University of Memphis

University of Memphis Digital Commons

Electronic Theses and Dissertations

1-1-2021

EFFICIENT COMPUTER SEARCH FOR MULTIPLE RECURSIVE GENERATORS

Kenneth Bobvah Pasiah

Follow this and additional works at: https://digitalcommons.memphis.edu/etd

Recommended Citation

Pasiah, Kenneth Bobvah, "EFFICIENT COMPUTER SEARCH FOR MULTIPLE RECURSIVE GENERATORS" (2021). *Electronic Theses and Dissertations*. 2942.
https://digitalcommons.memphis.edu/etd/2942

This Dissertation is brought to you for free and open access by University of Memphis Digital Commons. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of University of Memphis Digital Commons. For more information, please contact khggerty@memphis.edu.

EFFICIENT COMPUTER SEARCH FOR SUPER-ORDER MULTIPLE RECURSIVE GENERATORS

by

Kenneth Bobvah Pasiah

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Mathematical Sciences

The University of Memphis

May 2021

Copyright © 2021 Kenneth Bobvah Pasiah

All rights reserved

DEDICATION

I will like to dedicate this work to my late parents, Daniel Pasiah and Beatrice Liba, who passed away on September 23, 1998 and April 6, 2010, respectively.

ACKNOWLEDGMENTS

I wish to thank God Almighty for granting me good health and the necessary resources through out this program.
I am particularly grateful to my advisors Dr. Lih-Yuan Deng and Dr. Dale Bowman, for their constant support, patience with me, and guidance through out this study. They took the pains to reply to my numerous emails and concerns, and above all, provided me with the necessary resources that have enabled me to successfully accomplished this project. I cannot thank them enough. To my committee members, Dr. Su Chen and Dr. Ching-Chi Yang, I wish to thank both of them, for their time and valuable inputs. I will like to thank Mr. Eric Spangler for the tutorial sessions he had with me on how to use the High Performance Computing (`HPC`), especially at the beginning of this project.
I am very grateful to my wife, Mireille Marbu, my sons Kenneth Junior and Precious Nuvaha, and my little daughter Victory Dassi, who kept asking me whether I have gotten a job in the library, where I usually spent most of my time. My constant absence from the house was to get this job done. I am equally grateful to the following family members and friends here in the USA, for their encouragement, moral, and financial support: Anthony Pasiah, Florence Pasiah, Dr. Emilia Pasiah, and Engelbert Nwana. My numerous friends, love ones, former students, former colleagues, and the rest of the family members in Cameroon who kept encouraging me daily through phone calls, emails, and text messages are not forgotten.

ABSTRACT

Kenneth Bobvah Pasiah, Ph.D. The University of Memphis, May 2021. Efficient Computer Search for Super-order Multiple Recursive Generators. Major Professors:Dr. Lih-Yuan Deng and Dr. Dale Bowman.

Pseudo-random numbers (PRNs) are the basis for almost any statistical simulation and this depends largely on the quality of the pseudo-random number generator(PRNG) used. In this study, we used some results from number theory to propose an efficient method to accelerate the computer search of super-order maximum period multiple recursive generators (MRGs). We conduct efficient computer searches and successfully found prime modulus p, and the associated order $k, (k = 40751; k = 50551, k = 50873)$ such that $R(k, p)$ is a prime. Using these values of $k's$, together with the generalized Mersenne prime algorithm, we found and listed many efficient, portable, and super-order MRGs with period lengths of approximately $10^{380278.1}$, $10^{471730.6}$, and $10^{474729.3}$. In other words, using the generalized Mersenne prime algorithm, we extended some known results of some efficient, portable and maximum period MRGs. In particular, the DX/DL/DS/DT large order generators are extended to super-order generators. For $r \leq k$, super-order generators in MRG(k,p) are quite close to an "ideal" generator. For $r > k$, the r-dimensional points lie on a relatively small family of equidistant parallel hyperplanes in a high dimensional space. The "goodness" of these generators depend largely on the distance between these hyperplanes. For LCGs, MRGs, and other generators with lattice structures, the spectral test, which is a theoretical test that gives some measure of uniformity greater than the order k of the MRG, is the most perfect figure of merit. A drawback of the spectral test is its computational complexity. We used a simple and intuitive method that employs the LLL algorithm, to calculate the spectral test. Using this method, we extended the search for "better" DX-k-s-t farther than the known value of $k = 25013$. In particular, we searched and listed "better" super-order DX-k-s-t generators for $k = 40751, k = 50551$, and $k = 50873$.
Finally, we examined, another special class of MRGs with many nonzero terms known as the

DW-k generator. The DW-k generator's iteration can be implemented efficiently and in parallel, using a k-th order matrix congruential generator (MCG) sharing the same characteristic polynomial. We extended some known results, by searching for super-order DW-k generators using our super large k values that we obtained in this study. Using extensive computer searches, we found and listed some super-order, maximum period DW(k; A, B, C, $p = 2^{31} - v$) generators.

Contents

List of Tables

Chapter 1

Introduction

Pseudo-random numbers(PRNs) are the basis for almost any statistical simulation. These PRNs are generated using pseudo-random number generator (PRNG) or algorithmic random number generator (ARNG), which is a deterministic algorithm, thus not truly random. Pseudo-random number generators (PRNGs) play a very important part in many areas of scientific research, such as, Markov Chain Monte Carlo (MCMC) simulation, computer modeling, and secure communication. The validity of these studies depends on how good these pseudo-random numbers (PRNs) are. The goal of PRNGs in many computer applications is to produce a sequence of variates that is very hard to distinguish from a sequence of truly random numbers. Therefore, it is extremely important to design a good pseudo-random number generator(PRNG) with some desirable properties, such as: long period length, uniformity and equi-distribution in higher dimensions, efficiency, portability and good empirical performance.

Maximum period MRGs of order k have become popular PRNGs in many areas of scientific research because of its great properties of long period length, uniformity and equi-distribution over spaces up to k dimensions, efficiency, portability, and excellent empirical performances. As the order of k gets larger and larger, these nice properties get more intense. Nevertheless, finding maximum period MRGs also gets more difficult.

