Generalized Feedback Shift Register Pseudorandom Number Algorithm

T. G. LEWIS

University of Missouri at Rolla, Rolla, Missouri

AND

W. H. PAYNE

Washington State University, Pullman, Washington

ABSTRACT. The generalized feedback shift register pseudorandom number algorithm has several advantages over all other pseudorandom number generators These advantages are (1) it produces multidimensional pseudorandom numbers; (2) it has an arbitrarily long period independent of the word size of the computer on which it is implemented, (3) it is faster than other pseudorandom number generators, (4) the "same" floating-point pseudorandom number sequence is obtained on any machine, that is, the high order mantissa bits of each pseudorandom number agree on all machines—examples are given for IBM 360, Sperry-Rand-Univac 1108, Control Data 6000, and Hewlett-Packard 2100 series computers; (5) it can be coded in compiler languages (it is portable), (6) the algorithm is easily implemented in microcode and has been programmed for an Interdata computer

KEY WORDS AND PHRASES pseudorandom numbers, Lehmer, Tausworthe, feedback shift register, linear recurrence mod 2, primitive polynomial, GF(2), tests of randomness, lattice, wave properties, Fourier analysis, Kendall's algorithm

CR CATEGORIES: 3.15, 5.5, 8.1

The Lehmer congruential multiplicative pseudorandom number generator has been shown to have n-space nonuniformity [1, 5]. This shortcoming is particularly severe for small word machines. As an alternative, the feedback shift register (FSR) pseudorandom number generator (RNG) is claimed to be uniform in n-space if p-bit words are decimated into n subwords [7]. Figures 1 and 2 give examples of nonuniformity of Lehmer RNG's for 9-bit words in two- and three-space. FSR generators with primitive generating function, $X^p + X^q + 1$, small q or q near $(p-1)/2$, should be avoided because of bad runs properties [8]. However, careful selection of p, q provides satisfactory random numbers in low-dimensional space [10].

Perhaps FSR sequences offer the best prospects for n-space improvement. Kendall's algorithm is moderately fast on most machines, but the period is fixed by the word size and it is difficult to implement in multiprecision [10]. Moreover, decimation in order to gain n-space uniformity further shortens cycle length and resolution. This problem is intrinsic to periodic sequences. A cyclic sequence of m numbers, when taken in pairs, locates only m of m^2 points in a two-dimensional m by m grid. In general, $m^n - m$ grid

Authors' addresses. T. G Lewis, Computer Science Department, University of Missouri at Rolla, Rolla, MO. 65401, W. H. Payne, Computer Science Department and Computing Center, Washington State University, Pullman, WA 99163.

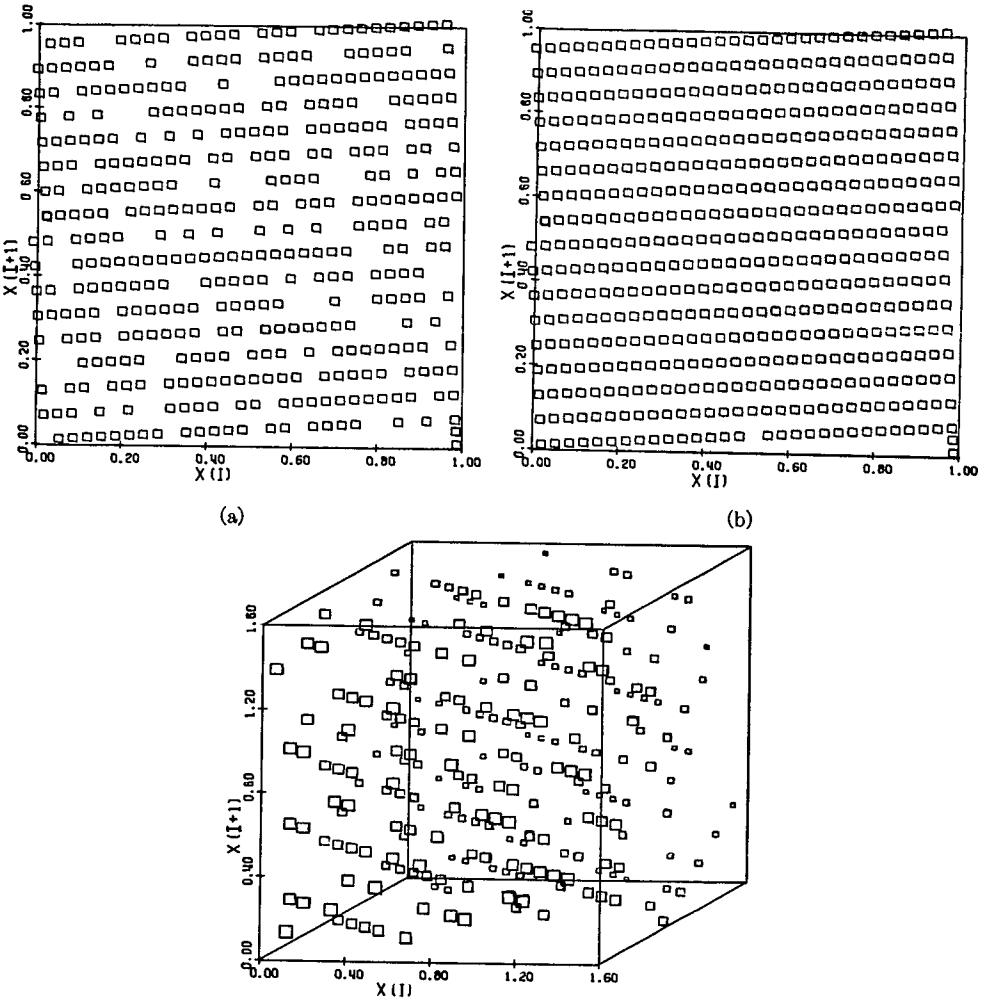


FIG 1 Continued growth of "crystals" in $X_{i+1} \equiv 17X_i - 1 \pmod{512}$, (a) 384 points in 2-space, (b) 512 points in 2-space, (c) plot in 3-space of 256 successive triplets

points will never be located in n-space by n-tuples taken from an m-periodic sequence, i.e. most cross products are missing. This sparseness in n-space underlies nonuniformity of all periodic RNG's. Apparently, what is needed is an RNG which allows repeated numbers within a full period sequence. Such repeated numbers could fill in m^n points, for some n.

