JOURNAL OF COMPUTATIONAL PHYSICS 49, 478–489 (1983)

An Algorithm for Pseudo Random Number Generation Suitable for Large Scale Integration

ROBERT B. PEARSON

Cen–Saclay, 91191 Gif-sur-Yvette, Cédex France

Received June 15, 1982

An algorithm for producing pseudo random numbers is described. It is based on linear shift register sequences, and has the desirable properties of minimal size and rapid execution, producing a new random number each clock cycle. The mathematical basis for its operation is described.

I. INTRODUCTION

There are many applications which call for long sequences of pseudo random numbers. An important example in the field of computational physics is the Monte Carlo simulation [1] of systems in statistical mechanics. The author has been working on a project to integrate a special purpose processor for carrying out Monte Carlo simulations onto a single VLSI "chip." This device will be described elsewhere [2]. One problem which came up in this design was the need for a self-contained pseudo random number generator (PRNG). This generator has to meet very demanding requirements.

(i) It must fit into the layout of a single chip.

(ii) It must produce a new random number every clock cycle.

(iii) It must produce random numbers of very high quality.

The first requirement is the most difficult, and has led to an algorithm which is different from most standard algorithms implemented in software. The last two requirements essentially force one to consider a generator based on shift register sequences, or Tausworth sequences [3]. This paper is intended to describe the mathematical basis for this algorithm since it may be of more general application than our device.

There exist sofware methods for generating Tausworth sequences where the entire random number is generated in parallel by the "exclusive or" instruction [4], from a table of the p previous random numbers. These algorithms are simple and can easily be designed into hardware to generate a random number each clock cycle. But, in order for the generator to be of sufficiently high quality p must be large ($\gtrsim 100$) and the storage requirements are too great for this application. The requirement that a new nmber be produced each clock cycle makes it necessary to generate all of the

0021-9991/83 $3.00

bits in parallel. Thus one is led to consider the most general shift register scheme where m bits are produced by m exclusive or 's each clock cycle. The mathematics of this situation, which is a straightforward generalization of the simple case [5], is described in Section II of this paper. We will find that it is possible to have m parallel fed back shift registers in which the period of the random number, and each subset of its bits is $2^n - 1$, where n is the *total* number of bits of storage.

In Section III some examples will be discussed of PRNGs which satisfy the requirements found in Section II. Also compiled there are lists of "good" generators which are not exhaustive but allow one to carry out a design for most practical situations.

II. Multi-Bit Fedback Shift Registers

Figure 1 shows the configuration of a typical fedback shift register (FSR). During each clock cycle each of the bits is replaced by the value of the preceding bit except for the first one which is replaced by the modulo 2 sum of some set of the bits in the shift register. If the initial state of the register is all zeros, then it will remain so for all time. If the initial state is nonzero, then the register will cycle through some number of states less than or equal to $2^n - 1$. The mathematics of this situation is very interesting, but for our purposes it is sufficient to note that the important properties of the sequence of bits produced by the FSR can be deduced from something called the generating function. Suppose we denote the values of the bits in the register at time t in the ith bit position, labeled from 1, by $X_i(t)$. Then defining $X(t) = X_1(t)$,

$$X(t) = \sum_{i=1}^{n} c_i X_i(t-1), \qquad \text{mod } 2, \tag{2.1}$$

$$X_i(t) = X_{i-1}(t-1), \qquad i > 1, \tag{2.2}$$

where $c_i = 1$ if the ith bit is incluced in the sum and zero, otherwise. The generating function is defined by

$$G(\lambda) = \sum_{t=0}^{\infty} X(t)\,\lambda^t. \tag{2.3}$$

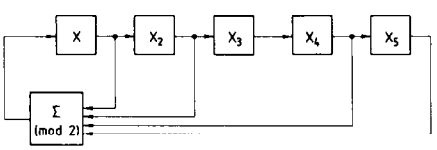

FIG. 1. A typical shift register with 5 bits, and characteristic polynomial $1 + \lambda + \lambda^2 + \lambda^4 + \lambda^5$.

