

Your continued donations keep Wikipedia running!

Mersenne twister

From Wikipedia, the free encyclopedia

The **Mersenne twister** is a pseudorandom number generator linked to CR developed in 1997 by Makoto Matsumoto (松本 眞)

and Takuji Nishimura (西村 拓士)^[1] that is based on a matrix linear recurrence over a finite binary field F_2 . It provides for fast generation of very high quality pseudorandom numbers, having been designed specifically to rectify many of the flaws found in older algorithms.

Its name derives from the fact that period length is chosen to be a Mersenne prime. There are at least two common variants of the algorithm, differing only in the size of the Mersenne primes used. The newer and more commonly used one is the Mersenne Twister MT19937, with 32-bit word length. There is also a variant with 64-bit word length, MT19937-64, which generates a different sequence.

Because of technical limitations, some web browsers may not display some **special characters** in this article.

Contents

- 1 Application
- 2 Advantages
- 3 Algorithmic detail
- 4 Pseudocode
- 5 SFMT
- 6 References
- 7 External links

Application

Unlike Blum Blum Shub, the algorithm in its native form is not suitable for cryptography. Observing a sufficient number of iterates (624 in the case of MT19937) allows one to predict all future iterates. Combining the Mersenne twister's outputs with a hash function solves this problem, but slows down generation.

Another issue is that it can take a long time to turn a non-random initial state into output that passes randomness tests, due to its size [size of what?]. A small Lagged Fibonacci generator or Linear congruential generator gets started much quicker and is usually used to seed the Mersenne Twister. If only a few numbers are required and standards aren't high it is simpler to use the seed generator. But the Mersenne Twister will still work.

For many other applications, however, the Mersenne twister is fast becoming the pseudorandom number generator of choice. Since the library is portable, freely available and quickly generates good quality

pseudorandom numbers it is rarely a bad choice.

It is designed with Monte carlo simulations and other statistical simulations in mind. Researchers primarily want good quality numbers but also benefit from its speed and portability.

Advantages

The commonly used variant of Mersenne Twister, MT19937 has the following desirable properties:

1. It was designed to have a colossal period of $2^{19937} - 1$ (the creators of the algorithm proved this property). In practice, there is little reason to use larger ones, as most applications do not require 2^{19937} unique combinations (in decimal, 2^{19937} is approximately $4.315425 \times 10^{6001}$).
2. It has a very high order of dimensional equidistribution (see linear congruential generator). Note that this means, by default, that there is negligible[elaborate] serial correlation between successive values in the output sequence.
3. It is faster than all but the most statistically unsound generators.
4. It passes numerous tests for statistical randomness, including the stringent Diehard tests.

Algorithmic detail

The Mersenne Twister algorithm is a twisted generalised feedback shift register^[2] (twisted GFSR, or TGFSR) of rational normal form (TGFSR(R)), with state bit reflection and tempering. It is characterized by the following quantities:

- w : word size (in number of bits)
- n : degree of recurrence
- m : middle word, or the number of parallel sequences, $1 \leq m \leq n$
- r : separation point of one word, or the number of bits of the lower bitmask, $0 \leq r \leq w - 1$
- a : coefficients of the rational normal form twist matrix
- b, c : TGFSR(R) tempering bitmasks
- s, t : TGFSR(R) tempering bit shifts
- u, l : additional Mersenne Twister tempering bit shifts

with the restriction that $2^{nw} - r - 1$ is a Mersenne prime. This choice simplifies the primitivity test and k -distribution test that are needed in the parameter search.

For a word x with w bit width, it is expressed as the recurrence relation

$$x_{k+n} := x_{k+m} \oplus (x_k^u \mid x_{k+1}^l)A \quad k = 0, 1, \dots$$

with \mid as the bitwise or and \oplus as the bitwise exclusive or (XOR), x^u, x^l being x with upper and lower bitmasks applied. The twist transformation A is defined in rational normal form

$$A = R = \begin{pmatrix} \mathbf{0} & I_{n-1} \\ a_n & (a_{n-1}, \dots, a_0) \end{pmatrix}$$

with I_{n-1} as the $(n-1) \times (n-1)$ identity matrix (and in contrast to normal matrix multiplication, bitwise XOR replaces addition). The rational normal form has the benefit that it can be efficiently expressed as

$$\mathbf{x}A = \begin{cases} \mathbf{x} \gg 1 & x_0 = 0 \\ (\mathbf{x} \gg 1) \oplus \mathbf{a} & x_0 = 1 \end{cases}$$