The purpose of this paper is to present a new, completely general FSR algorithm for generating arbitrarily long sequences of random numbers possessing desirable n-space properties on any word size machine, and to demonstrate the feasibility of microprogramming a single instruction with high speed/cost ratio, which produces a random number each time it is executed.

1. *Generalized Feedback Shift Register Algorithm (GFSR)*

Define $\oplus$ to be the EXCLUSIVE-OR operator which is equivalent to addition modulo 2. Kendall's algorithm is used to select successive n-tuples from the basic sequence $\{a_l\}$, where $a_k = a_{k-p+q} \oplus a_{k-p}$, $k = p, p+1, \cdots$, given $a_{p-1}, \cdots, a_0$ and feedback shift reg-

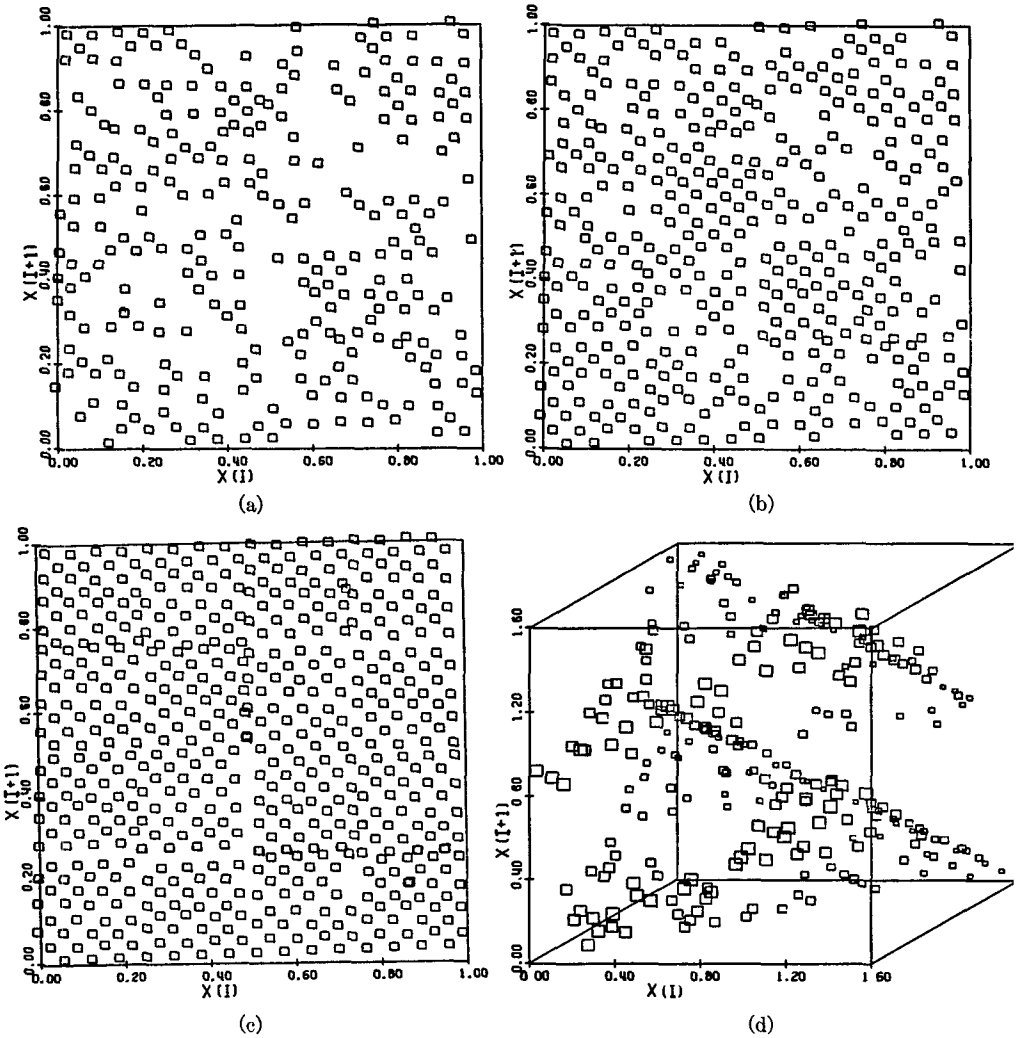


FIG. 2. RNG FSR: $X^9 + X^4 + 1$, (a) 127, (b) 255, (c) 511 points plotted (successive pairs) in 2-space, (d) 255 successive 3-tuples plotted in 3-space

A_0	1 1 1 1 1	A_{10}	0 1 0 0 0	A_{20}	1 0 1 1 0
A_1	1 1 0 0 0	A_{11}	1 1 0 1 0	A_{21}	0 1 0 1 1
A_2	0 1 1 1 0	A_{12}	1 1 1 0 0	A_{22}	0 0 0 0 1
A_3	0 0 1 0 1	A_{13}	0 0 0 1 1	A_{23}	0 1 0 0 1
A_4	0 0 1 0 0	A_{14}	1 1 0 1 1	A_{24}	1 0 0 1 1
A_5	0 1 1 0 1	A_{15}	1 0 1 0 1	A_{25}	0 1 1 1 1
A_6	1 1 1 1 0	A_{16}	1 0 0 0 0	A_{26}	0 1 1 0 0
A_7	1 0 0 0 1	A_{17}	1 0 1 0 0	A_{27}	1 0 1 1 1
A_8	1 1 1 0 1	A_{18}	1 1 0 0 1	A_{28}	0 0 0 1 0
A_9	0 1 0 1 0	A_{19}	0 0 1 1 1	A_{29}	1 0 0 1 0
				A_{30}	0 0 1 1 0

FIG 3 "Kendall sequence" for the polynomial $x^5 + x^2 + 1$

ister based on primitive polynomial $x^p + x^q + 1(10)$. For example, $x^5 + x^2 + 1$ and $a_0 = a_1 = a_2 = a_3 = a_4 = 1$ yields $\{a_i\}_0^{30} = \{1111100011011101010000100101100\}$. Selecting 5-tuples, A_i, by Kendall's algorithm produces the random numbers seen in Figure 3.