If Eq. (2.1) is multiplied by λ^t and summed over t and $X_i(t)$ is replaced by $X_1(t-i+1)$ by Eq. (2.2), we obtain an expression for G in terms of the c's and the initial values of the shift register bits $X_i(t)$, usually called the "seed."

$$G(\lambda) = \sum_{j=0}^{n-1} \lambda^j \sum_{i=j+1}^{n} c_i X_{i-j}(-1) \Big/ \left(1 + \sum_{j=1}^{n} c_i \lambda^i\right). \tag{2.4}$$

The denominator of this expression depends only on the c's and is independent of the initial configuration of the system. It is called the characteristic polynomial of the shift register. The numerator and denominator are polynomials in λ of degree less than or equal to $n-1$ and n, respectively. We assume without loss of generality that c_n is 1. Several important things are related to the characteristic polynomial. In particular it may be shown that [5]:

(1) A necessary condition that the period of the shift register be maximal, i.e., $2^n - 1$, is that the characteristic polynomial be primitive over the field of integers mod 2. In other words it cannot be factored where the arithmetic of the coefficients is performed mod 2.

(2) If the polynomial is primitive, then the period of the shift register divides $2^n - 1$. In particular if $2^n - 1$ is a prime, called a Mersenne prime, then the shift register has maximal period.

If the polynomial is factorizable, then the period is less than maximal. A maximal length FSR sequence is called a pseudo-noise (PN) sequence. Because there exist several Mersenne primes at convenient values of n we will generally consider only shift registers with n a Mersenne number.

Given the bit sequence from the FSR we could construct random numbers by the algorithms mentioned in the Introduction, but we have decided that to generate a full word we want m feedback points with a mod 2 sum performed at each point simultaneously. The situation might appear as in Fig. 2. We can label the value of the ith bit in the jth row by $X_{i,j}(t)$ and we have at time t

$$X_{i,j}(t) = X_{(i-1),j}(t-1), \qquad i > 1. \tag{2.5}$$

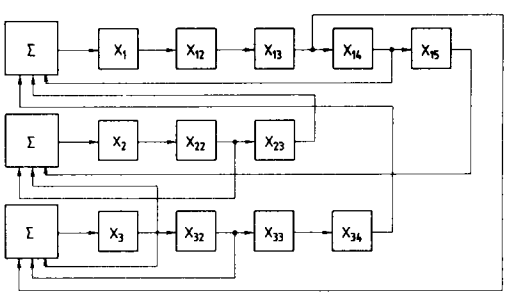

FIG. 2. A nontrivial 3 bit FSR.

The new value of $X_{1,j}(t) \equiv X_j(t)$ could, in principle, come from *any* of the stored bits

$$X_j(t) = \sum_{k=1}^{m} \sum_{i=1}^{n_k} c_{ijk} X_{i,k}(t-1), \qquad \text{mod } 2. \tag{2.6}$$

Again the important quantity is the generating function except now there are m of them

$$G_j(\lambda) = \sum_{t=0}^{\infty} X_j(t)\lambda^t. \tag{2.7}$$

If we, as before, multiply Eqs. (2.5) and (2.6) by λ^t and sum we obtain

$$G_j(\lambda) = \sum_{l=1}^{m} \sum_{k=0}^{n_l-1} \sum_{i=k+1}^{n_l} \lambda^k c_{ijl} X_{(l-k),l}(-1) + \sum_{l=1}^{m} \sum_{i=1}^{n_l} \lambda^i c_{ijl} G_l(\lambda). \tag{2.8}$$

