JOURNAL OF ALGORITHMS 13, 527–545 (1992)

How to Predict Congruential Generators

HUGO KRAWCZYK*

Computer Science Department, Technion, Haifa, Israel

Received September 1987; revised May 1991

In this paper we show how to predict a large class of pseudorandom number generators. We consider congruential generators which output a sequence of integers $s_0, s_1, \ldots,$ where s_i is computed by the recurrence $s_i \equiv \sum_{j=1}^{k} \alpha_j \Phi_j(s_0, s_1, \ldots, s_{i-1}) \pmod{m}$ for integers m and α_j, and integer-valued functions Φ_j, $j = 1, \ldots, k$. The predictors know the functions Φ_j in advance and have access to the elements of the sequence prior to the element being predicted, but they do not know the modulus m or the coefficients α_j with which the generator actually works. We prove that both the number of mistakes made by the predictors and the time complexity of each prediction are bounded by a polynomial in k and $\log m$, provided that the functions Φ_j are computable (over the integers) in polynomial time. This extends previous results about the predictability of such generators. In particular, we prove that multivariate polynomial generators, i.e., generators where $s_i \equiv P(s_{i-n}, \ldots, s_{i-1}) \pmod{m}$, for a polynomial P of known degree in n variables, are efficiently predictable. © 1992 Academic Press, Inc.

1. INTRODUCTION

A *number generator* is a deterministic algorithm that given a sequence of initial values outputs an (infinite) sequence of numbers. Some generators, called *pseudorandom number generators*, are intended to output sequences of numbers having some properties encountered in truly random sequences. Such generators appear in diverse applications as probabilistic algorithms, Monte Carlo simulations, cryptography, etc. For cryptographic applications a crucial property for the sequences generated is their *unpredictability*. That is, the next element generated should not be efficiently predictable, even given the entire past sequence.

We follow the approach of [3], in which the predicting scenario is viewed as a game between the predictor and the generator. For each

*This research was supported by Grant No. 86-00301 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel. Current address: IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

0196-6774/92 $5.00

element in the sequence the predictor outputs its guess and the generator subsequently responds with the true value of this element. The predictor outputs each guess after having seen all the preceding elements in the sequence. The efficiency of the predicting algorithm is measured by both the number of prediction mistakes (i.e., those guesses that do not correspond to the actual element being predicted) and the prediction time (i.e., the time it takes to compute each guess).

A pseudorandom number generator that has received much attention is the so-called *linear congruential generator*, an algorithm that on input integers a, b, m, s_0 outputs a sequence $s_1, s_2, \ldots,$ where

$$s_i \equiv as_{i-1} + b \pmod{m}.$$

These generators and the statistical properties of sequences produced by them have been extensively studied by many authors (cf. [13, 16]).

Boyar [17] proved that linear congruential generators are efficiently predictable even when the coefficients and the modulus are unknown to the predictor. She showed an upper bound of $O(\log m)$ on the number of mistakes made by the predictor and that the prediction time is bounded by a polynomial in $\log m$. Later, Boyar [3] extended her own method, proving the predictability of a large family of generators. She considered *general congruential generators*, where the element s_i is computed as

$$s_i \equiv \sum_{j=1}^{k} \alpha_j \Phi_j(s_0, s_1, \ldots, s_{i-1}) \pmod{m} \tag{1}$$

for integers m and α_j, and integer-valued functions Φ_j, $j = 1, \ldots, k$. She showed that these sequences can be predicted, for some class of functions Φ_j, by a predictor knowing these functions in advance, but not given the coefficients α_j or the modulus m. A predictor for congruential generators is considered as *efficient* if the number of mistakes and the prediction time are both bounded by a polynomial in k and $\log m$.

Boyar's method requires that the functions Φ_j have the *unique extrapolation property*. The functions $\Phi_1, \Phi_2, \ldots, \Phi_k$ have the *unique extrapolation property with extrapolation length* r, if for every pair of generators working with the above set of functions, the same modulus m and the same initial values, if both generators coincide in the first r values generated, then they output the same infinite sequence. Note that these generators need not be identical (i.e., they may have different coefficients). The number of mistakes made by Boyar's predictors depends on the extrapolation length. Her method yields efficient predictors provided that the functions Φ_j have a *small* extrapolation length. The linear congruential generator is an example of a generator having the extrapolation property (with length 2). Boyar also proved this property for two extensions of the linear congruential generator. Namely, the generators in which the element s_i satisfies the recurrence

$$s_i \equiv \alpha_1 s_{i-k} + \cdots + \alpha_k s_{i-1} \pmod m$$

and those for which

$$s_i \equiv \alpha_1 s_{i-1}^2 + \alpha_2 s_{i-1} + \alpha_3 \pmod m.$$

The first case with length $k + 1$, the second with length 3. She also conjectured the predictability of generators having a polynomial recurrence

$$s_i \equiv p(s_{i-1}) \pmod m$$

for an unknown polynomial p of known degree.

A natural generalization of the above examples is a generator having a *multivariate polynomial recurrence*, that is, a generator outputting a sequence $s_0, s_1, \dots,$ where

$$s_i \equiv P(s_{i-n}, \dots, s_{i-1}) \pmod m$$

for a polynomial P in n variables. Polynomials P of total degree at most d and n variables are a special case of general congruential generator, where the Φ_j run over all monomials of total degree $\leq d$ and $k = \binom{n+d}{d}$. Lagarias and Reeds [15] showed that multivariate polynomial recurrences have the unique extrapolation property. Furthermore, for the case of a one-variable polynomial of degree d, they proved this property with length $d + 1$, thus settling Boyar's conjecture concerning the efficient predictability of such generators. However, for the general case they did not give an effectively computable bound for the extrapolation length of such recurrences. In particular, the extrapolation length is not known to be polynomial in $\binom{n+d}{d}$, so that at present the Boyar method is not known to be an efficient method for predicting multivariate polynomial recurrences. (It yields an efficient predictor if we restrict ourselves to the prediction of polynomials of fixed degree of a fixed number of variables, in which case the extrapolation length can be viewed as a constant.)