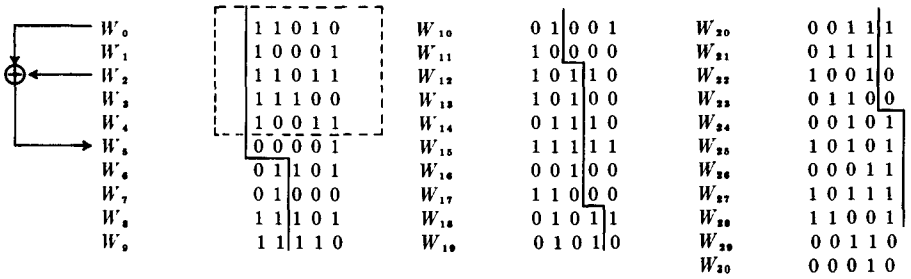


FIG. 4. GFSR sequence for polynomial $x^5 + x^2 + 1$ with delay -6 (=25) between each column

It is important to observe that each A_i (with the exception of 0) occurs once and only once in the full period of a Tausworthe pseudorandom sequence (the "Kendall sequence" is one specific "Tausworthe sequence") [7].

The idea behind the generalized feedback shift register pseudorandom number algorithm (GFSR) is that the basic shift register sequence $\{a_i\}$ based on primitive trinomial $x^p + x^q + 1$ is set into j columns, $j \leq p$, with a judiciously selected *delay* between columns. An example will make the basic GFSR algorithm clear. Again choose primitive trinomial $x^5 + x^2 + 1$. The basic sequence of $\{a_i\}$ is copied in the first column of Figure 4. For this particular polynomial the second column was formed by *delaying* the first columns by 25 ($25 - 31 = -6$, depending on the orientation of the "circle" of bits) bit positions; the third column was obtained by *delaying* the second column by another -6 bit positions; and so on until all five columns have been filled.

Since each column obeys the recurrence $a_k = a_{k-p+q} \oplus a_{k-p}$, each word must also obey $W_k = W_{k-p+q} \oplus W_{k-p}$.

Carefully observe that each W_i occurred once and only once in the full period of $2^5 - 1 = 31$ numbers.

Do arbitrary delays between the columns ensure that each number between 1 and $2^p - 1$ occurs once and only once in each period? No, but any "starting matrix" (W_0, W_1, W_2, W_3, W_4 for $x^5 + x^2 + 1$) can be checked to see if this desirable property holds and, second, the number of different sequences with this property can be analytically computed.

For clarity we will proceed using the example in Figure 4. The *companion* matrix for polynomial $x^5 + x^2 + 1$ is

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

so

$$\begin{matrix} W_0 & W_1 & W_2 & W_3 & W_4 \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & & 1 & 0 & 0 \\ 0 & &) & 1 & 0 \\ 1 & \text{.} & 0 & 1 & 1 \end{bmatrix}$$
$$\begin{matrix} W_1 & W_2 & W_3 & W_4 & W_5 \end{matrix}$$

or in matrix notation $\mathbf{W}_0\mathbf{C} = \mathbf{W}_1$. After $2^5 - 1 = 31$ applic ations of this matrix recurrence $\mathbf{W}_0\mathbf{C}^{31} = \mathbf{W}_0$ or $\mathbf{C}^{31}$ is congruent modulo 2 to the identity matrix. Clearly $\mathbf{C}$, $\mathbf{C}^2$, $\mathbf{C}^3$, $\cdots$, and $\mathbf{C}^{30}$ are all different by virtue of the period of the shift register sequence. Thus for all $\mathbf{W}_0\mathbf{C}^k$, $1 \leq k \leq 30$, to be different, it is only necessary that $\mathbf{W}_0$ have linearly independent rows. This is easily checked.

The number of different sequences which can be produced by the GFSR algorithm is equal to the number of ways different $\mathbf{W}_0$ matrices with linearly independent rows can be selected.

Let r_1, r_2, r_3, r_4, and r_5 denote the five rows of matrix $\mathbf{W}_0$. Row r_1 can be selected to be any of the $2^5 - 1 = 31$ nonzero binary vectors. Row r_2 can be any of the $2^5 - 1 - 1 = 30$ binary vectors which excludes r_1 and the zero vector. Row r_3 can be any of the $2^5 - 1 - 3 = 28$ binary vectors which excludes the zero vector and any linear combination of r_1 and r_2. Row r_4 can be any of the $2^5 - 1 - 7 = 24$ binary vectors excluding the zero vector and linear combinations of r_1, r_2, r_3. The same argument can be extended to selection of r_4. Thus with the GFSR algorithm it is possible to produce $(2^5 - 1)(2^5 - 2)(2^5 - 4)(2^5 - 8)(2^5 - 16) = (31)(30)(28)(24)(16) = 9{,}999{,}360$ different sequences based on primitive polynomial $x^5 + x^2 + 1$.

The GFSR algorithm is,

GFSR:
0. If $k \neq 0$, go to 2 (k initially zero)
1. Initialize, $W_0, \cdots, W_{p-1}$ using a delayed basic sequence, $\{a_i\}$ to obtain each column of $W_0, \cdots, W_{p-1}$.
2. $k \leftarrow k + 1$.
3. If $k > p$, set $k \leftarrow 1$.
4. $j \leftarrow k + q$.
5. If $j > p$, set $j \leftarrow j - p$.
6. EXCLUSIVE-OR, $W_k \oplus W_j$.
7. Store, $W_k \leftarrow W_k \oplus W_j$.

