation if the bit-vectors are stored in computer words.

Weakness of the Weyl generator

The period of the least significant bit of the Weyl generator is 2. Thus the least significant bits of our generators satisfy a linear recurrence of order $2n + 1$ over F_2. It would be better to return

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

where $\gamma \approx w/2$. This improvement is in the latest version (3.00) of `xorgens`.

Mixing operations

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

$$s = s' \oplus s''$$

(this is quite likely, e.g. $s = 1$, $s' = 2$, $s'' = 3$). 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.

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/`.

[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`.

[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/`.

[4] R. P. Brent, *Some uniform and normal random number generators:* `xorgens` version 3.00, 22 June 2006. `http://wwwmaths.anu.edu.au/~brent/random.html`.

[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`.

[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`.

[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`.

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

[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.

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

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

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

[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/`.

[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`.

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

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