COMBINING THE MERSENNE TWISTER AND THE XORGENS DESIGNS

M. VAN DE VEL

Abstract. We combine the design of two random number generators, Mersenne Twister and Xorgens, to obtain a new class of generators with heavy-weight characteristic polynomials (exceeded only by the well generators) and high speed (comparable with the originals). Tables with parameter combinations are included for state sizes ranging from 521 to 44497 bits and each of the word lengths 32, 64, 128. These generators passed all tests of the TestU01-package for each 32-bit integer part and each 64-bit derived real part of the output. We determine dimension gaps for 32-bit words, neglecting the non-linear tempering, and compare with an alternative experimental linear tempering.

Categories and Subject Descriptors: G.4 [Mathematical Software]: Random number generator.
General terms: Algorithms.
Additional Key Words and Phrases: Characteristic Polynomial, Dimension Gap, Mersenne Twister, Xorgens.

1. Introduction

The Mersenne Twister (mt), introduced by (Matsumoto & Nishimura, 1998), is one of the most popular random number generators (RNGs) of the moment. The bit size of the state space is a (prime) exponent $p$ of a Mersenne prime $2^p - 1$. As $p$ is not an integer multiple of the word size, the state space contains an incomplete word which requires a state transition “twisting” around the memory gap. The period of this generator is the associated Mersenne prime.

The original Mersenne Twisters had one of the sizes 11213, 19937, each with a few variants. The newer (and faster) SIMD oriented versions (Saito & Matsumoto, 2008) also exist in larger sizes. More recently, mt-designs were presented by (Saito & Matsumoto, 2013) taking profit from parallel processing with graphic processors (GPUs).

The Xorgens (xg) class represents a rather different design of random number generators due to (Brent, 2007). Building on earlier work of (Marsaglia, 2003), Brent defines the core activity of xg in terms of bit shifts and exclusive or (xor). The state has $2^t$ bits ($t \geq 8$). The xg class also includes generators of size 4224, 4480, which are not a power of 2. However, the size of an xg is always an integer multiple of the word size.

Except for these two, the maximal possible period, $2^{2t} - 1$, is a product of successive Fermat numbers $F_i, i = 1, \cdots, t - 1$. A proof that this period is attained requires knowledge of its factorization, which is available only for
$F_i, i \leq 11$, (Brent, 1989). (Nandapalan et al., 2012) describe GPU processing with the Xorgens design.

Due to the involvement of Fermat number factorizations, only “small” sizes are available for xg generators. An alternative design with Mersenne primes would give access to different and larger sizes and—as it imposes a non-integral word size of the state— one might expect variant behaviour through the “twist” of MT. This lead us to a sequence of well-behaved generators with heavy-weight characteristic polynomials (CPs). In fact, the only RNGs known to us to have even heavier CPs belong to the well class ((Panneton et al., 2006)). Our new RNGs have been labeled Mersenne Xorgens (mxg) to honour our two sources of inspiration.

Table 1 presents speed measurements with a 2.40GHz Intel core. We used mt19937ar, which is a 2002 revision of the original MT by T. Nishimura and M. Matsumoto, and more recent memt19937-II by Shin Harase.

We refer to (Van de Vel, 2014) for our C code used for speed measuring of 64-bit mxgs. Adaptation of the code to 32-bit words is straightforward. The code for 128 bits requires a special header file (emmintrin.h) and some definitions of word operations like (arbitrary) shift and xorshift.