A rotating table of p words is kept in GFSR, which is implemented as a FORTRAN function in Figure 5. A FORTRAN function allows complete generality to be realized, as shown by the results for IBM 360/67, SRU 1108, CDC 6400, and HP 2116A seen in Figure 6. Although the compiled FORTRAN code of Figure 5 correctly implemented RAND for these four computers, some compilers may not implement full word logical operations or others may optimize logical computations. Nevertheless, GFSR pseudorandom numbers may be checked for validity against the results in Figure 6.

2. *Initialization of GFSR Algorithm*

The GFSR algorithm is self-initializing in the sense that delayed replicas are produced by the same procedure that generates full words. Linear independence of starting columns

```
      FUNCTION RAND(M,P,Q,INTSIZ)
C
C     M(P)=TABLE OF P PREVIOUS RANDOM NUMBERS.
C     P,Q=POLYNOMIAL PARAMETERS:X**P+X**Q+1.
C     .NOT. OPERATOR IMPLEMENTED IN ARITHMETIC.
C     INTSIZ=INTEGER SIZE (BITS) OF HOST MACHINE: E.G.,
C     IBM 360, 31; CDC 6000, 48; SRU 1100, 35; HP 2100, 15.
C
      LOGICAL AA,BB,LCOMPJ,LCOMPK
      INTEGER A,B,P,Q,INTSIZ,M(1)
      EQUIVALENCE (AA,A),(BB,B),(MCOMPJ,LCOMPJ),(MCOMPK,LCOMPK)
      DATA J/0/
      N=(2**(INTSIZ-1)-1)*2+1
      J=J+1
      IF(J.GT.P) J=1
      K=J+Q
      IF(K.GT.P) K=K-P
      MCOMPJ=N-M(J)
      MCOMPK=N-M(K)
      A=M(K)
      B=M(J)
      BB=LCOMPJ.AND.AA.OR.LCOMPK.AND.BB
      M(J)=B
      RAND=FLOAT(M(J))/FLOAT(N)
      RETURN
      END
```

FIG. 5. FORTRAN implementation of GFSR algorithm. Initialization is done by SETR in Figure 7

0 36963295936584470	0.36963297000000000
0 40631365776062010	0 40631372000000000
0 42877840995788570	0 42877845000000000
0 47411382198333740	0.47411389000000000
0 95315784215927120	0.95315778000000000
(a)	(b)
0.36963297409225149	0 36964017152786255
0.40631371808778027	0 40632343292236328
0.42877845193692465	0.42878508567810059
0.47411388879095284	0 47410506010055542
0 95315778681866803	0 95318460464477539
(c)	(d)

FIG 6 The first five normalized pseudorandom numbers from GFSR $x^{98} + x^{27} + 1$, delayed column = 9800, produced on the (a) IBM 360, (b) SRU 1108, (c) CDC 6400, and (d) HP 2116A computers

is guaranteed if the maximum delay measured from the leftmost column is less than the full period, $2^p - 1$ (if the "constant delay" between each column is relatively prime to $2^p - 1$, then maximum delay can exceed the full period). Using this procedure every p-tuple is generated (except all zeros) before any p-tuple repeats. Each initial column is a p-tuple, and therefore, must be independent of all others. For example, in Figure 4, initialization can be done from most significant to least significant bits (left-to-right) starting with [11111]. The recurrence relation is applied 25 times, here, to get the next column [10110]. A second 25 applications results in [00010], a third 25 applications results in [10101], and finally, [01101] is obtained. The recurrence is applied by calling the GFSR RNG with zero fills placed to the right. Each new column is shifted to the right after being generated, and the original column [11111] is replaced at the extreme left column. In more realistic (bigger word size) generators, the full period will not be exhausted by initialization in keeping with conditions for linear independence (here, linear independence was guaranteed because 25 is relatively prime to 31).

For ease in implementation the [111...] starting p-tuple is used in SETR as given in Figure 7. However, SETR applies $5000p$ additional delays so that the leading [111...] column is "recurrenced" also, thus, giving a random pattern of leading bits rather than all ones. This additional "recurrencing" is necessary to allow [111...] to "die out." The effects of such a regular pattern carry over to later p-tuples. For example, [111...], $p = 98$, $q = 27$, is transformed to 72 zeros, 26 ones; next to 45 zeros, 53 ones, etc. The groupings of all ones or all zeros become increasingly smaller and more "random" through repeated application of the recurrence relation. "Damping" of the initial p-tuple is done in SETR by applying the recurrence relation $5000p$ times to full word starting values. It should be noted that if generality relative to word size is not desired, then additional "recurrencing" is not necessary when initialization is performed from least significant to most significant bits. The most significant bit will have been delayed $p \cdot$DELAY times since it is "recurrenced" once for each bit in a p-bit word (the least significant bit is a one). Thus, the additional "randomizing" of the initial [111...] pattern will have been done without additional labor.

Finally, SETR returns a value for the number of linearly independent columns available to the initialization procedure.[1]

3. *Generality of GFSR*

The parallel nature of GFSR immediately generalizes to L-bit (integer size) machines independent of the relation between L and p. Thus, for $L < p$, many repeated numbers will occur, but cycle length, m, is still $2^p - 1$. The case where $L = 3$ and $x^5 + x^2 + 1$,