This set of linear equations is solved to give

$$G_j(\lambda) = \sum_{k=1}^{m} F_{jk}^{-1} S_k, \tag{2.9}$$

where S_j is the inhomogeneous term on the rhs of (2.8) and

$$F_{jk} = \delta_{jk} + \sum_{i=1}^{n_k} c_{ijk}\lambda^i. \tag{2.10}$$

For example, the F corresponding to Fig. 2 is

$$F = \begin{pmatrix} 1+\lambda^4 & \lambda^3 & \lambda^4 \\ \lambda^5 & 1+\lambda^2 & \lambda \\ \lambda^3 & 0 & 1+\lambda+\lambda^2 \end{pmatrix}. \tag{2.11}$$

The structure is identical to (2.4) with the numerator, which only depends on the "seed," replaced by the S_j and the denominator replaced by F^{-1}. F may be inverted by Crammers rule as a ratio of determinants, often written

$$F_{ij}^{-1} = (-1)^{i+j} \det F(j \mid i)/\det F. \tag{2.12}$$

The important point is that each of the G_j is given as a ratio of polynomials in λ with the *same* denominator. Thus each bit of the generator behaves as though it were produced by the same effective FSR, with, however, a different seed for each bit. The effective FSR has a characteristic polynomial which is the determinant of F. The degree of $\det F$, or the length of the effective FSR, is bounded by

$$n = \sum_{j=1}^{m} n_j \tag{2.13}$$

and this bound is only reached if (i) $c_{n_k|k} = 1$ for some j, and (ii) these j's are all distinct. In particular cases, factors may cancel between numerator and denominator so that the effective FSRs for some bits are simpler. This is the case when each separate row of the system only couples to itself. But it is also possible, by making $\det F$ a primitive polynomial mod 2, to have the periods of all of the bits of the generator maximal for the total number of bits of storage, e.g. $2^n - 1$. This allows one to meet two of the criteria mentioned in the introduction. First, a number of generated each cycle and second, the period is as long as possible so the space required is a fraction $(1/m)$ of that needed for an equivalent generator of the conventional type.

Now we must consider the independence of the bits within the word [6]. To make progress we must be more specific than Fig. 2 about what we are going to do. The simplest feed back element is a two input XOR. For regularity of design we assume there are two taps on each shift register row. One near the center, and one at the end. Denote these positions by q_j and n_j. Again for simplicity assume one input to the XOR is from the q bit in the same row, and the other one is from the n bit of some other row. The requirement that $\det F$ be primitive leads to the requirement that all of the rows by connected in a circular chain in some order. Thus we have the simple result

$$\det F = \prod_{j=1}^{m} (1 + \lambda^{q_j}) + \prod_{j=1}^{m} \lambda^{n_j} = \prod_{j=1}^{m} (1 + \lambda^{q_j}) + \lambda^{n} \tag{2.14}$$

notice that $\det F$ depends on the lengths of the shift registers only through their sum so that one may take them to be as uniform as possible for a regular layout. One must find a set of q_j such that the polynomial in the form (2.14) is primitive.

For an F in this simple form there is some permutation of the bits so it appears as

$$F = \begin{pmatrix} 1 + \lambda^{q_1} & 0 & 0 & \cdots & \lambda^{n_m} \\ \lambda^{n_1} & 1 + \lambda^{q_2} & 0 & & 0 \\ 0 & \lambda^{n_2} & 1 + \lambda^{n_3} & & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \lambda^{n-1} & 1 + \lambda^{q_m} \end{pmatrix}. \tag{2.15}$$

By Cramers rule

$$F_{ij}^{-1} = \frac{1}{\det F} \begin{cases} \prod_{k=1}^{j-1} (1 + \lambda^{q_k}) \prod_{k=j}^{i-1} \lambda^{n_k} \prod_{k=i+1}^{m} (1 + \lambda^{q_k}), & j \geqslant i \\ \prod_{k=1}^{i-1} \lambda^{n_k} \prod_{k=i+1}^{j-1} (1 + \lambda^{q_k}) \prod_{k=j}^{m} \lambda^{n_k}, & j > i \end{cases} \tag{2.16}$$