In this paper we give a prediction method that is valid for any general congruential generator, i.e., any generator of the form (1). The only restriction on the functions Φ_j is that they are computable in polynomial time when working over the integers. This condition is necessary to guarantee the efficiency of our method. (The same is required in Boyar's method.) Thus, we remove the necessity of the unique extrapolation property and extend the predictability results to a large class of generators. In particular, we show that multivariate polynomial recurrence generators *are efficiently predictable.*

Our predicting technique uses ideas from Boyar's method, but our approach to the prediction problem is somewhat different. Boyar's method tries to simulate the generator by "discovering" its secrets: the modulus m and the coefficients α_j that the generator works with. Instead, our algorithm uses only the knowledge that these coefficients exist, but does not try to find them. Some algebraic techniques introduced by Boyar when computing over the integers are extended here to work also when computing over the ring of integers modulo m. A key technical concept introduced in the notion of *weak linear independence* of an ordered set of vectors over a ring R, which functions as a substitute for linear independence over rings R having zero-divisors. We show for $R = Z_n$ the bound $k \log_q n$ for the maximal number of weakly independent elements in R^k, where q is the smallest prime divisor of n (see Section 3.2).

2. Definitions and Notation

Definition. A *number generator* is an algorithm that given n_0 integers, called the *initial values* and denoted $s_{-n_0}, \ldots, s_{-1}$, outputs an infinite sequence of integers $s_0, s_1, \ldots,$ where each element s_i is computed deterministically from the previous elements, including the initial values.

For example, a generator of the form $s_i \equiv \alpha_1 s_{i-k} + \cdots + \alpha_k s_{i-1}$ $(\text{mod } m)$ requires a set of k initial values to begin computing the first elements $s_0, s_1, \ldots$ of the sequence. Thus, for this example, $n_0 = k$.

Definition. A (*general*) *congruential generator* is a number generator for which the ith element of the sequence is a $\{0, \ldots, m-1\}$-valued number computed by the congruence

$$s_i \equiv \sum_{j=1}^{k} \alpha_j \Phi_j(s_{-n_0}, \ldots, s_{-1}, s_0, \ldots, s_{i-1}) \ (\text{mod } m),$$

where α_j and m are arbitrary integers and Φ_j, $1 \le j \le k$, is an integer-valued function. For a given set of k functions $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_k\}$ a congruential generator working with these functions (and arbitrary coefficients and modulus) will be called a *Φ-generator*.

Example. Consider a generator which outputs a sequence defined by a multivariate polynomial recurrence, i.e., $s_i \equiv P(s_{i-n}, \ldots, s_{i-1})$ (mod m), where P is a polynomial in n variables and degree d. Such a generator is a Φ-generator in which each function Φ_j represents a monomial in P, and the α_j's are the corresponding coefficients. In this case we have $k = \binom{n+d}{d}$, and the functions (monomials) Φ_j are applied to the last n elements in the sequence.

Note that in the above general definition, the functions Φ_j work on sequences of elements, so the number of arguments for these functions may be variable. Some matrix notation will be more convenient.

Notation. $s(i)$ will denote the *vector* of elements (including the initial values) until the element s_i, i.e.,

$$s(i) = (s_{-n_0}, \dots, s_{-1}, s_0, \dots, s_i), \qquad i = -1, 0, 1, 2, \dots .$$

Thus, $\Phi_j(s_{-n_0}, \dots, s_{-1}, s_0, \dots, s_{i-1})$ will be written as $\Phi_j(s(i-1))$.

Let α denote the vector $(\alpha_1, \alpha_2, \dots, \alpha_k)$ and B_i, $i \geq 0$, denote the column vector

$$B_i = \begin{bmatrix} \Phi_1(s(i-1)) \\ \Phi_2(s(i-1)) \\ \vdots \\ \Phi_k(s(i-1)) \end{bmatrix}.$$

Then we can rewrite the Φ-generator's recurrence as

$$s_i \equiv \alpha \cdot B_i \pmod{m}. \tag{2}$$

Here, and in the sequel, $\cdot$ denotes matrix multiplication.

Finally, $B(i)$ will denote the matrix

$$B(i) = \begin{bmatrix} B_0 & B_1 & \cdots & B_i \end{bmatrix}.$$

For complexity considerations we refer to the *size* of the prediction problem as given by the binary length of the modulus m and the number k of coefficients with which the generator actually works. (Note that the k coefficients as well as the elements output by the generator have size at most $\log m$.) The efficiency of a predictor will be measured in terms of these parameters, which can be seen as measuring the amount of information hidden from the predictor.

Definition. A *predictor* for a Φ-generator is an algorithm that interacts with the Φ-generator in the following way. The predictor obtains as input the initial values with which the generator is working. For $i = 0, 1, 2, \ldots$ the predictor outputs its prediction for the element s_i and the generator responds with the true value of s_i.

An *efficient predictor for a family of congruential generators* is an algorithm which given a set of k functions $\mathbf{\Phi} = \{\Phi_1, \Phi_2, \ldots, \Phi_k\}$ corresponding to a Φ-generator in the family, behaves as a predictor for this Φ-generator, and there exist polynomials P and Q for which

(1) the computation time of every prediction is bounded by $P(k, \log m)$

(2) the number of prediction mistakes is bounded by $Q(k, \log m)$.