Tables with good parameters are presented in section 2. The selected generators passed all Smallcrush, Crush, and BigCrush tests in the package TestU01 ([‘Ecuyer & Simard, 2007]), applied to each 32-bit part.

| generator | 32 bit  | 64 bit  | generator | 32 bit  | 64 bit  | 128 bit  |
|-----------|---------|---------|-----------|---------|---------|----------|
| xg512     | 3.21s   | 2.47s   | mxg521    | 3.76s   | 2.74s   | 6.13s    |
| xg1024    | 3.16s   | 2.97s   | mxg1279   | 3.56s   | 3.65s   | 6.70s    |
| xg2048    | 1.56s   | 3.31s   | mxg2203   | 2.09s   | 3.49s   | 7.34s    |
| xg4096    | 3.25s   | 2.09s   | mxg3217   | 3.87s   | 3.51s   | 6.61s    |
| xg4224    | 1.56s   | 2.18s   | mxg4253   | 2.49s   | 2.57s   | 6.71s    |
| xg4480    | 1.67s   | 2.21s   | mxg4423   | 2.68s   | 3.59s   | 7.54s    |
| mt11213A  | 2.99s   | -       | mxg11213  | 3.46s   | 3.80s   | 7.16s    |
| mt19937ar | 2.96s   | 2.94s   | mxg19937  | 3.43s   | 3.46s   | 6.56s    |
| memt19937-II | 5.62s | -       | mxg21701  | -       | 3.73s   | 7.00s    |
| well19937 | 7.42s   | -       | mxg23207  | -       | 3.45s   | 6.60s    |
| well44497 | 12.98s  | -       | mxg4497   | -       | 3.70s   | 7.19s    |

Table 1. Speed measurements: producing $10^9$ integers

Well generators have an additional property of maximal equidistribution (ME), and so does memt19937-II. To measure deviation from ME, we computed the (total) dimension gaps $\Delta$ ((Panneton et al., 2006)) for MXG (dropping the non-linear Weyl tempering). We treated each 32-bit word separately. XG does slightly better for smaller sizes and is somewhat worse for larger sizes. On average, our scores are comparable with MTs. See section 2.

However, for a fair judgement we repeated our computations with an additional linear tempering (two left and right xorshifts), applied equally to each MXG of any word size. The improvements on dimension gaps ranged between “none” and “spectacular”: four tempered 64-bit mxgs deserve to be called near-ME generators (with all 32-bit words having $\Delta < 10$. There
is also one XG like this. This experiment suggests that simple, fashioned linear temperings of MXG may produce other fast and near-ME generators.

2. MXG Generators

2.1. Random number generators. Let \( w \) denote a convenient word length (32, 64, 128) and let \( n \) denote the number of \( w \)-bit words in the state of a generator. An eventual partial word is counted for one word and its bits are thought of as being most significant (upper bits); its lower bits are neglected. The words of the state constitute an \( n \)-vector

\[ x := (x_{n-1}, \ldots, x_1, x_0), \]

and its evolution is conceived as a continuation of the word sequence with words \( x_n, x_{n+1}, \ldots \) and with a sliding window of \( n \) consecutive words. Each step is perceived as a state transition \( x \mapsto f(x) \). In practice, one keeps running through the same memory positions by incrementing the word index modulo \( n \). The output is given by a word-valued operator \( x \mapsto o(x) \). One action cycle of the generator involves application of \( f \), then of \( o \).

It is assumed here that at least \( f \) is linear. Hence we deal with a linear recurrence having an associated characteristic polynomial (CP). For a state with \( p \) bits, the maximal possible period is \( 2^p - 1 \) and is attained iff the CP is primitive, \cite{(Lidl & Niederreiter, 1994)}.

2.2. The Mersenne Twister. We have a state space with \( n \cdot w - r \) bits, where \( 0 < r < w \). Only the upper \( w - r \) bits of \( x_0 \) are used. The integer \( m < n \) (the step) determines how far to look ahead in the state space for a transition. Consider the upper and lower masks

\[ u_{w-r} := 1 \ldots 10 \ldots 0 \quad l_r := 0 \ldots 0 \underbrace{1 \ldots 1}_{r}. \]

The \( k \)-th state transformation of a Mersenne twister is given by

\[ x_{k+n} := x_{k+m} \oplus (x_k \cdot u_{w-r} \oplus x_{k+1} \cdot l_r)M, \quad (k \geq 0). \]