If we agree that we have guaranted that the system has full period, for example by making n a Mersenne number and $\det F$ primitive, then it simplifies our analysis to consider a special starting point. Since then every nonzero configuration of X's will occur once in a full period we may consider the case where all $X_{ij}(-1) = 0$ except

$X_{n_m m}$. Then all the S_j in (2.9) are zero except S_1 which is given by $S_1 = 1$. Then we have explicitly

$$G_j = \prod_{k=1}^{j-1} \lambda^{n_k} \prod_{k=j+1}^{m} (1 + \lambda^{q_k})/\det F. \tag{2.17}$$

This result contains information about relationships between the various bits. Consider the first two bits. We have

$$G_1 = \prod_{k=2}^{n} (1 + \lambda^{q_k})/\det F \tag{2.18a}$$

$$G_2 = \lambda^{n_1} \prod_{k=3}^{n} (1 + \lambda^{q_k})/\det F \tag{2.18b}$$

or

$$(1 + \lambda^{q_2})\, G_2 + \lambda^{n_1} G_1 = 0 \qquad (\text{mod } 2). \tag{2.19}$$

Translated into a recurrence relation for X_1, X_2 this is identical to the statement that X_2 is fed with the XOR of itself and X_1 delayed by q_2 and n_1, respectively, which we knew. Now consider bit one and any other bit. We have

$$(\det F)\, G_j = \lambda^{n_1 + n_2 \cdots + n_{j-1}} \prod_{k=j+1}^{n} (1 + \lambda^{q_k}) \tag{2.20}$$

assume for the moment that $n_1 + n_2 + \cdots + n_{j-1} \leqslant n/2$; then we have

$$A(\lambda)\, G_1 + B(\lambda)\, G_j = 0 \tag{2.21}$$

only if

$$A(\lambda) = \lambda^{n_1 + n_2 + \cdots + n_{j-1}} + \cdots . \tag{2.22}$$

In terms of recurrence relations the 1st bit only effect the jth bit after a delay of $n_1 + n_2 + \cdots + n_{j-1}$ cycles. Again this is obvious. In general when we consider collections of two or more bits we find that the simplest relationships between them involve delays of the order of the shortest path between any two of them. This has implications about how uniformly distributed k-tuples of random numbers are. If one has a set of bits which are algebraically independent, then they will takes on values during a full cycle which are uniformly distributed. It is desirable to have the most significant bits of the random number as uniform as possible. Successive values of the most significant bit are independent up to the full length of the effective shift register, n. So that to one bit accuracy the numbers are uniformly distributed on n-cubes. If n_{12} is the shortest distance on the cycle from the most significant to the next most significant bit, then the first n_{12} pairs of bits will be independent and to two bit

accuracy the random numbers will be uniformly distributed on n_{12}-cubes. Clearly, we should make n_{12} as large as possible which is $[n/2]$. Similarly we should spread the remaining bits around so as to maximize the distances between more significant bits. For example if there are 16 bits in the random number generator one could arrange them, with respect to the order implied by (2.15), as

$$1, 9, 5, 10, 3, 11, 6, 12, 2, 13, 7, 14, 4, 15, 8, 16.$$

Let us keep the lengths of each shift register n_j as uniform as possible. Such an arrangement approaches the ideal limit [6] for equidistribution where to t-bit accuracy one has numbers uniformly distributed on $[n/t]$-cubes. In the present scheme we approach this performance with the order of equidistribution to t bit accuracy given by

$$[n/2^{\lceil \ln_2 t \rceil}]. \tag{2.23}$$

For practical purposes this is quite acceptable.