We call such a family *efficiently predictable*.

In the above definition we may consider the functions Φ_j as given to the algorithm by means of "black-boxes" or oracles to these functions. In this case the output of such an oracle is the integer value of the function before it is reduced according to the secret modulus.

Observe that when computing its prediction for s_i the predictor has seen the entire segment of the sequence before s_i, and the initial values. The only secret information kept by the generator is the coefficients and the modulus. If the predictor is not given the initial values then our method cannot be applied to *arbitrary* Φ-generators. However, in typical cases (including the multivariate polynomial recurrence), generators have recurrences depending only on the last n_0 elements, for some number n_0. In this case the predictor may consider the first n_0 elements generated as initial values and begin predicting after the generator outputs them.

Finally, we introduce the following notion concerning the complexity of the functions Φ_j, when acting on the vectors $s(i)$, but computed over the integers (and not reduced modulo m). This will be referred to as the *non-reduced complexity* of the functions Φ_j. The performance of our predicting algorithm will depend on this complexity.

DEFINITION. Φ-generators having non-reduced time-complexity polynomial in $\log m$ and k are called *non-reduced polynomial-time Φ-generators*.

3. The Predicting Method

The predictor tries to infer the element s_i for knowledge of all the previous elements of the sequence, including the initial values. It does not know the modulus m with which the generator is working, so it uses different estimates for this m. Its first estimate is $\hat{m} = \infty$, i.e., the predictor begins by computing over the integers. After some portion of the sequence is revealed and taking advantage of possible prediction mistakes, a new (finite) estimate $\hat{m}_0$ for m is computed. Later on, new values for $\hat{m}$ are computed in such a way that each $\hat{m}$ is a (non-trivial) divisor of the former estimate and all are multiples of the actual m. Eventually $\hat{m}$ may reach the true value of m. (For degenerate cases, like a generator producing a constant sequence, it may happen that m will never be reached but this will not affect the prediction capabilities of the algorithm.)

We shall divide the predicting algorithm into two *stages*. The first stage is when working over the integers, i.e., $\hat{m} = \infty$. The second one is after the first finite estimate $\hat{m}_0$ was computed. The distinction between these two stages is not essential, but some technical reasons make it convenient. In fact, the algorithm is very similar for both stages.

The idea behind the algorithm is to find linear dependencies among the columns of the matrix $B(i)$ and to use these dependencies in making the prediction of the next element s_i. More specifically, we try to find a representation of B_i as a linear combination (modulo the current $\hat{m}$) of the previous B_j's (that are known to the predictor at this time). If such a combination exists, we apply it to the previous elements in the sequence (i.e., previous s_j's) to obtain our *prediction* for s_i. If not correct, we made a mistake but gain information that allows us to refine the modulus $\hat{m}$. A combination as above will not exist if B_i is independent of the previous columns. We show that under a suitable definition of independence, called *weak independence*, the number of possible *weakly independent* B_i's cannot be *too large*. Therefore only a *small* number of mistakes is possible, allowing us to prove the efficiency of the predictor.

The number of mistakes made by the predictor, until it is able to refine the current $\hat{m}$, will be bounded by a polynomial in the size of this $\hat{m}$. Also the total number of distinct moduli $\hat{m}$ computed during the algorithm is bounded by the size of the first (finite) $\hat{m}_0$. Thus, the total number of possible mistakes is polynomial in this size, which in turn is determined by the length of the output of the non-reduced functions Φ_j. This is the reason for which the non-reduced complexity of these functions is required to be polynomial in the size of the true m and k. In this case the total number of mistakes made by the predictor will also be polynomial in these parameters. The same is true for the computation time of every prediction.

The algorithm presented here is closely related to Boyar's [3]. Our first stage is exactly the same as the first stage there. That is, the two algorithms begin by computing a multiple of the modulus m. Once this is accomplished, Boyar's strategy is to find a set of coefficients $\{\alpha'_j\}_{j=1}^{k}$ and a sequence of moduli $\hat{m}$ which are refined during the algorithm until no more mistakes are made. For proving the correctness and efficiency of her predictor, it is required that the generator satisfies the *unique extrapolation property* (mentioned in the Introduction). Here, we do not try to find the coefficients. Instead, we extend the ideas of the first stage and apply them also in the second stage. In this way the need for an extrapolation property is avoided, allowing the extensions of the predictability results.

3.1 *First Stage*

Let us describe how the predictor computes its prediction for s_i. At this point the predictor knows the whole sequence before s_i, i.e., $s(i-1)$, and so far it has failed to compute a finite multiple of the modulus m, so it is still working over the integers. In fact, the predictor is able at this point to compute all the vectors $B_0, B_1, \dots, B_i$, since they depend only on $s(i-1)$. Moreover, our predictor keeps at this point, a submatrix of $B(i-1)$, denoted by $B(i-1)$, of linearly independent (over the rationals) columns. (For every i, when predicting the element s_i, the predictor checks if the column B_i is independent of the previous ones. If this is the case then B_i is added to $\overline{B(i-1)}$ to form $\overline{B(i)}$.) Finally, let us denote by $\overline{s(i-1)}$ the corresponding *subvector* of $s(i-1)$, having the entries indexed with the same indices appearing in $\overline{B(i-1)}$.

Prediction of s_i in the First Stage. The predictor begins by computing the (column) vector B_i. Then, it solves, **over the rationals**, the system of equations

$$\overline{B(i-1)} \cdot x = B_i.$$

If no solution exists, B_i is independent of the columns in $\overline{B(i-1)}$ so it sets

$$\overline{B(i)} = \left[\overline{B(i-1)}\, B_i\right]$$

and it fails to predict s_i.