The tokens ‘\( \oplus \)’ and ‘\( . \)' represent bit-wise addition (“xor”), resp., multiplication (“and”). The recurrence equation contains a \( w \times w \) bit matrix

\[ M := \begin{pmatrix}
0 & 1 & 0 & \ldots & 0 \\
0 & 0 & 1 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & 1 \\
a_{w-1} & a_{w-2} & \ldots & \ldots & a_0
\end{pmatrix} \]

which does a right shift of a bit word \((b_{w-1}, \ldots, b_0)\) by 1 and adds the word formed by the bottom row if \( b_0 \neq 0 \). The standard MT recovers rather slowly from a near-zero state. See \cite{(Panneton et al., 2006)} for a description of how the WELL design avoids this phenomenon (escaping zero-roland) because of a high CP weight. The actual output of MT is a word tempered with right- and left shifted bit-wise additions (xorshifts). See \cite{(Matsumoto & Nishimura, 1998)} for a precise description of the tempering.
2.3. The Xorgens class. XG generators (Brent, 2007) have \( n \) entire words, a step parameter \( m \) (as with MT; Brent uses \( s := n−m \)), and four parameters \( a, b, c, d \) controlling left and right shifts. The \( k \)-th state transformation is
\[
x_{k+n} := x_{k+m}(I \oplus C)(I \oplus D) \oplus x_k(I \oplus A)(I \oplus B), \quad (k \geq 0).
\]
The symbol \( I \) denotes the \( w \times w \) unit matrix. In addition, \( A, C \) are matrices reflecting left shifts by the respective amounts \( a, c \), whereas \( B, D \) represent right shifts by the respective amounts \( b, d \).

Brent prefers tempering with a Weyl generator, whence the output becomes non-linear over the bit field. This generator starts from any integer \( w_0 \) that fits with the word length \( w \). At the \( k \)-th step the number \( w_k \) is generated as \( w_{k-1} \) increased modulo \( 2^w \) with a Weyl constant; a good choice is the odd integer approximation to \( 2^w \cdot (3 − \sqrt{5})/2 \). The actual output of XG is a numerical addition modulo \( 2^w \),
\[
x_{k+n} + (w_k(I \oplus W)),
\]
where the matrix \( W \) represents a right shift over a fixed amount (the Weyl shift). This constant is chosen as 16 (32-bit XG) and 27 (64-bit XG).

2.4. MXG generators. Random number generators of the MXG class apply the Xorgens recurrence to a state with the bit size of a Mersenne prime exponent. Let \( r \) be the shortage of bits to have entire words. We borrow from MT the idea of joining the upper \( w−r \) bits of a word with the lower \( r \) bits of the next word. With the above notations, the recurrence becomes
\[
x_{k+n} := x_{k+m}(I \oplus C)(I \oplus D) \oplus (x_k \cdot u^{w−r} \oplus x_{k+1} \cdot l')(I \oplus A)(I \oplus B).
\]
We chose for tempering with the simple Weyl generator to keep to our principle: MXG is simply XG with the twist of MT. For the same reason, we also copied the reliable XG initialisation to MXG.

2.5. The search process. Having determined the generic transformations that run an MXG, our next task is to assign parameters \( n, m, a, b, c, d \), depending on the involved Mersenne prime, to obtain a maximal period. Brent’s priorities to decide among multiple solutions can be followed only to a certain extent.

We consider pairs \( (a, b) \) and \( (c, d) \) which are relatively prime, satisfy opposite strict inequalities, and obey \( a + b \leq w \) and \( c + d \leq w \), where \( w \) denotes the word length. Left-right symmetry of pairs (as in XG) is ruined by the “twist” design. Our search process counts down on the parameter \( \delta := \min(a, b, c, d) \) from \( w/2 − 1 \) downto 6 (for 32-bit designs) respectively, 17 (for 64-bit designs) and 40 (for 128-bit designs), giving primary attention to high \( \delta \).