We have discussed the properties of period and equidistribution for our algorithm and shown that they can be controlled. These are important, but one would like to understand further properties of the generator. A particularly important one which is often considered is the serial correlations of lagged pairs of numbers over a full period of length $P = 2^n - 1$.

$$c_k = \frac{1}{P}\sum_{i=1}^{P} Y_i Y_{i+k} - \left(\frac{1}{P}\sum_{i=1}^{P} Y_i\right)^2, \tag{2.24}$$

where considering the random number as a fraction in $[0, 1]$

$$\begin{aligned} Y_i &= 0 \cdot X_1 X_2 \cdots X_m \qquad \text{(Base 2)} \\ &= \sum_{l=1}^{m} X_l(i)\, 2^{-l}. \end{aligned} \tag{2.25}$$

An important property of PN sequences is that the bit correlations obey

$$\frac{1}{P}\sum_{i=1}^{P} X_l X_{l+k} - \left(\frac{1}{P}\sum X_l\right)^2 = O(1/P), \qquad k \neq 0. \tag{2.26}$$

So in particular the correlation between Y_l and Y_{l+k} is $O(1/P)$ unless some of the bit sequences in the two numbers coincide *exactly*. Recall that in our algorithm each of the bits in the random word comes from the same sequence generated by an effective shift register with characteristic polynomial $\det(F)$, but with different seeds. Consequently, there can be no correlations in the same bit position until a full period, but there can be correlations between different bits of the words in some fraction of a

period. As discussed above two adjacent bits in the cycle obey an equation like (2.19) which means that there is some l such that $G_1 = \lambda^l G_2 \pmod F$ or

$$\lambda^{n_1+l} = (1+\lambda^{q_2}) \pmod F. \tag{2.27}$$

The solution of this equation for l is very difficult. The only positive thing to offer on this point is that a general polynomial will be uniformly distributed on $0 < l < P-1$, and if $F(\lambda)$ is sufficiently free of special characteristics then the polynomials $(1+\lambda^q)$ can be considered general polynomials, or the bits of the generator should be uniformly distributed on the full period. There is one situation in which we can predict the delays of the bits in the generator, and that is when all of the $(1+\lambda^q)$ factors can be expressed in terms of one single factor $(1+\lambda^{q_0})$. For example $(1+\lambda)$, $(1+\lambda^2)$, $(1+\lambda^8)$ are all powers of $(1+\lambda)$. In this case the total delay around the generator is a known multiple of the basic delay introduced by $(1+\lambda^{q_0})$, and must add up to one full period of the generator. Thus one may compute the delay between each pair of bits.

III. Random Number Generators

Before constructing a PRNG based on the algorithm of Section II one must decide on the number of bits of resolution, and the period or equivalently the equidistribution properties required. These fix the constants m and n in Eq. (2.14), but not the q's. They are determined by the requirement that $f(\lambda) = \det F(\lambda)$ be a primitive polynomial, and questions of layout. The first requirement is easier to satisfy than one might think. Whereas there are a fairly small number of trinomials to a given order, there are a huge number of polynomials which can be written in the form (2.14). A given polynomial can be easily tested for the property of being primitive. The method used here [7] requires an amount of time which is cubic in the order of the polynomial. First one constructs a matrix D_{ij} such that its characteristic is

$$\lambda^{2i} = \sum_{j=0}^{n-1} D_{ij}\lambda^j \qquad \text{mod} f, \quad 0 \leqslant i,j \leqslant n-1. \tag{3.1}$$

Then one computes the rank of the matrix

$$M_{ij} = \delta_{ij} + D_{ij} \qquad \text{mod } 2. \tag{3.2}$$

There is a theorem to the effect that the rank of M is n less the number of factors in the decomposition of $f(\lambda)$ into primitive factors

$$f(\lambda) = \prod_{i=1}^{\alpha} [g_i(\lambda)]^{n_i}. \tag{3.3}$$

TABLE 1

Values of n for Which $2^n - 1$ is Prime

1	31	1279	9941
2	61	2203	11213
3	89	2281	-
5	107	3217	
7	127	4253	
13	521	4423	
19	607	9689	

In the case that the rank is $n - 1$, f is a power of a primitive polynomial. If the order of f is also prime, which it is for n a Mersenne number, then $f(\lambda)$ is primitive. Because we wish to guarantee a maximal length period we will always choose n to be a Mersenne number. The known Mersenne numbers are listed in Table 1.