[1] For any particular computer SETR should only be run once and the table constants entered in a DATA statement in RAND. The DO 6 and DO 5 were necessary (rather than DO 5, DO 5): some compilers multiply L*5000 and implement a single DO loop. Since L*5000 is often large, this value sometimes exceeds the maximum value which can be held in an index register on some computers.

```
      INTEGER FUNCTION SETR(M,P,DELAY,Q,INTSIZ)
C
C     SETR=COLUMN NUMBER OF REPEATING ONE PATTERN.  IF SETR <= P,
C     THEN AN IMPROPER SHIFT LENGTH HAS BEEN SELECTED.
C     M(P)=TABLE OF RANDOM NUMBERS TO BE INITIALIZED.
C     P,Q=POLYNOMIAL PARAMETERS: X**P+X**Q+1.
C     DELAY=RELATIVE DELAY BETWEEN COLUMNS OF M(P), IN BITS.
C     INTSIZ=INTEGER SIZE (BITS) OF HOST MACHINE: E.G.,
C     IBM 360, 31; CDC 6000, 48; SRU 1100, 35; HP 2100, 15.
C
      INTEGER DELAY,P,Q,ONE,INTSIZ,M(1)
      SETR=P+1
      ONE=2**(INTSIZ-1)
      DO 1 I=1,P
1     M(I)=ONE
      DO 4 K=1,INTSIZ
      DO 2 J=1,DELAY
2     X=RAND(M,P,Q,INTSIZ)
      KOUNT=0
      DO 3 I=1,P
      ITEMP=ONE/2**(K-1)
      ITEMP=(M(I)-M(I)/ONE*ONE)/ITEMP
      IF(ITEMP.EQ.1) KOUNT=KOUNT+1
      IF(K.EQ.INTSIZ) GO TO 3
      M(I)=M(I)/2+ONE
3     CONTINUE
      IF(KOUNT.EQ.P) SETR=K
4     CONTINUE
      DO 6 I=1,5000
      DO 5 J=1,P
5     X=RAND(M,P,Q,INTSIZ)
6     CONTINUE
      RETURN
      END
```

FIG 7. FORTRAN implementation of SETR to initialize GFSR algorithm

W_0	1 1 0	W_{10}	0 1 0	W_{20}	0 0 1
W_1	1 0 0	W_{11}	1 0 0	W_{21}	0 1 1
W_2	1 1 0	W_{12}	1 0 1	W_{22}	1 0 0
W_3	1 1 1	W_{13}	1 0 1	W_{23}	0 1 1
W_4	1 0 0	W_{14}	0 1 1	W_{24}	0 0 1
W_5	0 0 0	W_{15}	1 1 1	W_{25}	1 0 1
W_6	0 1 1	W_{16}	0 0 1	W_{26}	0 0 0
W_7	0 1 0	W_{17}	1 1 0	W_{27}	1 0 1
W_8	1 1 1	W_{18}	0 1 0	W_{28}	1 1 0
W_9	1 1 1	W_{19}	0 1 0	W_{29}	0 0 1
				W_{30}	0 0 0

FIG. 8. GFSR algorithm for $L = 3$, polynomial $x^5 + x^2 + 1$

from Figure 4, is demonstrated in Figure 8. Here 2^{p-L} nonzero duplicates and $2^{p-L} - 1$ zeros are produced in one full period.

Very long period sequences can be generated on any L-bit machine merely by selecting p large. A partial table of primitive polynomials of large p is reproduced in Figure 9 [11]. A complete table can be found in [12, 13].

4. *Period of GFSR*

An "unlimited" period is possible without increasing word size of host machines. For example, $M = 2^{532} - 1$ is obtained using $x^{532} + x^{37} + 1$ from the table in Figure 9. To exhaust this cycle would require many years on a very fast computer, i.e. if 10^6 numbers/second were generated, approximately 10^{150} years would be needed to complete the cycle!

p	q
47	5,14,20,21
95	11,17
98	11,27
111	10,49
124	37
170	23
250	103
380	47
476	15,141
532	37

FIG 9. Primitive polynomials, $x^p + x^q + 1$, p large

More importantly, though, is the repeatability of numbers within a full period. Thus, an extended sequence is obtained with desirable n-space properties.

5. *Mean, Variance, and Correlation of GFSR*

The theoretical *mean* and *variance* of a GFSR sequence is guaranteed by periodicity;

$$\mu = \frac{1}{m}\sum_{i=0}^{m-1} W_i; \qquad m = 2^p - 1.$$

For an L-bit machine, 2^{p-L} nonzero duplicates and $2^{p-L} - 1$ zeros will be generated before the entire sequence repeats;

$$\mu = \frac{2^{p-L}}{m}\sum_{i=1}^{2^L-1} i + \frac{2^{p-L}-1}{m}\sum_{i=1}^{2^L-1} 0 = \frac{2^{p-L}}{m}\left[\frac{(2^L-1)(2^L)}{2} - 1\right] \approx \frac{1}{2}\left(\frac{2^{p+L}}{m}\right).$$

In the normalized (0, 1) interval $\mu_0 \approx \frac{1}{2}$.

The *variance*,

$$\sigma^2 = \frac{1}{m}\sum_{i=0}^{m-1} W_i^2 - \mu^2 = \frac{(2^{p-L})}{m}\sum_{i=1}^{2^L-1} i^2 - \mu^2 \approx \frac{1}{3}\left(\frac{2^{p+2L}}{m}\right) - \mu^2$$

and normalized to (0, 1),

$$\sigma_0^2 \approx \tfrac{1}{3} - \tfrac{1}{4} = \tfrac{1}{12}.$$

The correlation obtained by averaging over the entire period,

$$E[R(t)] = \frac{1}{m}\sum_{j=0}^{m-1}\frac{1}{N}\sum_{i=0}^{N-1} W_i W_{i+t},$$

can be derived using techniques identical to Tausworthe's [7]. However, intuition indicates that columns with nearly the same delay and hence nearly equal, bit-by-bit, should result in large correlation coefficients. The correlation intuitively must decrease as relative delay increases. Full period analysis by Tausworthe does not properly model this microstructure and tells us nothing about short sequences. Therefore, an empirical rule is used which computes the maximum correlation coefficient over a range $0 \leq t \leq 50$. A plot of maximum correlation coefficient versus relative delay between columns shown in Figure 10 indicates a relative column delay of $100p$ or more to be satisfactory. It is possible to find smaller delays which also give satisfactory correlation, but several polynomials were tested and all found to be "safe" with delays of order $100p$. Finally, selection of delays which are multiples of p assures linear independence of starting values discussed in Section 2.