It is well known from the literature that checking whether an MRG has maximum period is equivalent to checking whether its characteristic polynomial (2.4) is a k-degree primitive polynomial over $\mathbb{Z}_p$. Alanen and D. E. Knuth, 1964 and D. Knuth, 1998, proposed three necessary and sufficient

conditions for a polynomial in (2.4) to be primitive. One of the major drawback of this algorithm is the difficulty of factorizing a huge number $R(k,p) = (p^k - 1)/(p-1)$, when k and p are large. There are two common ways of bypassing this difficulty, either for a given p, one can find k such that $R(k,p)$ is relatively easy to factorize or for a known prime k, one can find a prime p such that $R(k,p)$ is a prime [Generalized Merssene Prime (GMP)]. Details of the first approach can be seen in Deng and H. Q. Xu, 2005 and Deng, J.-J. H. Shiau, and Lu, 2012a.
In this study, we will focus on the second approach. That is, for a prime k, we find p for which $R(k,p) = (p^k - 1)/(p-1)$ is a prime (GMP). For a 32-bit RNG, for a prime k, we find v such that $p = 2^{31} - v$ and $R(k,p)$ are primes. With this approach, we used some results from number theory to accelerate the computer searches and we find some numbers, $k's$, for which $R(k,p)$ is a prime.
In Chapter 2, we look at some concepts in the development of PRNGs from existing literature. In Section 2.1, we review Multiple Recursive Generators (MRGs) which are natural extensions of LCGs. In Section 2.3, we discuss the equi-distribution property and the spectral test, which is a theoretical test that provides some measures of uniformity of MRGs above the dimension k. In Section 2.4, we look at the relationships between the MRGs and Matrix Congruential Generators (MCGs). We used these relationships for the efficient and parallel implementations of some MRGs that we will extend in our study. In a nutshell, in Chapter 2, we review some important literary concepts that will enable us to search for some efficient and portable super-order MRGs in this study.

In Chapter 3, we discuss methods used in searching for algorithms for super-order maximum period MRGs. The key issue is to develop a method that can find a k-th degree primitive polynomial efficiently by using our approach, as stated above. We emphasized again that, Alanen and D. E. Knuth, 1964 and D. Knuth, 1998, provided three sets of necessary and sufficient conditions for $f(x)$ as defined in (2.4) to be a primitive polynomial. This algorithm has two major shortcomings. The first is that there is no early exit strategy for non-primitive polynomials when checking the conditions, thus, a great amount of computing time would be wasted. The second shortcoming that slows down their algorithm in finding a large order MRG, involved factoring a huge number like $(p^k - 1)/(p - 1)$. This can be bypassed by solving an "easier" problem of primality, see L'Ecuyer, Blouin, and Couture, 1993. Deng, 2004, proposed an efficient search algorithm with a built-in early exit strategy that can bypass these shortcomings. Primality testing also becomes a drawback if the degree of k gets too large.

Using approaches similar to those proposed in Deng, J.-J. H. Shiau, and Lu, 2012b, we adopted an alternative approach in this study; quickly rule out, for each prime order k considered, the prime modulus p for which $R(k, p) = (p^k - 1)/(p - 1)$ has "small" (say, less than 10^{12}) prime factors. Legendre's theorem, which is a particular result from number theory, is used to accelerate the computer searches.

We successfully found prime modulus p, and the associated order k, $(k = 40751, k = 50551, k = 50873)$ such that $R(k, p)$ is a prime, after an efficient computer searched. Of course this is a great improvement from the known existing maximum k value of 25013 as reported in Deng, J.-J. H. Shiau, and Lu, 2012b. With these values of $k's$, we searched for super-order MRGs using the Generalized Mersenne prime Algorithm (*algorithm GMP*) proposed by Deng, 2004. It should be noted that not all MRGs are efficient in generating random numbers or having good empirical/theoretical performances. Therefore, in using the algorithm GMP to search for good maximum period MRGs, it is important that we restrict the search within some special classes of MRGs. It is in this regard that we considered extending the DX/DL/DS/DT generators and the DW-k generators in this study.

With these efficient computer searches using the *algorithm GMP*, we identified, and listed many efficient and portable MRGs of super-orders, $40751, 50551$, and 50873 which respectively have equi-distribution property up to $40751, 50551$, and 50873 dimensions and period lengths of approximately $10^{380278.1}$, $10^{471730.6}$, and $10^{474729.3}$.

In Chapter 4, we look at some traditional ways of computing the spectral test. This concept is clearly illustrated in Section 4.2 with a simple LCG[B,23] generator. Generally, the spectral distance is calculated by solving some minimization problems. However, solving such a problem becomes increasingly difficult as the dimension, k, of the generator, increases. We discuss a new and simple method proposed by Winter, 2014, using the `LLL` algorithm in Section 4.3. In Section 4.4, we searched and listed "better" super-order `DX-k-s-t` generators for $k = 40751, k = 50551$ and $k = 50873$.

In Chapter 5, we examined yet, another special class of `MRGs` with many nonzero terms, known as the `DW-k` generator as proposed by Winter, 2014. Using the relationships between `MRGs` and `MCGs`, and our k values that we obtained in Chapter 3, and after a profound computer searched, we extended the `DW-k` generators to super-order generators with efficient and parallel implementations.

A summary of the major findings of this study is reported in Chapter 6.

Chapter 2

Literature Review

2.1 Some Developments in PRNGs

The goal of PRNGs in many computer applications is to produce a sequence of variates that is very hard to distinguish from a sequence of truly random numbers. Therefore it is desirable for an ideal PRNG to satisfy some desirable properties, such as: long period length, high-dimensional equi-distribution, efficiency, portability, nice empirical performance. A PRNG has a *"k-distribution property"* or *"equi-distribution property over k-dimensions"* if every r-tuple ($1 \leq r \leq k$) of numbers appears exactly the same number of times, except for the all-zero tuple that appears one time less, Lidl and Niederreiter, 1994, Theorem 7.43.