If a solution exists, let c denote the solution (vector) computed by the predictor. The prediction for s_i, denoted $\hat{s}_i$, will be

$$\hat{s}_i = \overline{s(i-1)} \cdot c.$$

The predictor, once having received the true value for s_i, checks whether this prediction is correct or not (observe that the prediction $\hat{s}_i$ as computed above may not even be an integer). If correct, it has succeeded and goes on predicting s_{i+1}. If not, i.e., $\hat{s}_i \neq s_i$, the predictor has made a mistake, but now it is able to compute $\hat{m}_0 \neq \infty$, the first multiple of the modulus m, as follows. Let l be the number of columns in matrix $\overline{B(i-1)}$

and let the solution c be

$$c = \begin{bmatrix} c_1/d_1 \\ c_2/d_2 \\ \vdots \\ c_l/d_l \end{bmatrix}.$$

Now, let d denote the least common multiple of the denominators in these fractions, i.e., $d = \operatorname{lcm}(d_1, \dots, d_l)$. The value of $\hat{m}_0$ is computed as

$$\hat{m}_0 = |d\hat{s}_i - ds_i|.$$

Observe that $\hat{m}_0$ is an integer, even if $\hat{s}_i$ is not. Moreover, this integer is a multiple of the true modulus m with which the generator is working (see Lemma 1 below).

Once $\hat{m}_0$ is computed, the predictor can begin working modulo this $\hat{m}_0$. So the first stage of the algorithm is terminated and it goes on into the second one.

The main facts concerning the performance of the predicting algorithm during the first stage are summarized in the next lemma.

LEMMA 1. (a) *The number $\hat{m}_0$ computed at the end of the first stage is a nonzero multiple of the modulus m.*

(b) *The number of mistakes made by the predictor in the first stage is at most $k + 1$.*

(c) *For non-reduced polynomial time Φ-generators, the prediction time for such s_i during the first stage is polynomial in* $\log m$ *and k.*

(d) *For non-reduced polynomial time Φ-generators, the size of $\hat{m}_0$ is polynomial in* $\log m$ *and k. More precisely, let M be an upper bound on the output of each of the functions Φ_j, $j = 1, \dots, k$, working on $\{0, \dots, m-1\}$-valued integers. Then, $\hat{m}_0 \leq (k+1)k^{k/2}mM^k$.*

Proof. (a) From the definition of the generator we have the congruence $s_j \equiv \alpha \cdot B_j \pmod{m}$ for all $j \geq 0$; therefore,

$$\overline{s(i-1)} \equiv \alpha \cdot \overline{B(i-1)} \pmod{m}. \tag{3}$$

Thus,

$$\begin{aligned} d\hat{s}_i &= d\overline{s(i-1)} \cdot c && \text{(by definition of } \hat{s}_i\text{)} \\ &\equiv d\alpha \cdot \overline{B(i-1)} \cdot c \pmod{m} && \text{(by (3))} \\ &= d\alpha \cdot B_i && (c \text{ is a solution to } \overline{B(i-1)} \cdot x = B_i) \\ &\equiv ds_i \pmod{m} && \text{(by definition of } s_i(2)\text{)}. \end{aligned}$$

So we have shown that $d\hat{s}_i \equiv ds_i \pmod{m}$. Observe that it cannot be the case that $d\hat{s}_i = ds_i$, because this implies $\hat{s}_i = s_i$, contradicting the incorrectness of the prediction. Thus, we have proved that $\hat{m}_0 = |d\hat{s}_i - ds_i|$ is indeed a nonzero multiple of m.

(b) The possible mistakes in the first stage are when a rational solution to the system of equations $\overline{B(i-1)} \cdot x = B_i$ does not exist, or when such a solution exists but our prediction is incorrect. The last case will happen only once because after that occurs the predictor goes into the second stage. The first case cannot occur "too much." Observe that the matrices $B(j)$ have k rows, thus the maximal number of independent columns (over the rationals) is at most k. So the maximal number of mistakes made by the predictor in the first stage is $k + 1$.

(c) The computation time for the prediction of s_i is essentially given by the time spent computing B_i and solving the above equations. The functions Φ_j are computable in time polynomial in $\log m$ and k, so the computation of the vector B_i is also polynomial in $\log m$ and k. The complexity of solving the system of equations, over the rationals, is polynomial in k and in the size of the entries of $\overline{B(i-1)}$ and B_i (see [8, 19, Chap. 3]). These entries are determined by the output of the (nonreduced) functions Φ_j, and therefore their size is bounded by a polynomial in $\log m$ and k. Thus, the total complexity of the prediction step is polynomial in $\log m$ and k, as required.

(d) As pointed out in the proof of claim (c), a solution to the system of equations in the algorithm can be found in time bounded polynomially in $\log m$ and k. In particular, this guarantees that the *size* of the solution will be polynomial in $\log m$ and k. (By size we mean the size of the denominators and numerators in the entries of the solution vector.) Clearly, by the definition of $\hat{m}_0$, the polynomiality of the size of the solution c implies that the size of $\hat{m}_0$ is itself polynomial in $\log m$ and k.