It is instructive to work out in detail the properties of a small period generator so we can gain some confidence in what we are doing. In Table 2 we give a list of all sets of q's less than (4, 5, 5, 5), respectively, for which

$$(1 + \lambda^{q_1})(1 + \lambda^{q_2})(1 + \lambda^{q_3})(1 + \lambda^{q_4}) + \lambda^{19} \tag{3.4}$$

is primitive. The choice (2, 2, 3, 3) meets our requirements, and gives an effective FSR with characteristic polynomial

$$1 + \lambda^4 + \lambda^6 + \lambda^{10} + \lambda^{19}. \tag{3.5}$$

The full generator could be arranged as in Fig. 3. Notice that the delays between bits are, by the shortest path

$$n_{12} = 9, \quad n_{13} = 5, \quad n_{14} = 4, \quad n_{23} = 5, \quad n_{24} = 5, \quad n_{34} = 9, \tag{3.6}$$

so we expect the folowing equidistribution properties: for the most significant bit 19 distribution, or the first two bits 9 distribution, for the first three bits 5 distribution, and for the full word 4 distribution. Under the circumstances this is the best we can

TABLE 2

Generators with $m = 4$, $n = 19$

1125	2233
1255	2235
1334	2334
1335	3335
1345	3345
1455	3355

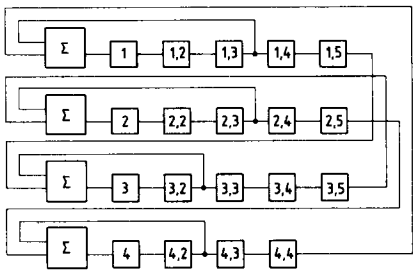

FIG. 3. A 4 bit random number generator with $m = 4$, $q = 2, 2, 3, 3$, $n = 19$.

hope to achieve. No other wiring arrangement could do better, but some could do worse. The period of the generator is $2^n - 1 = 524287$ which is easily accessible to simulation of a full period, and the above expectations are indeed fulfilled. Further, we can compute the delays of the four bits from each other in the sequence of the effective FSR. And discover that their relative positions are

$$0, \quad 262144, \quad 59298, \quad 321442, \tag{3.7}$$

which are reasonably distributed in the full period. We have searched for other generators with wider words and longer periods. It is impractical to try to generate or list all polynomials of a given order so we have adopted the strategy of searching at random for sets of q's which generate primitive effective FSRs. We select the q's normally distributed with a small width around an average of which is about $n/2m$ or half the length of a row. In Tables 2 through 8 we give generators for $m = 4$, 8, 12, 16, 24, and 32 each for one or two different n's.

We have studied several of the longer period generators with the standard heuristic tests and they perform within expectations. There is a problem of a subtle nature in that for long enough period these generators become essentially untestable. One must have a theoretical understanding of how they work or there is no chance of being sure that they reach the full period and distribution properties claimed.

TABLE 3

Randomly Selected Generators with $m = 4$, $n = 31$

1122	2237	2355
1155	2246	2457
1337	2247	2556
1357	2267	3344
1467	2333	3366
1557	2345	3444

TABLE 4

Randomly Selected Generators with $m = 8$, $n = 61$

11122455	13345668	24456789
11124555	13445567	33344555
11344556	144668810	34455569
11345666	14566777	34667779
12334567	23344458	45667888
12334579	23345566	
13344447	24444456	

TABLE 5

Randomly Selected Generators with $m = 12$, $n = 61$[a]

$1_8 3_2 4_2$	$1_4 2\ 3_2 4_4 5$
$1_7 3_2 4_3$	$1_3 2_6 3_2 4$
$1_6 2_3 3_2 4$	$1_3 2_3 3\ 4_4 5$
$1_5 2_3 3_2 4_2$	$1\ 2_4 3_3 4\ 6_2 7$

[a] Subscripts represents repeat factors.