Notation

In this study, we defined p as a large prime number and $\mathbb{Z}_p \equiv \{0, 1, 2, \cdots, p-1\}$ is the finite field of p elements under the usual operations of addition and multiplication with modulus p.
$\mathbb{Z}_p^r$ and $\mathbb{Z}_p^k$, denotes the set of r-dimensional and k-dimensional vectors, respectively, with elements in $\mathbb{Z}_p$.
$\mathbb{R}^r$ denotes the set of r-dimensional vectors with elements in $\mathbb{R}$.
$\mathbb{Z}_p^{k \times k}$, denotes the set of $k \times k$ matrices, with elements in $\mathbb{Z}_p$.
MRG(k,p) denotes the set of maximum period k-th order MRG with prime modulus p.
The function $\phi(x)$ denotes the Euler's totient function (Euler's phi function) which gives the number of integers between 1 and x that are relatively prime (coprime) to x.

Where need be, we may restate some of these notations for emphasis.

Linear congruential Generators(`LCGs`)

An `LCG` proposed by D. H. Lehmer, 1951, is a simple recursive relation defined as follows:

$$X_i = (BX_{i-1} + A)\, mod\, p, \quad i \geq 0 \tag{2.1}$$

X_i, A, B, p are positive integers and the seed $X_{-1} \neq 0$ is chosen from $\mathbb{Z}_p$. If $A \neq 0$, it is possible to achieve the full period p. Nevertheless, according to MARSAGLIA, 1972, the ”effective period” cannot be greater than the period of the corresponding `LCG` with $A = 0$ as

$$X_i = (BX_{i-1})\, mod\, p, i \geq 0 \tag{2.2}$$

where $X_{-1} \neq 0$. When p is a prime number and B is a primitive element (primitive root) modulo p, the LCG in equation (2.2) has period $p - 1$. B is a primitive root modulo p, if for any prime factor q of $(p - 1)$, $B^{(p-1)/q} \neq 1 \; mod \; p$. The total number of primitive roots is $\phi(p - 1)$, where $\phi(x)$ is the Euler's totient function counting the number of integers between 1 and x that are relatively prime to x.

`LCGs` are popular for their simplicity, efficiency, and well-known theoretical properties. Nevertheless, they have short periods, lack equi-distribution in dimensions higher than 1, and have questionable empirical performances and all `LCGs` have failed badly some stringent empirical tests in L'Ecuyer and Simard, 2007.

Multiple Recursive Generators (MRGs)

Due to these shortcomings of LCGs, it is natural to find better classes of PRNGs. One of such better classes is the MRG. MRGs are natural extensions of LCGs, where the next value is computed iteratively from the previous k values. An MRG is defined recursively as

$$X_i = (\alpha_1 X_{i-1} + \alpha_2 X_{i-2} + \cdots + \alpha_k X_{i-k}) \, mod \, p, \quad i \geq k \tag{2.3}$$

for any initial seeds $(X_0, X_1, \cdots, X_{k-1}) \neq (0, 0, \cdots, 0)$, where the modulus p is a large prime number and the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ are integers between 0 and $p-1$, inclusively. The seeds can be chosen arbitrarily and k is a positive integer known as the order of the MRG. The maximum period of MRG is $p^k - 1$. This can be achieved by choosing the coefficients $\alpha_1, \alpha_2, \cdots, \alpha_k$ such that the polynomial

$$f(x) = x^k - \alpha_1 x^{k-1} - \cdots - \alpha_k \tag{2.4}$$

is a primitive polynomial, see, for example Deng, J.-J. H. Shiau, and Lu, 2012b.
X_i can be converted into a real value between 0 and 1 by using either $U_i = X_i/p$ or as recommended in Deng and H. Xu, 2003, $U_i = (X_i + 0.5)/p$

Checking whether an MRG has maximum period is equivalent to checking whether its characteristic polynomial (2.4) is a k-degree primitive polynomial over $\mathbb{Z}_p$. As mentioned in Chapter 1, Alanen and D. E. Knuth, 1964 and D. Knuth, 1998 proposed three necessary and sufficient conditions for a polynomial in (2.4) to be primitive. Their algorithm, as stated below, check the 3 conditions in order.

Algorithm AK

1. $(-1)^{k-1}\alpha_k$ must be primitive root $mod\ p$.

2. $x^R = (-1)^{k-1}\alpha_k\ mod\,(f(x), p)$, where $R = \frac{(p^k-1)}{(p-1)}$

3. For each prime factor q of R, the degree of $x^{\frac{R}{q}}\ mod\,(f(x), p)$ is positive.

If all the three conditions are met, then $f(x)$ is a primitive polynomial.
However, when k and p are large, it is difficult to factorize $R(k, p) = (p^k - 1)/(p - 1)$ which is one of the conditions. Besides, there is no early exit strategy for a failed search for a non primitive polynomial. Deng, 2004, called this number $R(k, p) = (p^k - 1)/(p - 1)$, a *Generalized Mersenne Prime(GMP)* and proposed an efficient algorithm*(algorithm GMP)* that bypasses the difficulty of factorizing the large number $R(k, p) = (p^k - 1)/(p - 1)$ and provides an early exit strategy for a failed search to achieve better efficiency. It should be noted that the idea of bypassing factoring a large number was first suggested by L'Ecuyer, Blouin, and Couture, 1993.

As we mentioned earlier, an `MRG` with order k and period p is denoted as `MRG(k,p)`. `MRGs` have very desirable features, such as: huge period length, excellent empirical performance, equi-distribution property up to k dimensions [see, Lidl and Niederreiter, 1994, Theorem 7.43, for details]. There is strong statistical justification for `MRGs` as well. An `MRG` will become "more and more uniform" with larger number of nonzero terms in the summation, for details, see, for example, Deng, 2016 and Deng and George, 1990.
When available, maximum period `MRGs` with large order k are preferred because as the order k increases for a given prime modulus p, the large period and equi-distribution properties become more advantageous, it has extremely long period length $p^k - 1$ and, have excellent empirical performances. Furthermore, D. Knuth, 1998, page 95, proposed that a generator with longer period should be used because it would have better "accuracy" in higher dimensions and commented again in D. Knuth, 1998, page 30, that, *"all known evidence indicates that the result will be very satisfactory source of random numbers"* for the sequence generated by an `MRG`.