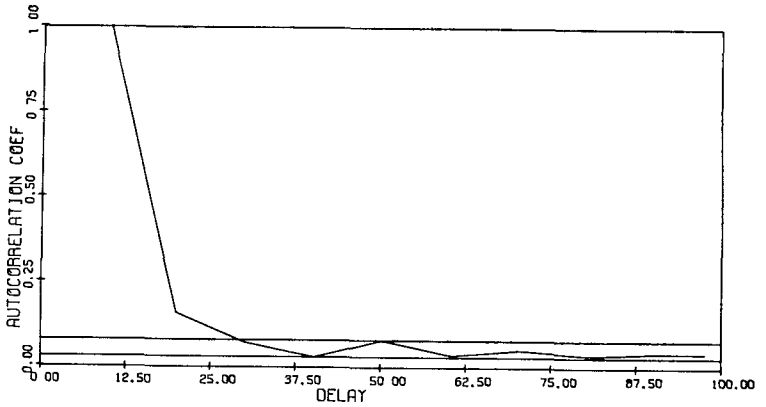
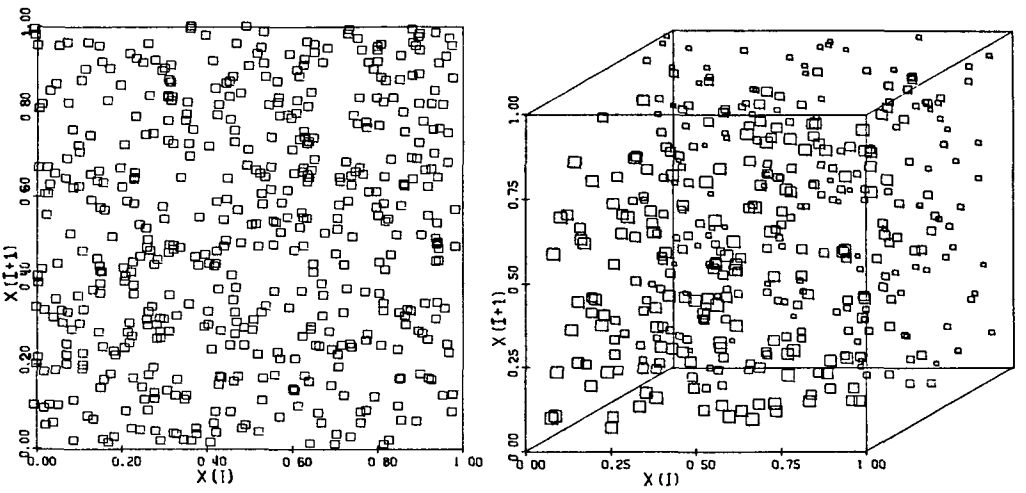


FIG. 10 Maximum correlation versus relative column delay (measured in p units) for GFSR RNG using 15-bits, $x^{98} + x^{27} + 1$



(a) (b)

FIG 11 (a) Two-dimensional plot of GFSR, $X^{31} + X^{13} + 1$, 9-bit word size and delay of 93, (b) three-dimensional plot of GFSR, $X^{31} + X^{13} + 1$, delay = 93

6. *Multidimensional Uniformity of GFSR*

Figure 11 shows a much improved 9-bit generator compared with Lehmer and Kendall RNG's shown in Figures 1 and 2. The underlying reason, of course, is the longer than $m = 2^9$ period, and the repeatability of numbers within one cycle of the generator.

To fill in m^n cells in n-space, $(2^L)^n < 2^p - 1$, or $nL \leq p$. Therefore, a necessary condition for n-space uniformity is that $n \leq p/L$. For example, suppose $L = 15$, $p = 98$, then uniformity may be possible up to dimension, $n = 6$.

n-Space uniformity cannot be guaranteed without knowledge of the order of numbers generated by GFSR. Known order can be exploited for the purpose of designing tests which the GFSR will fail. Rather than take this negative approach, greater emphasis is placed upon n-space uniformity for actual use.

GFSR THEOREM. *The sequence of L-bit numbers generated by GFSR, $x^q + x^p + 1$, $p \geq L$, has*

(1) *period, $m = 2^p - 1$, if $x^p + x^q + 1$ is primitive,*

(2) *normalized mean*, $\mu_0 \approx \frac{1}{2}$,
(3) *normalized variance*, $\sigma_0^2 \approx \frac{1}{12}$,
(4) *potential n-space uniformity for* $n \leq p/L$.

7. *Conclusions*

The advantages of GFSR RNG's are:

(1) *Speed*: One EXCLUSIVE-OR versus a multiply/reduction modulo m for Lehmer, and two EXCLUSIVE-ORS/two shifts for Kendall's algorithm.

(2) *Generality*: A standard FORTRAN subprogram can be implemented on any computer independent of the word size. A small word size merely reduces the resolution of random numbers produced, but high order bits will be unchanged on any machine. Comparison with sequences obtained on other machines using a FORTRAN program are given in Figure 6.

(3) *"Unlimited" period*: Any primitive polynomial can be implemented when sufficient memory is available for storage of p words. For example, $x^{98} + x^{27} + 1$ can be used on a 16-, 24-, 32-, 36-, 48-, 60-bit machine and a cycle length of $2^{98} - 1$ realized on them all.

Moreover, the speed/cost ratio is further enhanced when GFSR is implemented as a microprogrammed instruction. Figure 12 shows a microprogram which computes a pseudorandom number each time RND REG1, TABLE, is issued. The Interdata 4 machine instruction $D4_{16}$ was wired into read-only-memory as shown in Figure 13.

Impressive speed (24 microseconds on the Interdata 4, which is equivalent to 5–6 microseconds on an IBM 360/65) and extended period are realized on a small word size machine (16 bits). The period of $2^{98} - 1$ greatly surpasses the previously attainable periods of $2^{15} - 1$ or $2^{16} - 1$.

The table for RND REG1, TABLE is organized as shown in Figure 14. Thus, $x^n + x^s + 1$ is generated using index I.

8. *Testing*

The following empirical tests on 10,000 GFSR-produced numbers were made using $X^{98} + X^{27} + 1$, 15-bits, and chi-square statistic [2–4, 6].

The *frequency test* counts the number of numbers falling in each of 100 equal cells in the (0, 1) interval.