The list of admissible steps \( m \) reveals another difference with XG. Contrary to an observation of Brent for XG, \( m \) (actually, \( s := n−m \)) need not be relatively prime with the number of words, \( n \), to obtain a primitive characteristic polynomial of an MXG. For instance, Table 3 below lists a successful example of size 2281, where \( n = 36 \) and \( m = 32 \). As the involved words of a transition should not be neighbors (which would interfere with
the “Mersenne twist”) we let \( m \) range in \( 2 \ldots n - 1 \). The resulting search space is roughly ten times the size of Brent’s at comparable xg sizes.

The probability of a polynomial with a large prime degree \( p \) being primitive is nearly exactly \( 1/p \), \([\text{Lidl & Niederreiter, 1994}]\). Hence most of the search area is wasteland which should be crossed as fast as possible. Unfortunately, the efficient inversive-decimation method of \([\text{Matsumoto & Nishimura, 1998}]\) to decide on primitivity of the CP with \( O(p^2) \) operations seems to be more complex for mxg. Refer to 2.7.

Contrasting with Brent’s view, we hold that the weight of the characteristic polynomial should have more importance than just a final tie solver. Although a high CP weight (optimally, near fifty percent) does not guarantee a good generator, it does measure the intensity of mixing state bits. Moreover, a high weight facilitates recovery from near-zero states and divergence of nearly identical states. Therefore, it is a reasonable search directive. We aim at CP weights ranging between \( \frac{1}{3} \) and \( \frac{2}{3} \) of the state bitsize.

2.6. Description of the results. We searched (a fair portion of) the parameter space for suitable parameter combinations with a heavy-weight primitive CP. The corresponding generators survived all tests in the batteries Small Crush (10), Crush (96), and Big Crush (106) of TestU01, \([\text{Cruch & Simard, 2007}]\). As this package requires an RNG to produce 32-bit integers (if not reals), we considered each generator separately with each 32-bit word of its output. In case of 128 bits, we tested real numbers derived from the lower- and upper 64 bits separately. Rare failures always showed a so-called “p-value” in between \( 10^{-4} \) and \( 10^{-3} \) and did not reappear when the misbehaving test was performed three more times.

We also computed the total dimension gap \( \Delta \) —a global measure for the deviation from maximal equidistribution, \([\text{Panneton et al., 2006}]\)— for each 32-bit part of our generators and of all xgs, dropping the standard non-linear Weyl tempering. Computing on 32-bit parts is not unusual, cf. \([\text{Saito & Matsumoto, 2008}]\). For a fair judgement, we also computed \( \Delta \) after applying a fixed linear tempering,

\[
\begin{align*}
y & \gets y \gg 1; \quad y \gets y << 11; \\
y & \gets y \gg 7; \quad y \gets y << 3;
\end{align*}
\]

of an untempered word \( y \).

2.6.1. The 32-bit case. We found generators with a primitive CP for Mersenne primes \( M_{13} \ldots M_{24} \). See table 2. Only at \( M_{13}, M_{14}, M_{15}, \) and \( M_{18} \) our criterion on high CP weights is met, yet the weights are much higher than those of nearby 32-bit xg sizes (see \([\text{Brent, 2007}]\) or \( \text{http://maths-people.anu.edu.au/~brent/ftp/random/xortable.txt} \)).

\[
\begin{align*}
xG512: & \quad 185; \quad xG1024: \quad 225; \quad xG2048: \quad 213; \\
xG4096: & \quad 251; \quad xG4224: \quad 243; \quad xG4480: \quad 251.
\end{align*}
\]

For 32-bit mt of size 11213 we found \( \Delta = 4087 \) whereas the standard 32-bit mt of size 19937 has \( \Delta = 6750 \) \([\text{Saito & Matsumoto, 2008}]\). These values are more than twice our tempered mxg values. For 32-bit xg generators we found the following \( \Delta \)-values (untempered,tempered):

\[
\begin{align*}
xG512: & \quad (142, 92); \quad xG1024: \quad (141, 100); \quad xG2048: \quad (465, 335); \\
xG4096: & \quad (845, 454); \quad xG4224: \quad (1838, 622); \quad xG4480: \quad (2038, 590).
\end{align*}
\]
### Table 2. MXG designs (32-bit words)