Nevertheless, as k or p increases, it is difficult to find parameters $\alpha_1, \alpha_2, \cdots \alpha_k$ for order k and modulus p such that the MRG in (2.3) is of maximum period.
As we mentioned earlier in Chapter 1, not all MRGs are efficient in generating random numbers or having good empirical/theoretical performances. Therefore, in using algorithm GMP to search for good maximum-period MRGs, it is important that we restrict the search within some special classes of MRGs. It is in this regard that we considered the DX/DL/DS/DT generators and the DW-k generators in this study.

2.2 Some special classes of multiple recursive generators

Although MRGs have an increased maximum period of $p^k - 1$, MRGs are less efficient than LCGs because of the several multiplications that are involved. To improve the speed of generation, Grube, 1973, L'Ecuyer and Blouin, 1988, and L'Ecuyer, Blouin, and Couture, 1993, suggested using two non-zero terms α_j and α_k $(1 \le j < k)$ of the MRG in (2.3) and provided portable implementations of MRGs satisfying these conditions. Deng and Lin (2000) proposed to set as many coefficients of α_i in an MRG to be 0 and/or ± 1 as possible. In particular, they proposed a fast multiple recursive generator (FMRG) which is a special MRG with maximum period $p^k - 1$ of the form

$$X_i = (BX_{i-k} \pm X_{i-1})\, mod\, p, \quad i \ge k \tag{2.5}$$

which required only a single multiplication and are almost as efficient as LCGs.

DX generators

Based on this idea of FMRGs, Deng and H. Xu, 2003, proposed a system of special MRGs, called DX-k generators, which are portable, efficient, and maximum period MRGs, where all nonzero coefficients of the recurrence are equal and k is the order of recurrence. Deng, 2005 modified and extended the DX generators as follows:

1. DX-k-1-t[**FMRG**] $(\alpha_t = 1, \alpha_k = B), \quad 1 \le t < k,$

$$X_i = (X_{i-t} + BX_{i-k})\, mod\, p, \quad i \ge k \tag{2.6}$$

2. DX-k-2-t $(\alpha_t = \alpha_k = B), \quad 1 \le t < k,$

$$X_i = B(X_{i-t} + X_{i-k})\, mod\, p, \quad i \ge k \tag{2.7}$$

3. DX-k-3-t $(\alpha_t = \alpha_{\lceil \frac{k}{2} \rceil} = \alpha_k = B), \quad 1 \le t < \lceil \frac{k}{2} \rceil,$

$$X_i = B(X_{i-t} + X_{i-\lceil \frac{k}{2} \rceil} + X_{i-k})\, mod\, p, \quad i \ge k \tag{2.8}$$

4. DX-k-4-t $(\alpha_t = \alpha_{\lceil \frac{k}{3} \rceil} = \alpha_{\lceil \frac{2k}{3} \rceil} = \alpha_k = B), \quad 1 \le t < \lceil \frac{k}{3} \rceil,$

$$X_i = B(X_{i-t} + X_{i-\lceil \frac{k}{3} \rceil} + X_{i-\lceil \frac{2k}{3} \rceil} + X_{i-k})\, mod\, p, \quad i \ge k \tag{2.9}$$

where $\lceil x \rceil$ is the ceiling function of x, which is the least integer $\ge x$. According to Deng and H. Xu, 2003, these generators are referred to as DX-k-s-t. s is the number of nonzero coefficients and their indices are about $k/(s-1)$ apart. The parameter t is introduced to slightly expand the class of generators. It is the smallest index j for which $\alpha_j = B$. DX-k-s is the special case when $t = 1$. The FMRG proposed by Deng and Lin, 2000 is another special case with $s = 1$ and $t = 1$. PRNGs with very few nonzero coefficients has a disadvantage of "bad initialization effect", that is, when the k-dimensional state vector is close to the zero vector, the subsequent members generated may stay within a neighborhood of zero for many of them before they can break away from this near zero state. This property is not desired in the sense of randomness. This "bad initialization effect" was first observed by Panneton, L'Ecuyer, and Matsumoto, 2006, for MT19937. MT19937 is a popular generator proposed by Matsumoto and Nishimura, 1998, it has a period length of

$2^{19937} - 1 \approx 10^{6001}$ and equi-distribution up to 623 dimensions. The `DX` generators belong to this category of `PRNGs` since it has very few nonzero coefficients.

In the next subsection, we look at an extension of the `DX` generators to a general class of efficient and portable `MRGs` with many nonzero terms. This class of `MRGs` can be designed to overcome ”bad initialization effect”, however, these `MRGs` tend to be inefficient. It is possible to find efficient implementations for certain generators by rewriting equation (2.10) below, as a simple higher-order recurrence equation, see, Deng and J.-J. H. Shiau, 2015 for details.

A general class of efficient generators

According to Deng and Bowman, 2017, to construct a general class of efficient generators, consider a special class of `MRGs` that have at most two different nonzero coefficients, for example, A and B. Let S_A and S_B be two index sets defined as $S_A = \{j|\alpha_j = A\}$ and $S_B = \{j|\alpha_j = B\}$, such that $S_A \cap S_B = \emptyset$. Deng, Li, J.-J. Shiau, and Tsai, 2008, proposed a general class of generators of the form:

$$X_i = A \sum_{j \in S_A} X_{i-j} + B \sum_{j \in S_B} X_{i-j} \; mod \; p \tag{2.10}$$

This class of generators is computationally efficient if we have only few elements in both S_A and S_B. The `DX` generators defined earlier is a special case of the generator in equation (2.10)

1. DL Generators

Deng, Li, J.-J. Shiau, and Tsai, 2008, examined the structure of the generators of the form in equation (2.10) with restrictions on the forms of S_A and S_B and proposed a class of `DL` generators with $S_A = \{1, 2, \cdots, t-1\}$ and $S_B = \{t, t+1, \cdots, k\}$ for $1 \leq t < k$. Then

$$X_i = A(X_{i-1} + X_{i-2} + \cdots + X_{i-t+1}) + B(X_{i-t} + X_{i-t-1} + \cdots + X_{i-k}) \; mod \; p, \; i > k \tag{2.11}$$

where t can be useful to expand the search space for maximal period DL generators.