Yule's test counts the number of sums-of-5 digits falling in each of 45 cells. This is applied to each of the four most significant decimal digits obtained by scaling the normalized (0, 1) pseudorandom number.

The *gap test* counts the length of gaps between successive like digits formed by scaling normalized RNG number to decimal digits. The test is applied to digits 0 through 9.

The *autocorrelation test* computes the maximum normalized autocorrelation coefficient up to lag 50. Acceptable correlation is in the interval (.03, .08).

The D^2 *test* compares the theoretical distribution of a random line in two dimensions with the distribution obtained empirically.

The *serial test* counts the number of pairs of numbers from an RNG falling in 100 equal cells. Here, pairs of decimal digits are combined to give an integer from zero to ninety-nine. A frequency test is applied to the 100 cells.

The *runs test* computes the longest run, the frequency of each run compared with the theoretical distribution, and number of runs above/below the mean. The total number of runs is compared with the expected number.

The *minimum/maximum-of-n-test* compares the empirical with the theoretical distribution of the minimum/maximum-of-n numbers. The number of extremes is counted for each of 100 cells and a frequency test is applied for $n = 2, 4, 6 \cdots, 20$.

MODEL 4 MICRO CODE

```
                        ORG X'428'
0428  5004    RND       L   RAH,H(RND)
0429  3080              C   SB
042A  4453              L   MR4,MAR          SAVE TABL IN MR4.
042B  43A3              L   MR3,MDR          SAVE I IN MR3.
042C  5802              L   AR,X'2'
042D  C550              A   MAR,MAR,NF+NC
042E  3100              C   MR               GET S,N: LOC TABL+2.
042F  C050              A   MR0,MAR,NF+NC    SAVE TABL+4 IN MR0.
0430  58FF              L   AR,X'FF'
0431  82A3              N   MR2,MDR,NF       MASK-OFF S; N IN MR2.
0432  48AF              L   AR,MDR,CS
0433  91FF              N   MR1,X'FF'        CROSS SHIFT AND MASK-C
0434  4833              L   AR,MR3           COMPARE I:N.
0435  E824              S   AR,MR2,NC
0436  1138              B   L,OK             BRANCH IF I<N.
0437  5300              L   MR3,X'00'        OTHERWISE, SET I=0
0438  4813    OK        L   AR,MR1
0439  C130              A   MR1,MR3,NF+NC    J=I+S
043A  4813              L   AR,MR1
043B  E224              S   MR2,MR2,NC       (MR2)=J-N.
043C  113E              B   L,OK2            BRANCH IF J<N.
043D  4123              L   MR1,MR2          OTHERWISE, SET J=J-N.
043E  4818    OK2       L   AR,MR1,SL+NC     D1=2*J.
043F  C500              A   MAR,MR0,NF+NC    LOC TABL+D1+4.
0440  3100              C   MR               GET TABL(J).
0441  4838              L   AR,MR3,SL+NC     D2=2*I.
0442  C500              A   MAR,MR0,NF+NC    LOC TABL+D2+4.
0443  48A3              L   AR,MDR           SAVE TABL(J).
0444  3100              C   MR               GET TABL(I).
0445  AEA3              X   YD,MDR,NF        EX-OR TABL(J)+TABL(I).
0446  4AE3              L   MDR,YD
0447  3200              C   MW               RESULT IN TABL(I).
0448  5801              L   AR,X'01'         INCREMENT.
0449  CA30              A   MDR,MR3,NF+NC    I=I+1.
044A  4543              L   MAR,MR4
044B  3200              C   MW               NEW I IN LOC TABL.
044C  4563              L   MAR,LOC
044D  066B              D   LOC,LOC,P2N
                        END
```

FIG 12. Interdata 4 microprogram for GFSR Execute, RND REG1, TABLE. The resulting random integer is returned to register REG1, and the address of TABLE locates the memory table. To interpret the code L-load, C = command, A = add, N = and, S = subtract, B = branch, X = exclusive-or, and D = decode. There are microcode registers MR0 through MR4, memory address register (MAR), and memory data register (MDR) which refer to core memory. The third operand modifies the operation for example, NC means "no carry"

The *conditional bit test* counts the number of one bits in the jth bit position given the $j-1$ previous bits. A binary tree is formed with a branch at each bit position. The tree of bit counts is then compared with the expected value of $(\frac{1}{2}) * 10{,}000 = 5000$. If the bits are indeed independent then the empirical value matches expected value.

The *finite Fourier transform (FFT) test* tests for a "flat" spectrum by the statistics

$$U = (S - \tfrac{1}{2})\sqrt{(12M)}, \qquad P_n = \sum_{r=1}^{n} |a_n|^2 \Big/ \sum_{r=1}^{MH} |a_n|^2, \quad n = 1, 2, \cdots, M+1,$$

$$S = \frac{1}{M}\sum_{n=0}^{M} p_n, \qquad M = \begin{cases} N/2, & N \text{ even}, \\ (N-3)/2, & N \text{ odd}, \end{cases}$$

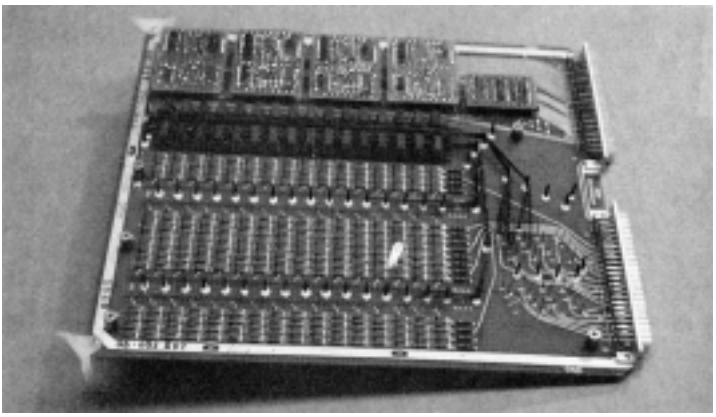


FIG. 13. Interdata 4 read-only-memory board with RND instruction wired in. Note the two rows of transformers with interwoven wire. Each transformer provides one bit of memory