| M | bits | n | r | a | b | c | d | weight | ∆ | untemp | temp |
|---|------|---|---|---|---|---|---|--------|---|--------|------|
| M13 | 521 | 17 | 23 | 10 | 11 | 15 | 14 | 11 | 261 | 150 | 38 |
| M14 | 607 | 19 | 1 | 3 | 17 | 13 | 7 | 22 | 303 | 167 | 76 |
| M15 | 1279 | 40 | 1 | 26 | 13 | 10 | 9 | 23 | 513 | 360 | 117 |
| M16 | 2203 | 69 | 5 | 16 | 16 | 13 | 10 | 11 | 855 | 433 | 206 |
| M17 | 2281 | 72 | 23 | 65 | 13 | 18 | 15 | 14 | 923 | 983 | 298 |
| M18 | 3217 | 101 | 15 | 95 | 19 | 13 | 15 | 16 | 1203 | 1040 | 520 |
| M19 | 4253 | 133 | 3 | 31 | 11 | 8 | 9 | 16 | 1045 | 1497 | 418 |
| M20 | 4423 | 139 | 25 | 79 | 15 | 14 | 11 | 18 | 1383 | 1157 | 363 |
| M21 | 9689 | 302 | 7 | 295 | 14 | 13 | 13 | 18 | 1647 | 4575 | 2077 |
| M22 | 9941 | 311 | 11 | 17 | 13 | 14 | 17 | 14 | 1765 | 3719 | 1799 |
| M23 | 11213 | 351 | 19 | 330 | 17 | 13 | 15 | 17 | 2021 | 5363 | 1883 |
| M24 | 19937 | 621 | 31 | 319 | 13 | 19 | 16 | 15 | 581 | 8047 | 2488 |

### Table 3. MXG designs (64-bit words)

| M | bits | n | r | a | b | c | d | weight | ∆ | word 1 | word 0 | untemp | temp |
|---|------|---|---|---|---|---|---|--------|---|--------|--------|--------|------|
| M13 | 521 | 9 | 55 | 4 | 32 | 27 | 28 | 33 | 261 | 146 | 2 | 184 | 5 |
| M14 | 607 | 10 | 33 | 6 | 31 | 26 | 27 | 34 | 303 | 126 | 0 | 133 | 1 |
| M15 | 1279 | 20 | 1 | 6 | 27 | 32 | 33 | 29 | 639 | 320 | 4 | 685 | 10 |
| M16 | 2203 | 35 | 37 | 23 | 23 | 29 | 25 | 22 | 1089 | 335 | 2 | 821 | 8 |
| M17 | 2281 | 36 | 23 | 23 | 25 | 19 | 19 | 23 | 1121 | 305 | 3 | 314 | 11 |
| M18 | 3217 | 51 | 47 | 29 | 22 | 35 | 37 | 21 | 1519 | 108 | 3 | 225 | 6 |
| M19 | 4253 | 67 | 35 | 8 | 25 | 26 | 25 | 23 | 1983 | 3648 | 80 | 3935 | 52 |
| M20 | 4423 | 70 | 57 | 62 | 31 | 28 | 23 | 34 | 2057 | 378 | 10 | 1038 | 22 |
| M21 | 9689 | 152 | 39 | 41 | 29 | 31 | 29 | 28 | 3925 | 11700 | 5403 | 12041 | 229 |
| M22 | 9941 | 156 | 43 | 93 | 27 | 34 | 31 | 28 | 4355 | 86 | 61 | 5010 | 69 |
| M23 | 11213 | 176 | 51 | 275 | 35 | 29 | 28 | 35 | 6913 | 5687 | 177 | 5863 | 154 |
| M24 | 19937 | 312 | 31 | 308 | 33 | 29 | 27 | 37 | 5765 | 5303 | 231 | 5304 | 293 |
| M25 | 44497 | 696 | 47 | 662 | 31 | 33 | 31 | 29 | 11663 | 49289 | 1462 | 49984 | 1971 |

#### 2.6.2. The 64-bit case.

The parameters given in table 3 for $M_{25}$ and $M_{27}$ have the highest CP weight found, but they do not meet our “1/3” criterion. Here are the corresponding Xorgens CP weights:

- $xg_{512}$: 231; $xg_{1024}$: 439; $xg_{2048}$: 745;
- $xg_{4096}$: 961; $xg_{4224}$: 951; $xg_{4480}$: 987.