Using higer-order recurrence, DL can be implemented efficiently as

$$X_i = X_{i-1} + A(X_{i-1} - X_{i-t}) + B(X_{i-t} - X_{i-(k+1)}) \; mod \, p \quad i \geq k+1 \tag{2.12}$$

where $X_0, X_1, \cdots, X_{k-1}$ are the initial seeds and X_k are computed according to equation (2.11). The efficiency and portability of the DL generators can be further improved by letting the coefficient of $A = 0, -1, 1$, or $-B$. This reduces one multiplication and several addition/subtraction operations. When $A = 0$ we have the simple form:

$$X_i = B(X_{i-t} + X_{i-t-1} + \cdots + X_{i-k}) \; mod \, p, \quad i \geq k, \; t \geq 1. \tag{2.13}$$

In this study, we consider the special case of $A = 0$ and $t = 1$, for simplicity. We refer to this as DL-k generators and is implemented efficiently by:

$$X_i = X_{i-1} + B(X_{i-t} - X_{i-(k+1)}) \; mod \, p \quad i \geq k+1 \tag{2.14}$$

For more about efficient implementation of DL generators, see Deng and J.-J. H. Shiau, 2015.

2. DS Generators

Deng, Li, J.-J. Shiau, and Tsai, 2008, proposed another set of generators that can be efficiently implemented, known as DS generators. These generators also have many nonzero terms and are defined as:

$$X_i = B \sum_{j=1}^{k} X_{i-j} - C X_{i-d} \; mod \, p, \quad i \geq k. \tag{2.15}$$

By introducing parameters B and C for the multipliers, and index d, the search parameter space is expanded. DS generators can be efficiently implemented using the following $(k+1)$-th order recurrence equation:

$$X_i = X_{i-1} + B(X_{i-1} - X_{i-k-1}) - C(X_{i-d} - X_{i-(d+1)}) \; mod \, p \quad i \geq k+1 \tag{2.16}$$

where $X_0, X_1, \cdots, X_{k-1}$ are the initial seeds and X_k are computed according to equation (2.15) According to Deng, Li, J.-J. Shiau, and Tsai, 2008, the DS generators have several special cases of interest. When $C = 0$, the DS generator is the same as the DL with $A = 0$ and $t = 1$. When $C = B$, the DS generator has exactly one zero coefficient at the d-th term:

$$X_i = B \sum_{j=1, j \neq d}^{k} X_{i-j} \; mod \, p, \tag{2.17}$$

which can be efficiently implemented as

$$X_i = X_{i-1} + B(X_{i-1} - X_{i-d} + X_{i-(d+1)} - X_{i-k-1}) \; mod \, p \quad i \geq k+1 \tag{2.18}$$

The parameter d can be chosen arbitrarily and for simplicity, the case $d = \lceil k/2 \rceil$ is referred to as DS-k generators.

For more on the efficient implementation of DS generators, see Deng and J.-J. H. Shiau, 2015.

3. DT Generators

As discussed in Deng, J.-J. Shiau, and Tsai, 2009, another class of MRGs, called DT generators, with many nonzero terms and unequal weights on each term is defined by

$$X_i = B^k X_{i-1} + B^{k-1} X_{i-2} + \cdots + B X_{i-k} \; mod \, p, \quad i \geq k. \tag{2.19}$$

DT generators can also be efficiently implemented by the following $(k+1)$-th order recurrence equation:

$$X_i = (B^{-1} + B^k)X_{i-1} - X_{i-k-1} \; mod \; p \quad i \geq k+1 \tag{2.20}$$

Where $D \equiv (B^{-1} + B^k) \; mod \; p$ can be pre-computed.

Details on efficient implementations of `DT` generators can be seen in Deng and J.-J. H. Shiau, 2015.

2.3 K-distribution property and Spectral Test

As seen earlier, some desirable properties of good multiple recursive generators (`MRGs`) are long period length, uniformity and equi-distribution in higher dimensions, efficiency, portability and good empirical performance. A `PRNG` has a *"k-distribution property"* or *"equi-distribution property over k-dimensions"* if every r-tuple ($1 \leq r \leq k$) of numbers appears exactly the same number of times, except for the all-zero tuple that appears one time less. That is, for an `MRG(k,p)`, over its whole period, $p^k - 1$, every r-tuple $(a_1, a_2, a_3, \cdots, a_r)$ of integers in $\mathbb{Z}_p^r$ appears exactly the same number of times (p^{k-r}), with the exception of the all-zero tuple $(0, 0, 0, \cdots, 0)$ that appears one time less ($p^{k-r} - 1$), Lidl and Niederreiter, 1994, Theorem 7.43. Thus for $r \leq k$, generators in `MRG(k,p)` are quite close to an "ideal" generator; only the all-zero tuple is generated one time less than the other r-tuples.

Nevertheless, for $U_i = X_i/p$ with the index i running through the entire period $(p^k - 1)$ of the maximum period (`MRG`) in (2.3), a key requirement for a good `MRG` is that the set of vectors of successive output values $\mathbf{T}_i = \left\{(U_i, U_{i+1}, \cdots, U_{i+r-1}) | i \in p^k - 1\right\}$ from all possible initial states, should cover the unit hypercube $[0, 1)^r$ very evenly. It should be noted that as proposed in Deng and H. Xu, 2003, the output U_i is often slightly modified, say, $U_i = (X_i + 0.5)/p$ to avoid returning zero. This has little impact on our work here, so we ignore it. The set of fixed non-negative integers, $I = \{0, 1, 2, \cdots, r-1\}$ could be thought of as the indices selected from the state to create all the possible r-tuples over all steps i in the period of the `MRG`.