The explicit bound on $\hat{m}_0$ can be derived as follows. Using Cramer's rule we obtain that the solution c to the system $\overline{B(i-1)} \cdot x = B_i$, can be represented as $c = (c_1/d, \ldots, c_l/d)$, where each c_j and d are determinants of l by l submatrices in the above system of equations. Let D be the maximal possible value of a determinant of such a matrix. We have that $d\hat{s}_i = d\overline{s(i-1)}c \leq lmD$ (here m is a bound on $\overline{s(i-1)}$ entries) and $ds_i \leq mD$, then $\hat{m}_0 = |d\hat{s}_i - ds_i| \leq (l+1)mD$. In order to bound D we use Hadamard's inequality which states that each n by n matrix $A = (a_{ij})$ satisfies $\det(A) \leq \prod_{i=1}^{n}(\sum_{j=1}^{n} a_{ij}^2)^{1/2}$. In our case the matrices are of order l by l, and the entries to the system are bounded by M (the bound on Φ_j output). Thus, $D \leq \prod_{i=1}^{l}(\sum_{j=1}^{l} M^2)^{1/2} = (lM^2)^{l/2}$, and we obtain

$$\hat{m}_0 \leq (l+1)mD \leq (l+1)m(lM^2)^{l/2} \leq (k+1)k^{k/2}mM^k.$$

The last inequality follows since $l \leq k$. □

3.2 *Second Stage*

After having computed $\hat{m}_0$, the first multiple of m, we proceed to predict the next elements of the sequence, but now we are working modulo a finite $\hat{m}$. The prediction step is very similar to the one described for the first stage. The differences are those that arise from the fact that the computations are modulo an integer and not necessarily over a field. In particular, the equations to be solved are considered in the ring of residues modulo $\hat{m}$. Let us denote the ring of residues modulo n by Z_n. In the following definition we extend the concept of linear dependence to these rings.

DEFINITION. Let $\nu_1, \nu_2, \ldots, \nu_l$ be a sequence of l vectors with k entries from Z_n. We say that this sequence is *weakly linearly dependent* mod n if $\nu_1 = 0$ or there exists an index i, $2 \leq i \leq l$, and elements $c_1, c_2, \ldots, c_{i-1} \in Z_n$, such that $\nu_i \equiv c_1\nu_1 + c_2\nu_2 + \cdots + c_{i-1}\nu_{i-1} \pmod{n}$. Otherwise, we say that the sequence is *weakly linearly independent*.

Note that the order here is important. Unlike the case in the traditional definition over a field, in the above definition it is *not* equivalent to say that *some* vector in the set can be written as a linear combination of the *others*. Another important difference is that it is not true, in general, that $k+1$ vectors of k components over Z_n must contain a weakly dependent vector. Fortunately, we have the following upper bound on the size of a weakly linearly independent set.

THEOREM 2. *Let $\nu_1, \nu_2, \ldots, \nu_l$ be a sequence of k-dimensional vectors over Z_n. If the sequence is weakly linearly independent* mod n, *then $l \leq k \log_q n$, where q is the smallest prime dividing n.*

Proof. Let $\nu_1, \nu_2, \ldots, \nu_l$ be a sequence of l vectors from Z_n^k and suppose this sequence is weakly linearly independent mod n. Consider the set

$$V = \left\{ \sum_{i=1}^{l} c_i v_i \;(\operatorname{mod} n) \colon c_i \in \{0, 1, \ldots, q-1\} \right\}.$$

We shall show that this set contains q^l different vectors. Equivalently, we show that no two (different) combinations in V yield the same vector.

Claim. For every c_i, $c'_i \in \{0, 1, \ldots, q-1\}$, $1 \leq i \leq l$, if $\Sigma_{i=1}^{l} c_i v_i \equiv \Sigma_{i=1}^{l} c'_i v_i$ (mod m) then $c_i = c'_i$ for $i = 1, 2, \ldots, l$.

Suppose this is not true. Then we have $\Sigma_{i=1}^{l}(c_i - c'_i)v_i \equiv 0$ (mod n). Denote $c_i - c'_i$ by d_i. Let t be the maximal index for which $d_t \neq 0$. This number d_t satisfies $-q < d_t < q$, so it has an inverse modulo n (recall that q is the least prime divisor of n), denoted d_t^{-1}. It follows that $v_t \equiv \Sigma_{i=1}^{t-1} - d_t^{-1} d_i v_i$ (mod n), contradicting the independence of v_t and thus proving the claim.

Hence, $|V| = q^l$ and, therefore,

$$q^l = |V| \leq |Z_n^k| = n^k$$

which implies that $l \leq k \log_q n$, proving the theorem. □

With the above definition of independence in mind, we define the matrix $\overline{B(i)}$ as a submatrix of $B(i)$ containing all the columns of $B(i)$ that are weakly linearly independent mod $\hat{m}$ of the previous columns. Note that $\hat{m}$ will have distinct values during the algorithm, so when writing $\overline{B(i)}$ we shall refer to its value modulo the current $\hat{m}$.

PREDICTION OF s_i IN THE SECOND STAGE. Let us describe the prediction step for s_i when working modulo $\hat{m}$. It suffices to point out the differences with the process described for the first stage. As before, we begin by computing the vector B_i (now reduced modulo $\hat{m}$) and solving the system of equations

$$\overline{B(i-1)} \cdot x \equiv B_i \;(\operatorname{mod} \hat{m}).$$

We stress that this time we are looking for a solution over $Z_{\hat{m}}$. In case a solution does not exist, we fail to predict, exactly as in the previous case. If a solution does not exist, B_i is linearly weakly independent of $\overline{B(i-1)}$ and we add it to $\overline{B(i-1)}$ to form the matrix $\overline{B(i)}$. If a solution does exist, we output our prediction, computed as before, but the result is reduced mod $\hat{m}$. Namely, we set $\hat{s}_i = \overline{s(i-1)} \cdot c$ (mod $\hat{m}$), where c is a solution to the above system of modular equations. If the prediction is correct, we proceed to predict the next element s_{i+1}. If not, we take advantage of this error to update $\hat{m}$. This is done by computing

$$m' = \gcd(\hat{m}, \hat{s}_i - s_i).$$

This m' will be the new $\hat{m}$ with which we shall work in the coming predictions.