where

$$\mathbf{x} := (x_k^u \mid x_{k+1}^l) \quad k = 0, 1, \dots$$

In order to achieve the $2^{nw-r} - 1$ theoretical upper limit of the period in a TGFSR, $\phi_B(t)$ must be a primitive polynomial, $\phi_B(t)$ being the characteristic polynomial of

$$B = \begin{pmatrix} 0 & I_w & \cdots & 0 & 0 \\ \vdots & & & & \\ I_w & \vdots & \ddots & \vdots & \vdots \\ \vdots & & & & \\ 0 & 0 & \cdots & I_w & 0 \\ 0 & 0 & \cdots & 0 & I_{w-r} \\ S & 0 & \cdots & 0 & 0 \end{pmatrix} \leftarrow m\text{-th row}$$

$$S = \begin{pmatrix} \mathbf{0} & I_r \\ I_{w-r} & \mathbf{0} \end{pmatrix} A$$

The twist transformation improves the classical GFSR with the following key properties:

- Period reaches the theoretical upper limit $2^{nw-r} - 1$ (except if initialized with 0)
- Equidistribution in n dimensions (e.g. linear congruential generators can at best manage reasonable distribution in 5 dimensions)

As like TGFSR(R), the Mersenne Twister is cascaded with a tempering transform to compensate for the reduced dimensionality of equidistribution (because of the choice of A being in the rational normal form), which is equivalent to the transformation $A = R \rightarrow A = T^{-1}RT$, T invertible. The tempering is defined in the case of Mersenne Twister as

$$\mathbf{y} := \mathbf{x} \oplus (\mathbf{x} \gg u)$$

$$y := y \oplus ((y \ll s) \& b)$$

$$y := y \oplus ((y \ll t) \& c)$$

$$z := y \oplus (y \gg l)$$

with \ll , \gg as the bitwise left and right shifts, and $\&$ as the bitwise and. The first and last transforms are added in order to improve lower bit equidistribution. From the property of TGFSR, $s + t \geq \lfloor w/2 \rfloor - 1$ is required to reach the upper bound of equidistribution for the upper bits.

The coefficients for MT19937 are:

- $(w, n, m, r) = (32, 624, 397, 31)$
- $a = 9908B0DF_{16}$
- $u = 11$
- $(s, b) = (7, 9D2C5680_{16})$
- $(t, c) = (15, EFC60000_{16})$
- $l = 18$

Pseudocode

The following generates uniformly 32 bit integers in the range $[0, 2^{32} - 1]$ with the MT19937 algorithm:

```

// Create a length 624 array to store the state of the generator
var int[0..623] MT
var int y
// Initialise the generator from a seed
function initialiseGenerator ( 32-bit int seed ) {
    MT[0] := seed
    for i from 1 to 623 { // loop over each other element
        MT[i] := last_32bits_of((69069 * MT[i-1]) + 1) // 69069 == 0x10dcd
    }
}

// Generate an array of 624 untempered numbers
function generateNumbers() {
    for i from 0 to 623 {
        y := 32nd_bit_of(MT[i]) + last_31bits_of(MT[(i+1)%624])
        if y even {
            MT[i] := MT[(i + 397) % 624] bitwise_xor (right_shift_by_1_bit(y))
        } else if y odd {
            MT[i] := MT[(i + 397) % 624] bitwise_xor (right_shift_by_1_bit(y)) bitwise_xor
        }
    }
}

// Extract a tempered pseudorandom number based on the i-th value
// generateNumbers() will have to be called again once the array of 624 numbers is exhausted
function extractNumber(int i) {
    y := MT[i]
    y := y bitwise_xor (right_shift_by_11_bits(y))
    y := y bitwise_xor (left_shift_by_7_bits(y) bitwise_and (2636928640)) // 0x9d2c5680
    y := y bitwise_xor (left_shift_by_15_bits(y) bitwise_and (4022730752)) // 0xefc60000
    y := y bitwise_xor (right_shift_by_18_bits(y))
}

```

```
    return y
}
```

SFMT

SIMD-oriented Fast Mersenne Twister (SFMT). SFMT is a variant of Mersenne Twister, introduced in 2006^[3], designed to be fast when it runs on 128-bit SIMD.