TABLE 6

Randomly Selected Generators with $m = 12$, $n = 127$

$2\ 5\ 6_4 7_2 8_2 10_2$	$4\ 5\ 6\ 7_2 8_3 9_3 13$	$5\ 6_2 7_5 8_2 9\ 11$
$4_2 5_2 6_2 7_4 8\ 9$	$5_3 6_2 7_2 9\ 10_2$	$6_2 8_6 9_2 10\ 11$
$4_2 5_2 6_2 7_2 8\ 9_2 11$	$5_2 6_2 7\ 8_4 9\ 10\ 11$	$6\ 7_2 8_4 9_2 12_3$
$4\ 5\ 6_2 7\ 8_3 9\ 10\ 11_2$	$5_2 6\ 7_2 8_2 9_3 12_2$	$7_2 8_7 10_2 12$

TABLE 7

Randomly Selected Generators $m = 16$, $n = 127$

$1\ 2_2 3\ 4\ 5\ 6_3 7_6 9$	$2_2 3\ 4_3 5\ 6_5 7\ 8_2 9$	$4_5 5_4 6_4 7\ 8\ 11$
$1\ 2\ 3_3 4_2 5_5 6_2 7_2$	$2\ 3_2 4\ 5_4 6_3 7_3 10_2$	$4_3 5_8 6_3 8\ 9$
$1\ 3\ 4_3 5_3 6_2 7_2 8_2 9\ 10$	$2\ 3\ 4_4 5\ 6_2 7_3 8_2 9_2$	
$1\ 4_4 6_4 7_4 9_2 10$	$2\ 3\ 4_2 5_2 6_2 7_5 8_2 11$	
$1\ 4_2 5_3 6_5 7\ 8_2 9_2$	$3_3 4_3 5_2 6_4 7_3 8$	
$1\ 4\ 5_2 6_7 7\ 8_3 9$	$3_3 4\ 5_3 6_4 10$	
$2_2 3_3 4_4 6_2 7$	$3\ 4_2 5_4 6_5 7_2 8\ 9_2 10$	

TABLE 8

Randomly Selected Generators for $n = 521$

$19\ 21_2 22\ 23_2 24_5 25\ 26_3 27$	$m = 16$
$21\ 22_2 23_4 24_3 25_4 26_2$	$m = 16$
$22_2 23_3 24_6 25_3 26_2$	$m = 16$
$11\ 12_2 13_4 14_2 16_3 17_2 18_3 20_2 21$	$m = 24$
$12_4 13_3 14_4 15_4 16_4 17_3 18\ 20$	$m = 24$
$8\ 9_5 10_6 11_4 12_7 13_5 14\ 15_2 16$	$m = 32$
$9\ 10_5 11_7 12_5 13_7 14_2 15_3 16_2$	$m = 32$

ACKNOWLEDGMENTS

The author would like to thank J. Richardson and C. L. Seitz for very useful discussions and Cal Tech computer science for their hospitality and support while part of this research was carried out.

REFERENCES

1. K. BINDER, "Monte-Carlo Methods" (K. Binder, Ed.), Springer-Verlag, Berlin, 1979.
2. R. PEARSON, J. RICHARDSON, R. SUAYA, AND D. TOUSSAINT, in preparation.
3. R. C. TAUSWROTH, *Math Comput.* **19** (1965), 201.
4. T. G. LEWIS AND W. H. PAYNE, *J. Assoc. Comput. Mach.* **20** (1973), 456.
5. S. W. GOLOMB, "Shift Register Sequenes," Holden–Day, San Francisco, 1967.
6. J. P. R. TOOTHILL, W. D. ROBINSON, AND D. J. EAGLE, *J. Assoc. Comput. Mach.* **20** (1973), 469.
7. E. R. BERLEKAMP, "Algebraic Coding Theory," McGraw–Hill, New York, 1968.