The total equidistribution deficits of 64-bit MXG are given in the last columns, together with the values in case of a fixed linear tempering. The 64-bit mt19937 has $\Delta$(word1,word0) = (4161, 10299). As to Xorgens, the values of $\Delta$(word1,word0), untempered/tempered, are:

- $xg_{512}$: (74/7, 123/4); $xg_{1024}$: (105/4, 350/11);
- $xg_{2048}$: (209/100, 416/102); $xg_{4096}$: (666/26, 1715/50);
- $xg_{4224}$: (1202/48, 2334/70); $xg_{4480}$: (2710/634, 3351/949).
2.6.3. **The 128-bit case.** As xorshift operators involve more bits with longer words, state transitions are likely to gain complexity. Our search process confirms that both the average and maximal weight of primitive polynomials grow significantly with the wordsize. Therefore, we extrapolated the MXG-design to 128 bit, using SIMD (Single Instruction Multiple Data) to handle 128-bit words. Table 4 presents good parameters. For each target state size, we achieved our criterion on high CP weights rather easily. [Saito & Matsumoto, 2008] report that the CP weight of 128-bit sfmt19937 is 6711 and that the 32-bit well19937 has CP weight 8585. Table 5 shows the dimension gaps of 128-bit generators for each 32-bit word. For sfmt19937, \( \Delta \) equals 14089 (64-bit) or 28676 (128-bit).

| \(M_i\) | bits | n | r | m | a | b | c | d | weight |
|--------|------|---|---|---|---|---|---|---|--------|
| 13     | 521  | 5 | 119| 3 | 57 | 68 | 65 | 57 | 261    |
| 14     | 607  | 5 | 33 | 3 | 57 | 61 | 57 | 56 | 303    |
| 15     | 1279 | 10| 1  | 3 | 56 | 71 | 69 | 58 | 639    |
| 16     | 2203 | 18| 101| 7 | 59 | 60 | 64 | 61 | 1103   |
| 17     | 2281 | 18| 23 | 3 | 60 | 67 | 67 | 59 | 1145   |
| 18     | 3217 | 26| 111| 18| 69 | 59 | 61 | 66 | 1597   |
| 19     | 4253 | 34| 99 | 26| 64 | 63 | 58 | 69 | 2097   |
| 20     | 4423 | 35| 57 | 29| 61 | 60 | 59 | 63 | 2163   |
| 21     | 9689 | 76| 39 | 69| 62 | 65 | 63 | 62 | 4621   |
| 22     | 9941 | 78| 43 | 11| 61 | 62 | 59 | 58 | 4681   |
| 23     | 11213| 88| 51 | 28| 61 | 63 | 64 | 61 | 5163   |
| 24     | 19937| 156| 31 | 85| 61 | 67 | 64 | 63 | 8823   |
| 25     | 21701| 170| 59 | 133| 63 | 64 | 62 | 61 | 9785   |
| 26     | 23209| 182| 87 | 99| 64 | 63 | 61 | 67 | 9965   |
| 27     | 44497| 348| 47 | 235| 62 | 63 | 65 | 63 | 17293  |

**Table 4. MXG designs (128-bit words)**

2.7. **Conclusion.** MXG generators combine the Xorgens and Mersenne Twister designs, have heavy CPs, and pass all standard statistical tests. Replacing the nonlinear Weyl tempering by a fixed linear tempering provides several near ME generators. Fashioned tempering, especially in the range of 64-bit generators, may produce more of these. It is an open problem whether, for an MXG of bitsize \( p \), primitivity of the CP can be decided with at most \( O(p^2) \) operations as is the case with MT [Matsumoto & Nishimura, 1998].