- It is roughly twice as fast as Mersenne Twister.^[4]
- It has a better equidistribution property of v-bit accuracy than MT but worse than WELL.
- It has quicker recovery from zero-excess initial state than MT, but slower than WELL.
- It supports various periods from $2^{607}-1$ to $2^{216091}-1$.

Intel SSE2 and PowerPC AltiVec are supported by SFMT.

References

1. ^ M. Matsumoto & T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Trans. Model. Comput. Simul.* **8**, 3 (1998). (<http://doi.acm.org/10.1145/272991.272995>)
2. ^ M. Matsumoto & Y. Kurita, "Twisted GFSR generators", *ACM Trans. Model. Comput. Simul.* **2**, 179 (1992); (<http://doi.acm.org/10.1145/146382.146383>) **4**, 254 (1994). (<http://doi.acm.org/10.1145/189443.189445>)
3. ^ <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html>
4. ^ <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/speed.html>

External links

- The academic paper for MT, and related articles by Makoto Matsumoto (<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/earticles.html>)
- Mersenne Twister home page, with codes in C, Fortran, Java, Lisp and some other languages (<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>)
- The GNU Scientific Library (GSL), containing an implementation of the Mersenne Twister (<http://www.gnu.org/software/gsl/>)
- Fast implementations of the Mersenne Twister in C and C++ (<http://www.cs.hmc.edu/~geoff/mtwist.html>)
- An implementation of the Mersenne Twister algorithm in C++ (<http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html>)
- Two good implementations of Mersenne Twister in Java, including the fastest known Java implementation (<http://cs.gmu.edu/~sean/research/>)
- Java implementations of both the 32 and 64 bit versions (<http://modp.com/release/javarng/>)
- Implementation of Mersenne Twister (<http://workshop.evolutionzone.com/2006/05/16/random-number-generator-mersenne-twister/>) for Processing (<http://processing.org/>)
- Implementation of Mersenne Twister for REALbasic (requires REALbasic 2006r1 or greater) (<http://www.aaronballman.com/programming/REALbasic/Rand.php>)
- Implementation of Mersenne Twister for Lisp (<http://lisp-p.org/jmt/>)

- Implementation of Mersenne Twister in Euphoria (<http://www.rapideuphoria.com/mt.zip>)
- Implementation of Mersenne Twister for C# (<http://www.c-sharpcorner.com/Code/2003/April/MersenneTwisterAlgo.asp>)
- Implementation of Mersenne Twister for Ada (<http://adrianhoe.com/adrianhoe/projects/adamt19937/>)
- Implementation of Mersenne Twister for Fortran 95 (http://www.coyotegulch.com/products/libcoyotl/twisted_road/index.html)
- Implementation of Mersenne Twister for Mathematica (http://modp.com/release/mma_mersenne_twister/)
- Implementation of Mersenne Twister for MATLAB (<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=6614&objectType=file>)
- Implementation of Mersenne Twister for Clean (http://clean.cs.ru.nl/Download/Download_Libraries/mt/body_mt.html)
- High-speed Implementation of Mersenne Twister (<http://herbert.gandraxa.com/herbert/rng.asp>) in Linoleum (a cross-platform Assembler), by Herbert Glarner
- Implementation of Mersenne Twister as an add-in for Microsoft Excel (<http://www.numtech.com/NtRand/>)
- CPAN module implementing the Mersenne Twister for use with Perl (<http://search.cpan.org/search?module=Math%3A%3ARandom%3A%3AMT%3A%3AAuto>)
- Implementation of Mersenne Twister for Haskell (<http://www.augustsson.net/Darcs/MT/>)
- Implementation of Mersenne Twister for Standard ML (<http://mlton.org/cgi-bin/viewsvn.cgi/mltonlib/trunk/org/mlton/ville/mersenne-twister/unstable/>)
- It also is implemented in gLib and the standard libraries of at least PHP, Python and Ruby.
- SIMD-oriented Fast Mersenne Twister (SFMT) (<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html>)
- C++ class implementing Mersenne Twister and SFMT (<http://charles.karney.info/random>)
- Flash Actionscript implementation of Mersenne Twister as a Class file (<http://www.deviantart.com/deviation/53259335/>)

Retrieved from "http://en.wikipedia.org/wiki/Mersenne_twister"

Categories: Articles to be expanded since June 2007 | All articles to be expanded | Pseudorandom number generators | Articles with example pseudocode

-
- This page was last modified 16:10, 25 August 2007.
 - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.)
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a US-registered 501(c)(3) tax-deductible nonprofit charity.