To show that the prediction algorithm as described here is indeed an *efficient predictor*, we prove the following facts.

LEMMA 3. *The following claims hold for the above predictor*:

(a) *The number m' computed above is a nontrivial divisor of $\hat{m}$ and a multiple of the modulus m.*

(b) *Let $\hat{m}_0$ be the modulus computed at the end of the first stage. The total number of mistakes made by the predictor during the second stage is bounded by $(k + 1)\log \hat{m}_0$, and thus it is polynomial in* $\log m$ *and k for a non-reduced polynomial time Φ-generator.*

(c) *For non-reduced polynomial time Φ-generators, the prediction time for each s_i during the second stage is polynomial in* $\log m$ *and k.*

Proof. (a) Recall that $m' = \gcd(\hat{m}, \hat{s}_i - s_i)$, so it is a divisor of $\hat{m}$. It is a nontrivial divisor because $\hat{s}_i$ and s_i are reduced mod $\hat{m}$ and m, respectively, and thus the absolute value of their difference is strictly less than $\hat{m}$. It cannot be zero because $\hat{s}_i \neq s_i$, as follows from the incorrectness of the prediction. The proof that m' is a multiple of m is similar to that of claim (a) of Lemma 1. It is sufficient to show that $\hat{s}_i - s_i$ is a multiple of m, since $\hat{m}$ is itself a multiple of m. We show this by proving $\hat{s}_i \equiv s_i$ (mod m):

$$\begin{aligned}
\hat{s}_i &\equiv \overline{s(i-1)} \cdot c \pmod{\hat{m}} && \text{(by definition of } \hat{s}_i\text{)}\\
&\equiv \alpha \cdot \overline{B(i-1)} \cdot c \pmod{m} && \text{(by (3))}\\
&\equiv \alpha \cdot B_i \pmod{\hat{m}} && \big(c \text{ is a solution to}\\
& && \quad \overline{B(i-1)} \cdot x \equiv B_i \pmod{\hat{m}}\big)\\
&\equiv s_i \pmod{m} && \big(\text{by definition of } s_i(2)\big).
\end{aligned}$$

As m divides $\hat{m}$, claim (a) follows.

(b) The possible mistakes during the second stage are of two types. Mistakes of the first type happen when a solution to the above congruential equations does not exist. This implies the weak independence modulo the current $\hat{m}$ of the corresponding B_i. In fact, this B_i is also weakly independent mod $\hat{m}_0$. This follows from the property that every $\hat{m}$ is a divisor of $\hat{m}_0$. By Theorem 2, we have that the number of weakly linearly independent vectors mod $\hat{m}_0$ is at most $k \log \hat{m}_0$. Therefore the number of mistakes due to lack of a solution is bounded by this quantity. The second type of mistake is when a solution exists but the computed prediction is incorrect. Such a mistake can occur only once per $\hat{m}$. After it occurs, a new $\hat{m}$ is computed. Thus, the total number of such mistakes is at most the number of different $\hat{m}$'s computed during the algorithm. These $\hat{m}$'s form a decreasing sequence of positive integers in which every element is a divisor of the previous one. The first (i.e., largest) element is $\hat{m}_0$ and hence the length of this sequence is at most $\log \hat{m}_0$. Consequently, the total number of mistakes during the second stage is at most $(k + 1)\log \hat{m}_0$, and by Lemma 1, claim (d) this number is polynomial in $\log m$ and k.

(c) By our assumption of the polynomiality of the functions Φ_j when working on the vectors $s(i)$, it is clear that the computation of each B_i (mod $\hat{m}$), takes time that is polynomial in $\log m$ and k. We only need to show that a solution to $\overline{B(i-1)} \cdot x \equiv B_i \pmod{\hat{m}}$ can be computed in time polynomial in $\log m$ and k. A simple method for the solution of a system of linear congruences like the above is described in [6] (and [3]). This method is based on the computation of the *Smith Normal Form* of the coefficients matrix in the system. This special matrix and the related transformation matrices can be computed in polynomial time, using an algorithm of [12]. Thus, finding a solution to the above system (or deciding that none exists) can be accomplished in time polynomial in $\log m$ and k. Therefore the whole prediction step is polynomial in these parameters. □

Combining Lemmas 1 and 3 we obtain that non-reduced polynomial-time Φ-generators are efficiently predictable.

THEOREM 4. *The predicting algorithm described above is an efficient predictor for non-reduced polynomial-time Φ-generators. The number of prediction mistakes is at most $(k+1)(\log \hat{m}_0 + 1) = O(k^2 \log(kmM))$, where $\hat{m}_0$ is the first finite modulus computed by the algorithm and M is an upper bound on the output of each of the functions Φ_j, $j = 1, \ldots, k$, working over integers in the set $\{0, \ldots, m-1\}$.*

As a special case we obtain

COROLLARY. *Every multivariate polynomial recurrence generator is efficiently predictable. The number of prediction mistakes for a polynomial recurrence in n variables and degree d is bounded by $O(k^2 \log(km^d))$, where $k = \binom{n+d}{d}$.*

Proof. A multivariate polynomial recurrence is a special case of a Φ-generator with $M < m^d$, as each monomial is of degree at most d and it is computed on integers less than m. Therefore, by Lemma 1(d) we obtain $\hat{m}_0 < (k+1)k^{k/2}m^{dk+1}$. The number k of coefficients is at most the number of possible monomials in such a polynomial recurrence which is $\binom{n+d}{d}$. The bound on the number of mistakes follows by substituting these parameters in the general bound of Theorem 4. □

Remark. Note that the number k of coefficients equals the number of possible monomials in the polynomial. For general polynomials in n variables and of degree d, this number is $\binom{n+d}{d}$. Nevertheless, if we consider special recurrences in which not every monomial is possible, e.g., $s_i \equiv \alpha_1 s_{i-n}^2 + \cdots + \alpha_n s_{i-1}^2 \pmod{m}$, then the number k may be much smaller and then a better bound on the number of mistakes for such cases is derived.