Geometrically speaking, these r-tuples can be thought of as r-dimensional points or vectors such that

$$\mathbf{T}_i = \big\{(X_i/p, X_{i+1}/p, \cdots, X_{i+r-1}/p) | i \in p^k - 1\big\} \tag{2.21}$$

forms a lattice of points in an r-dimensional spaces, $[0,1)^r$.

Generally, for any finite set of integers $I = \{i_1, i_2, \cdots, i_r\}$, where $0 \le i_1 < i_2 < \cdots < i_r$, consider a multi-set $\Omega_r(I)$ of all r-dimensional output vectors $(U_{i_1}, U_{i_2}, \cdots, U_{i_r})$ obtained when the initial state $S_0 = (X_0, X_1, \cdots, X_{k-1})$ of the `MRG` in (2.3) runs over all its p^k possibilities:

$$\Omega_r(I) = \{(U_{i_1}, U_{i_2}, \cdots, U_{i_r}) \in [0,1)^r | S_0 \in \mathbb{Z}_p^k\} \tag{2.22}$$

If S_0 is picked at random uniformly from $\mathbb{Z}_p^k$, then $(U_{i_1}, U_{i_2}, \cdots, U_{i_r})$ has the uniform distribution $\Omega_r(I)$ as in (2.22).

From the *equi-distribution property over k-dimensions,* for $r \le k$, whenever $i_r - i_1 < k$, or equivalently $(i_r - i_1 + 1) \le k$, $\Omega_r(I)$ as stated in (2.22), contains every vector $(\mathbb{Z}_p^k/p)$, that is, every r-dimensional vector whose coordinates are in $\{0, 1/p, \cdots, p - 1/p\}$ exactly p^{k-r} times each. Clearly, the output coordinates are all multiples of $1/p$ and this is the best uniformity we can desire for.

For $r > k$, this ideal uniformity is not possible because $p^r > p^k = |\Omega_r(I)|$. That is, it is not possible, because the number of possible r-tuples, p^r, is larger than the period length of the `MRG`. More intriguingly, when $i_r - i_1 \ge k$, that is, $(i_r - i_1 + 1) > k$, this uniformity is no longer guaranteed, even if r is small. In fact, like the well-known problem for the `LCG` in Marsaglia, 1968, it is well known that $\Omega_r(I)$ as in (2.22), is the intersection of a lattice in $\mathbb{R}^r$ with the unit hypercube $[0,1)^r$, that is, these r-dimensional points lie on a relatively small family of equidistant parallel hyperplanes in a high dimensional space; see, for example, L'Ecuyer, 1997 and D. E. Knuth, 2014. This means that there are families of equidistant parallel hyperplanes in $\mathbb{R}^r$ such that each family covers $\Omega_r(I)$ as in (2.22).

Let $d_r(k)$ (since it is influence by the dimension r and order k) denote the maximum distance between adjacent hyperplanes, taken over all families of parallel hyperplanes that cover $\Omega_r(I)$ as in (2.22). If $d_r(k)$ is small, then the generator is consider "good" because the "gap" between two adjacent hyperplanes in r-dimensional space is small. If the dimension r is much larger than k, the maximum "gap" $d_r(k)$ becomes so large that no `MRG`, of fixed order k, can be considered "good". The evaluation of the performance of a given generator based on $d_r(k)'s$ is often called the spectral test, for example, see D. E. Knuth, 2014. In fact, D. Knuth, 1998, notes that *"Not only do all good generators pass this test, all generators now known to be bad fail it. Thus, it is by far the most powerful test known, and it deserves particular attention".*

Since all maximum period `MRGs` have the "perfect" lattice structure for dimensions up to order k, the difference on the lattice structure among the generators of the same order k can only lie in dimensions $r \geq k+1$. If $d_r(k)$ is used as a measure of efficiency(or figure of merits) to compare generators, then the wish will be for this value to be as small as possible. According to Deng, J.-J. H. Shiau, and Lu, 2012b, a well known lower bound for $d_r(k)$ is

$$d_r^*(k) = \begin{cases} \frac{p^{-k/r}}{q_r}, & r > k \\ 1/p, & r \leq k \end{cases}$$

where the constant q_r depends only on r; for example, see D. E. Knuth, 2014, L'ecuyer, 1999 and Kao and Tang, 1997. If the exact value of $d_r^*(k)$ is known, Fishman and Moore, 1986, proposed to compare generators with different values of modulus by normalizing $d_r(k)$ with $\frac{d_r^*(k)}{d_r(k)}$ so that the value is between 0 and 1, the larger the better. The unfortunate thing is that the value of q_r is known only for $r \leq 8$, thus when r is large, there is no general formula for q_r.

2.4 Relationships between Multiple Recursive Generators (MRGs) and Matrix Congruential Generators(MCGs)

As discussed in Section 2.1, the MRG, as defined in equation (2.3), is the k-order extension of the LCG as defined in equation (2.2). There is a simple relationship between an MRG′s companion matrix and its characteristic equation. For the MRG defined in equation (2.3), its corresponding companion matrix is defined as

$$\mathbf{M}_f = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_k & \alpha_{k-1} & \alpha_{k-2} & \cdots & \alpha_1 \end{pmatrix} \tag{2.23}$$

Its characteristic polynomial is defined as

$$f(x) = det(x\mathbf{I} - \mathbf{M}_f)\, mod\, p. \tag{2.24}$$

The matrix congruential generator (MCG), which is a natural k-th dimensional extension of the LCG as defined in equation in (2.2), is defined as:

$$\mathbf{X}_i = \mathbf{B}\,\mathbf{X}_{i-1}\, mod\, p, i \geq 0 \tag{2.25}$$

where $\mathbf{X}_i$ is a k-dimensional vector in $\mathbb{Z}_p^k$, $\mathbf{X}_{-1}$ is a nonzero vector, and the matrix multiplier $\mathbf{B}$ is a $k \times k$ matrix in $\mathbb{Z}_p^{k\times k}$. Many authors, such as, Niederreiter, 1986, L'Ecuyer, 1990, Franklin, 1964 and Grothe, 1987, considered MCG in their work.