**References**

Brent, R.P. 1989. Factorisation of the eleventh Fermat number (preliminary report). *AMS Abstracts, Vol.10*, 89T–11–73.

Brent, R.P. 2007. Some long-period random number generators using shifts and xors. *ANZIAM Journal, Vol.48 (CTAC 2006)*, C188–C202.

l’Ecuyer, P., & Simard, R. 2007. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Trans. Math. Software, 33*(4), Art. 22.
REFERENCES

| bits | <word 3> | <word 2> | <word 1> | <word 0> |
|------|---------|---------|---------|---------|
| M_{13} | 521 | 132 | 60 | 157 | 2 | 148 | 32 | 160 | 4 |
| M_{14} | 607 | 668 | 347 | 813 | 64 | 669 | 274 | 821 | 57 |
| M_{15} | 1279 | 114 | 87 | 39 | 2 | 139 | 114 | 140 | 5 |
| M_{16} | 2203 | 2038 | 477 | 2646 | 2366 | 2463 | 124 | 2647 | 2645 |
| M_{17} | 2281 | 969 | 872 | 896 | 6 | 1076 | 619 | 1089 | 78 |
| M_{18} | 3217 | 1365 | 44 | 1526 | 2 | 1511 | 5 | 1526 | 1527 |
| M_{19} | 4253 | 2456 | 2052 | 2734 | 2 | 2714 | 18 | 2733 | 2732 |
| M_{20} | 4423 | 5083 | 2577 | 5976 | 5975 | 5609 | 1806 | 5976 | 5977 |
| M_{21} | 9689 | 14527 | 10927 | 14894 | 11646 | 14872 | 7612 | 14894 | 13113 |
| M_{22} | 9941 | 10478 | 2796 | 11950 | 11950 | 11950 | 11950 | 11949 |
| M_{23} | 11213 | 14103 | 5827 | 15165 | 15165 | 15165 | 15165 | 11166 |
| M_{24} | 19937 | 21411 | 17941 | 20932 | 9612 | 21414 | 12799 | 21436 | 10106 |
| M_{25} | 21701 | 32050 | 21218 | 33362 | 33362 | 33359 | 15562 | 33362 | 33361 |
| M_{26} | 23209 | 24930 | 20222 | 24954 | 2207 | 24933 | 14905 | 24951 | 4359 |
| M_{27} | 44497 | 75356 | 38085 | 78848 | 78846 | 77439 | 27083 | 78847 | 78846 |

Table 5. 128-bit Deficit, untempered/tempered

Lidl, R., & Niederreiter, H. 1994. *Introduction to finite fields and their applications*. Cambridge University Press.

Marsaglia, G. 2003. Xorshift RNGs. *Journal of Statistical Software*, **8**, No.14, 1–6.

Matsumoto, M., & Nishimura, T. 1998. Mersenne Twister: a 623-dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, **Vol.8 No 1**, pp. 3–30.

Nandapalan, N., Brent, R.P., Murray, L.M., & Rendell, A.P. 2012. High-Performance Pseudo-Random Number Generation on Graphic Processing Units. *Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics*, **LNCS 7203**, 609–618.

Panneton, F., L’Ecuyer, P., & Matsumoto, M. 2006. Improved Long-Period Generators Based on Linear Recurrences Modulo 2. *ACM Transactions of Mathematical Software*, **Vol.32 No 1**, 1–16.

Saito, M., & Matsumoto, M. 2008. SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. *Monte Carlo and Quasi-Monte Carlo Methods 2006*, 607–622.

Saito, M., & Matsumoto, M. 2013. Variants of Mersenne Twister Suitable for Graphic Processors. *ACM Transactions on Mathematical Software*, **39 (2)** 12.

Van de Vel, M. 2014. [https://www.researchgate.net/publication/267439134_mxg_a_new_clas_of_random_number_generators_in_between_Mersenne_Twister_and_Xorgens](https://www.researchgate.net/publication/267439134_mxg_a_new_clas_of_random_number_generators_in_between_Mersenne_Twister_and_Xorgens).