4. Vector-Valued Recurrences

The most interesting subclass of Φ-generators is the class of multivariate polynomial recurrence generators. Lagarias and Reeds [15] studied a more general case of polynomial recurrences in which a sequence of n-dimensional vectors over Z_m is generated, rather than a sequence of elements from Z_m, as in our case. These vector-valued polynomial recurrences have the form

$$\bar{s}_i \equiv \left(P_1(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n}) \pmod{m}, \ldots, P_n(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n}) \pmod{m}\right),$$

where each P_l, $1 \le l \le n$, is a polynomial in n variables and of maximal degree d. Clearly, these recurrences extend the single-valued case, since for any multivariate polynomial P which generates a sequence $\{s_i\}_{i=0}^{\infty}$ of Z_m elements, one can consider the sequence of vectors $\bar{s}_i = (s_i, s_{i-1}, \ldots, s_{i-n+1})$, where $\bar{s}_i = (P(s_{i-1}, \ldots, s_{i+n}) \pmod{m}, s_{i-1}, \ldots, s_{i-n+1})$.

Vector-valued polynomial recurrences can be generalized in terms of Φ-generators as follows. Consider n congruential generators $\Phi^{(1)}, \ldots, \Phi^{(n)}$, where $\Phi^{(l)} = \{\Phi_j^{(l)}\}_{j=1}^{k}$, and for each j, l, $\Phi_j^{(l)}$ is a function in n variables. For any set $\{\alpha_j^{(l)}: 1 \le j \le k, 1 \le l \le n\}$ of coefficients and modulus m, we define a vector-valued generator which outputs a sequence of vectors

$\bar{s}_0, \bar{s}_1, \ldots$, where each $\bar{s}_i = (\bar{s}_{i,1}, \ldots, \bar{s}_{i,n}) \in Z_m^n$ is generated by the recurrence

$$\bar{s}_i \equiv \left(\sum_{j=1}^{k} \alpha_j^{(1)} \Phi_j^{(1)}(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n}) \pmod m, \ldots, \right.$$
$$\left. \sum_{j=1}^{k} \alpha_j^{(n)} \Phi_j^{(n)}(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n}) \pmod m \right). \tag{4}$$

It is easy to see that vector-valued recurrences of the form (4) can be predicted in a similar way to the single-valued recurrences studied in the previous section. One can apply the prediction method of Section 3 to each of the "sub-generators" $\Phi_j^{(l)}$, $l = 1, \ldots, n$. Note that $\bar{s}_i$ is computed by applying the functions $\Phi_j^{(l)}$ to the vector $\bar{s}_{i-1}$ and that this $\bar{s}_{i-1}$ is *known* to the predictor at the time of computing its prediction for $\bar{s}_i$. Thus, each of the sequences $\{s_{i,l}\}_{i=0}^{\infty}$, $l = 1, \ldots, n$, are efficiently predictable and so is the whole vector sequence. The number of possible prediction errors is as the sum of possible errors in each of the sub-generators $\Phi^{(l)}$. That is, at most n times the bound of Theorem 4.

One can take advantage of the fact that the different sub-generators work with the same modulus m in order to accelerate the convergence to the true value of m. At the end of each prediction step, we have n (not necessarily different) estimates $\hat{m}^{(1)}, \ldots, \hat{m}^{(n)}$ computed by the predictors for $\Phi^{(1)}, \ldots, \Phi^{(n)}$, respectively. In the next prediction we put all the predictors to work with the same estimate $\hat{m}$ computed as $\hat{m} = \gcd(\hat{m}^{(1)}, \ldots, \hat{m}^{(n)})$. This works since each of the $\hat{m}^{(l)}$ is guaranteed to be a multiple of m (claim (a) in Lemmas 1 and 3). We obtain that the total number of mistakes (i.e., the number of vectors erroneously predicted) is bounded by $(nk + 1)(\log \hat{m}_0 + 1)$. Here nk is the dimension of the whole system of equations corresponding to the n $\Phi^{(l)}$-generators (as it is the total number of coefficients hidden from the predictor), and we can use the same bounding argument as in Lemma 3(b). Also, the bound on $\hat{m}_0$ from Lemma 1 is still valid. It does not depend on the number of sub-generators, since we predict each $\Phi^{(l)}$-generator (i.e., solve the corresponding system of equations) separately. Thus, we can restate Theorem 4 for the vector-valued case.

THEOREM 5. *Vector-valued recurrences of the form* (4) *are efficiently predictable provided that each $\Phi^{(l)}$-generator, $l = 1, \ldots, n$, has polynomial-time non-reduced complexity. The number of mistakes made by the above predicting algorithm is $O(nk^2 \log(kmM))$, where M is an upper bound on the output of each of the functions $\Phi_j^{(l)}$, $j = 1, \ldots, k$, $l = 1, \ldots, n$, working over* *integers in the set* $\{0,\ldots,m-1\}$. *In particular, for vector-valued polynomial recurrences in n variables and degree at most d the number of mistakes is* $O(nk^2\log(km^d))$, *where* $k=\binom{n+d}{d}$.

Remark. For simplicity we have restricted ourselves to the case (4) in which the sub-generators $\Phi^{(l)}$ work on the last vector $\bar{s}_{i-1}$. Clearly, our results hold for the more general case in which each of these sub-generators may depend on the whole vector sequence $\bar{s}_{-n_0},\ldots,\bar{s}_{i-1}$ output so far. In this case the number n of sub-generators does not depend on the number of arguments on which the sub-generators work, and the number of arguments does not effect the number of mistakes.