The characteristic polynomial of an `MCG` is defined as

$$f_{\mathbf{B}}(x) = det(x\mathbf{I} - \mathbf{B})\, mod\, p. \tag{2.26}$$

An integer matrix $\mathbf{B} \in \mathbb{Z}_p^{k\times k}$ has order h if

$$h = \min_{j>0}\{j : \mathbf{B}^j\, mod\, p = I\}$$

As a vector sequence ($\mathbf{X}_i,\ i \geq 0$), an `MCG` has the maximum period of $p^k - 1$ if and only if the matrix $\mathbf{B}$ has the order of $p^k - 1$. Furthermore, it is well known that, the matrix $\mathbf{B}$ has the order of $p^k - 1$ if and only if its corresponding characteristic polynomial $f_{\mathbf{B}}(x)$ as defined in equation (2.26) is a primitive polynomial.

Particularly, for every generator in `MRG(k, p)` that has primitive polynomial, say, $f(x)$, the corresponding companion matrix $\mathbf{M}_f$ as defined in equation (2.23), has the order $p^k - 1$. With this in mind, Grothe, 1987, proposed a class of `MCGs` with $\mathbf{B} = \mathbf{T}\mathbf{M}_f\mathbf{T}^{-1}$, where $\mathbf{T}$ is an invertible $k \times k$ matrix in $\mathbb{Z}_p^{k\times k}$ and $\mathbf{M}_f$ is the companion matrix to a generator in `MRG(k, p)`. Thus, the first relationship between `MCGs` and `MRGs` is that, `MCGs` can be constructed from the companion matrix $\mathbf{M}_f$ of a generator in `MRG(k, p)`. The `MCG` shares the same characteristic polynomial as the generator in `MRG(k, p)` because $f_{\mathbf{B}}(x) = det(x\mathbf{I} - \mathbf{T}\mathbf{M}_f\mathbf{T}^{-1}) = det(x\mathbf{I} - \mathbf{M}_f) = f(x)$ over $\mathbb{Z}_p$.

Basic concepts of MCGs

Let's consider the sequence of k-dimensional vectors generated by an `MCG`, $\mathbf{X}_1, \mathbf{X}_2, \cdots$. As stated earlier, the maximum period of this `MCG` is $p^k - 1$, as a result, this vector sequence will repeat after the period $p^k - 1$ is reached. Thus, any nonzero k-dimensional vector in $\mathbb{Z}_p^k$ will appear only once. This uniformity property over the k-dimensional cube of $\mathbb{Z}_p^k$ is a requirement for $\mathbf{X}_i/p$ to resemble a k-dimensional random vector in $[0, 1]^k$. In addition, any r-dimensional sub-vector

$r < k$ from $\mathbf{X}_i, i = 1, 2, \cdots$ will have the equi-distribution property over r-dimensional space. This is similar to what we had for a maximum period `MRG`. That is, all r-dimensional nonzero sub-vectors will occur p^{k-r} times except for all the r-dimensional zero sub-vectors which will occur one time less ($p^{k-r} - 1$ times).

Grothe, 1987, first observed, using the Cayley-Hamilton Theorem that, when taken as a k-dimensional vector sequence, ($\mathbf{X}_i, i \geq 0$), the `MCG` satisfies the same recursion as the generator in `MRG(k, p)` whose companion matrix $\mathbf{M}_f$ was used to define the matrix multiplier $\mathbf{B}$. That is, the vector sequence, ($\mathbf{X}_i, i \geq 0$), satisfies the following recursion:

$$\mathbf{X}_i = (\alpha_1 \mathbf{X}_{i-1} + \alpha_2 \mathbf{X}_{i-2} + \cdots + \alpha_k \mathbf{X}_{i-k})\, mod\, p, \quad i \geq k \tag{2.27}$$

Thus, the k sequences taken from each of the k rows in equation (2.27) can be viewed as k copies of the same `MRG` with different starting seeds. That is, as a vector sequence, a maximum period `MCG` can be viewed as running k copies of the same generator in `MRG(k, p)` with different starting seeds. We can view these as k output streams corresponding to the k-rows in the generating equation (2.27). This is our second relationship between `MRGs` and `MCGs`.
Each stream is produce by the same `MRG` sequence using the same generating equation with different starting shifts. Due to the huge period length $p^k - 1$, the random starting seeds can produce a reasonably large shift among k different streams. This ”row-wise” output method is suitable for parallel simulation by assigning a different stream for each central processing unit (`CPU`). Thus, it is important to consider the problem of choosing the matrix multiplier $\mathbf{B}$ with the characteristic polynomial $f(x)$ so that the corresponding `MCG` is efficient to implement in either ”column-wise” or ”row-wise” mode.

2.5 Summary

In this Chapter, we reviewed some existing literature that shall be useful in this study. Specifically, some developments in `PRNGs` in Section 2.1 and some special classes of `MRGs` in Section 2.2, which we shall use mainly in Chapter 3, the k-distribution property in Section 2.3, which we will employ mostly in Chapter 4, and the relationships between `MRGs` and `MCGs`, discussed in Section 2.4, will be utilized in Chapter 5, for the efficient and parallel implementation of the `DW-k` generators.

Chapter 3

Search for Super-order Multiple Recursive Generators

3.1 Methods Used in Searching for Super-order Multiple Recursive Generators

In Section 2.1, we stated *algorithm AK* as proposed by Alanen and D. E. Knuth, 1964. One of the major shortcoming of this algorithm was the difficulty of factorizing $R(k, p) = (p^k - 1)/(p - 1)$. Certainly, this is the major difficulty in evaluation to carry out the test for primitivity modulo p. To avoid this difficulty of factoring $R(k, p) = (p^k - 1)/(p - 1)$, one can search for p with a fixed k such that $R(k, p)$ is a prime number. Obviously, k has to be an odd prime number, see Deng, J.-J. H. Shiau, and Lu, 2012b for details.

There are some studies in the field of number theory concerning primes of the form $(a^k-1)/(a-1)$, in particular for "small" values of a (from 2 up to 12). For example, when $a = 2$, it is a Mersenne number, for example, see details in [Williams and Seah, 1979, Brillhart, D. Lehmer, Selfridge, Tuckerman, and Wagstaff, 2002].