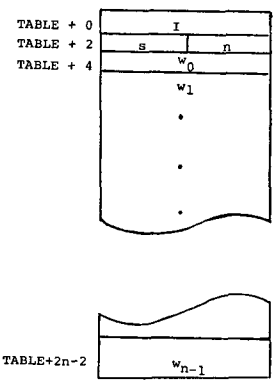


FIG 14. Organization of TABLE for microprogram RND GFSR uses $x^n + x^s + 1$ by EXCLUSIVE-OR of w_n with w_{n+s}

and the transformed sequence

$$A_n = \frac{1}{N}\sum_{j=0}^{N-1} R_j \exp(-2\pi ijn/N); \qquad n = 0, 1, \cdots, N-1; \quad R_j = \text{random number.}$$

The *scattering experiment simulation* compares the expected distribution of scattering angles with theoretically expected values. A point on the surface of a unit sphere is randomly chosen thus giving a solid angle. This solid angle represents the deflection of a neutron after colliding with an atom. The solid angle is rejected to weight according to $\sigma(\theta) = \pi(1 - \theta/\pi) : 0 \leq \theta \leq \pi$. The empirical distribution of θ is compared with the theoretical using a chi-square test on 36 intervals in $(0, \pi)$.

A *visual test* is done by observing 2, 3-space plots of successive n-tuples as shown in Figure 11. A CRT display is particularly useful for observing order by dynamic plotting of points as they are generated.

9. *Results of Tests*

The 15-bit GFSR algorithm performed satisfactorily in the above tests at the 5 percent level of significance. Failure was noted at $n = 8, 14, 20$ in the minimum-of-n test as predicted. However, no failures were noted for $n = 10, 12, 16, 18$ and all n tested in maximum-

of-n test. Thus, a partial indication of nonuniformity for $n > 6$ is given by these tests. For applications requiring uniformity above $n = 6$ one merely uses a higher degree polynomial. For 24 mantissa bits and $X^{98} + X^{27} + 1$, no failure was detected on any of the tests.

The GFSR algorithm must be considered statistically qualified as an RNG while superior in speed and generality to other contemporary RNG's.

ACKNOWLEDGMENTS. We wish to thank Fred M. Ives for his microprogramming expertise and assistance for wiring the read-only-memory board. Also, we wish to thank Peter D. Gross, W. R. Thorson, and A. F Kupinski for running programs on the SRU 1108 and CDC 6400 computers. Terry E. Hamm, compiler writer, Data Products Division, Hewlett-Packard, Palo Alto, California, filled us in on the history of the DO loop and ran our programs with the new HP compiler on the 2116A computer.

The initial draft of the paper was circulated to several scientists. R. R. Coveyou, L. R. Turner, K. C. Wang, and J. RB. Whittlesey made many valuable suggestions on shortcomings and how to improve this paper. Frank Engle, Jr., Chairman, ANSI X3J3 Standards Committee, pointed out to us that RAND did not conform to ANSI X3.9-1966 FORTRAN specifications. He coded two conforming versions of RAND which execute much more slowly than our nonstandard code. T. G. Ostrom gave us the solution to the problem of counting the number of different zero-one matrices with linearly independent rows. Finally, J. P. R. Tootill not only corrected some of our ideas about Kendall's algorithm but modified his "decimated-Kendall-sequence" algorithm [9] to study our $x^{98} + x^{27} + 1$ sequence and reported, "We paid no particular attention to efficiency of method or coding, so our algorithm as modified would win no prizes for speed, but it did establish that your sequence to 16-bit accuracy is, in fact, 6-distributed." Again we express appreciation to all these scientists.

REFERENCES

1. COVEYOU, R. R., AND MACPHERSON, R. D. Fourier analysis of uniform random number generators *J. ACM 14*, 1 (Jan 1967), 100–119.
2. KENDALL, M. G., AND SMITH, B. B. Randomness and random sampling numbers *J Roy Statist. Soc 101* (1938), 162–164.
3. LEWIS, P. A W , GOODMAN, A S , AND MILLER, J. M A pseudorandom number generator for the SYSTEM/360 *IBM Syst. J. 8*, 2 (1969), 136–146.
4. MACLAREN, M D , AND MARSAGLIA, G. Uniform random number generators. *J. ACM 12*, 1 (1965), 83–89.
5. MARSAGLIA, G. Random numbers fall mainly on the planes. *Proc Nat. Acad. Sci. 61*, 1 (Sept 1968), 25–28.
6. PAYNE, W. H , AND LEWIS, T. G Conditional bit sampling: Accuracy and speed In *Mathematical Software*, J. R. Rice (Ed), Academic, New York, 1971, pp 331–345.
7. TAUSWORTHE, R. C Random numbers generated by linear recurrence modulo two *Math. Comput 19* (1965), 201–209.
8. TOOTILL, J P. R., ROBINSON, W. D , AND ADAMS, A G The runs up-and-down performance of Tausworthe pseudo-random number generators. *J ACM 18*, 3 (1971), 381–399.
9. TOOTILL, J. P. R , ROBINSON, W. D , AND EAGLE, D. J. An asymptotically random Tausworthe sequence. *J. ACM 20*, 3 (July 1973), 469–481 (this issue)
10. WHITTLESEY, J RB. A comparison of the correlational behavior of random number generators for the IBM 360. *Comm ACM 11*, 9 (Sept 1968), 641–644
11. WHITTLESEY, J. RB., AND GRIESE, P. Multi-dimensional pseudorandom non-uniformity. Proc. of the UMR-Mervin J Kelly Communication Conference, Oct. 1970, U. of Missouri at Rolla, Rolla, Mo , pp. 15-4-1–15-4-6
12. ZIERLER, N. Primitive trinomials whose degree is a Mersenne exponent. *Inform Contr. 15* (1969), 67–69.
13. ZIERLER, N , AND BRILLHART, J. On primitive trinomials (mod 2), II. *Inform. Contr. 14* (1969), 566–569

RECEIVED AUGUST 1971; REVISED JULY 1972