5. Concluding Remarks

Our prediction results concern number generators outputting all the bits of the generated numbers and do not apply to generators that output only parts of the numbers generated. Recent works treat the problem of predicting linear congruential generators which output only parts of the numbers generated [9, 14, 21].

A theorem by Yao [22] states that pseudorandom (bit) generators are unpredictable by polynomial-time means if and only if they pass any polynomial time statistical test. That is, predictability is a *universal statistical test* in the sense that if a generator is unpredictable, then it will pass any statistical test. Thus, a generator passing this universal test will be suitable for *any* "polynomially bounded" application. Nevertheless, for specific applications, some weaker generators may suffice. As an example, for their use in some simulation processes, all that is required from the generators is suitable distribution properties of the numbers generated. In the field of probabilistic algorithms the correctness of the algorithm is often analyzed assuming the total randomness of the coin tosses of the algorithm. However, in special cases weaker assumptions suffice. For example, Bach [2] shows that simple linear congruential generators suffice for guaranteeing the correctness and efficiency of some probabilistic algorithms, even though these generators are clearly predictable. In [7] linear congruential generators are used to "expand randomness." Their method allows the deterministic "expansion" of a truly random string into a sequence of pairwise independent pseudorandom strings.

Provable unpredictable generators exist, assuming the existence of *one-way functions* [4, 22, 10, 11]. In particular, assuming the intractability of factoring, the following pseudorandom bit generator is unpredictable [5, 1, 21]. This generator outputs a bit sequence $b_1, b_2, \ldots$, where b_i is the least significant bit of s_i, $s_i \equiv s_{i-1}^2$ (mod m), and m is the product of two large primes.

Acknowledgments

I thank Oded Goldreich for his help and guidance during the writing of this paper and for the many things I have learned from him. Also, I thank Johan Hastad for suggesting an improvement to my original bound on the number of prediction mistakes and the referees for helping to make the presentation clearer.

References

1. W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, RSA and Rabin functions: Certain parts are as hard as the whole, *SIAM J. Comput.* **17** (1988), 194–209.
2. E. Bach, Realistic analysis of some randomized algorithms, *in* "Proceedings, 19th ACM Symp. on Theory of Computing, 1987," pp. 453–461.
3. J. Boyar, Inferring sequences produced by pseudo-random number generators, *J. Assoc. Comput. Mach.* **36**, No. 1 (1989), 129–141.
4. M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* **13** (1984), 850–864.
5. L. Blum, M. Blum, and M. Shub, A simple unpredictable pseudo-random number generator, *SIAM J. Comput.* **15** (1986), 364–383.
6. A. T. Butson and B. M. Stewart, Systems of linear congruences, *Can. J. Math.* **7** (1955), 358–368.
7. B. Chor and O. Goldreich, On the power of two-point based sampling, *J. Complexity* **5** (1989), 96–106.
8. J. Edmonds, Systems of distinct representatives and linear algebra, *J. Res. Nat. Bur. Stand.* (*B*), **71** (1967), 241–245.
9. A. M. Frieze, J. Hastad, R. Kannan, J. C. Lagarias, and A. Shamir, Reconstructing truncated integer variables satisfying linear congruences, *SIAM J. Comput.* **17** (1988), 262–280.
10. O. Goldreich, H. Krawczyk, and M. Luby, On the existence of pseudorandom generators, *in* "Proceedings, 29th IEEE Symp. on Foundations of Computer Science, 1988," pp. 12–24.
11. R. Impagliazzo, L. A. Levin, and M. G. Luby, Pseudo-random generation from one-way functions, *in* "Proceedings, 21th ACM Symp. on Theory of Computing, 1989," pp. 12–24.
12. R. Kannan and A. Bachem, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM J. Comput.* **8** (1979), 499–507.
13. D. E. Knuth, "The Art of Computer Programming, Vol. 2: Seminumerical Algorithms," Addison-Wesley, Reading, MA, 1969.
14. D. E. Knuth, Deciphering a linear congruential encryption, *IEEE Trans. Inf. Theory* **IT-31** (1985), 49–52.
15. J. C. Lagarias and J. Reeds, Unique extrapolation of polynomial recurrences, *SIAM J. Comput.* **17** (1988), 342–362.
16. H. Niederreiter, Quasi-Monte Carlo methods and pseudo-random numbers, *Bull. Am. Math. Soc.* **84** (1978), 957–1041.

17. J. B. PLUMSTEAD (BOYAR), Inferring a sequence generated by a linear congruence, *in* "Proceedings, 23rd IEEE Symp. on Foundations of Computer Science, 1982," pp. 153–159.

18. J. B. PLUMSTEAD (BOYAR), "Inferring Sequences Produced by Pseudo-Random Number Generators," Ph.D. thesis, University of California, Berkeley, 1983.

19. A. SCHRIJVER, "Theory of Linear and Integer Programming," Wiley, Chichester, 1986.

20. J. STERN, Secret linear congruential generators are not cryptographically secure, *in* "Proceedings, 28th IEEE Symp. on Foundations of Computer Science, 1987."

21. U. V. VAZIRANI AND V. V. VAZIRANI, Efficient and secure pseduo-random number generation, *in* "Proceedings, 25th IEEE Symp. on Foundations of Computer Science, 1984," pp. 458–463.

22. A. C. YAO, Theory and applications of trapdoor functions, *in* "Proceedings, 23rd IEEE Symp. on Foundations of Computer Science, 1982," pp. 80–91.