After finding prime modulus p for `GMPs`, Deng, 2004, proposed an efficient search algorithm called algorithm `GMP`, which will exit early when $f(x)$ in (2.4) is not a primitive polynomial. This early exit strategy saves a huge amount of time, especially when k is large. According to Deng, 2004, algorithm `GMP` is $1000+$ folds more efficient over the *algorithm AK* in D. Knuth, 1998. Furthermore, Deng and Bowman, 2017, illustrated this with a concrete example.

Some notions about prime factors of $R(k,p) = (p^k - 1)/(p-1)$

As mentioned earlier, as k increases, the likelihood of finding p such that $R(k,p) = (p^k - 1)/(p-1)$ is a prime by computer search decreases and the computing time for some probabilistic primality testing procedures increases drastically, see, for example Deng, J.-J. H. Shiau, and Lu, 2012b. The major drawback with most of the probabilistic primality testing programs is that, irrespective of whether the number under test is a prime or not, the amount of computing time is about the same. Therefore, it is important to have an early exit strategy to quickly screen out some $p's$ with a composite number $R(k,p)$, which will help to accelerate the search, see Deng, J.-J. H. Shiau, and Lu, 2012b, for details. With the help of some classical number theory results, such an efficient screening procedure is developed.

Theorem 1 below gives a characterization of the prime factors of $R(k,p)$. It is a special case of Legendre's *[Andrien-Marie Legendre (1752 - 1833)]* theorem for the prime factors of the form $a^k \pm b^k$, where a and b are integers with $gcd(a,b) = 1$ and k is any positive integer.

Theorem 1. [Theorem 2.4.3 in Williams, 1998, page 41]
Let k be a prime. For every prime factor q of $(p^k - 1)/(p-1)$, $q = 1 \ mod \ 2k.$

Thus, it is easy to see that any prime factor, say, q, of $(p^k - 1)/(p-1)$ should be of the form $2kc + 1$, for some integer c, because both q and k are odd primes. With this, we can rule out any prime number such that $q \neq 1 \ mod \ 2k$ to be a factor of $(p^k - 1)/(p-1)$.

Deng, J.-J. H. Shiau, and Lu, 2012a, used this simple result to find complete factorization for $k = 7499$ and $k = 20897$ with $p = 2^{31} - 1$ and with this, they found several efficient and portable `MRGs` of order $k = 7499$ and $k = 20897$. According to Deng, J.-J. H. Shiau, and Lu, 2012b, it remains an open question whether there is any k such that $(p^k - 1)/(p-1)$ is a `GMP`, for $p = 2^{31} - 1$.

In this study, we are looking for prime modulus p such that $(p^k - 1)/(p-1)$ is a prime, for $p = 2^{31} - v$, v a positive integer. With the help of Theorem 1, we can greatly accelerate this

computer search by ruling out all primes $q \neq 1 \ mod \ 2k$. To further quantify the savings, Deng, J.-J. H. Shiau, and Lu, 2012b used another powerful theorem in number theory, to illustrate this fact, for details on this, see Deng, J.-J. H. Shiau, and Lu, 2012b, Theorem 2.
Once a prime factor is found, the search program can go on to verify the next candidate p. Based on the aforementioned observation, Deng, J.-J. H. Shiau, and Lu, 2012b, developed the following screening strategies.

Strategies for Finding Generalized Mersenne Primes

According to Deng, J.-J. H. Shiau, and Lu, 2012b, the strategies that are used in finding proper (k, p) pairs such that $(p^k - 1)/(p - 1)$ is a prime can be summarized as follows:

1. For a given prime order k, compute Q_n, the product of all prime numbers $q \leq n$ of the form $2kc + 1$, for some integer c, say, with $n = 10^{12}$.

2. For a given prime p, check whether there is any common factor between Q_n and $(p^k - 1)/(p - 1)$:

   (a) If the primality test fails, move on to another p;

   (b) Otherwise, record the prime modulus p as the result for the current k, and then go on to repeat the whole procedure with the next prime order k.

With these strategies in mind, we used the primality tester software `PFGW` for the actual search test. Given a range $[x, y]$ for p and an order k, we used the strategies above to search for a prime p within a set where $(p - 1)/2$ is also a prime, a strategy adopted in L'Ecuyer, Blouin, and Couture, 1993. For 32-bit `PRNGs`, we started from the upper limit $x = 2^{31} - 1$ and moved downward until the search was successful or the lower limit y was reached. Deng, 2008, found some $p's$ for which $R(k, p)$ is a `GMP` for k up to 10007 and Deng, J.-J. H. Shiau, and Lu, 2012b, found some $p's$ for which $R(k, p)$ is a `GMP` for even larger values of k up to $k = 25013$. In this study, and for the 32-bit

PRNGs, with the skipping strategy described earlier, we were able to extend the search and find GMPs super large values for some $k's$ ($k = 40751, k = 50551$, and $k = 50873$.) The results for these $k's$, together with their corresponding v and p values for which $R(k,p) = (p^k - 1)/(p - 1)$ is a GMP are given in Table 1. Over 45 days of Central Processing Unit (CPU) time were spent[in an Intel(R), Xeon(R) CPU E5-2680 computer with a processing speed of 2.50GHz] in searching for the new $p's$ listed in Table 1.

Table 1. List of k, v, and $p = 2^{31} - v$, for which $(p^k - 1)/(p - 1)$ is a prime(GMP)

k	v	$p = 2^{31} - v$	$log_{10}(p^k - 1)$
40751	890301	2146593347	380278.1
50551	758421	2146725226	471730.6
50873	1359861	2146123787	474729.3

To address one of the major shortcomings of algorithm AK in Section 2.1 in finding MRGs, which involved factoring a huge number like $p^k - 1$, we must mention that L'Ecuyer, Blouin, and Couture, 1993, proposed a method for bypassing this difficulty for $k \leq 7$ and later L'Ecuyer, 1999, extended the method for $k \leq 13$ that focused on finding p such that

$$R(k,p) = (p^k - 1)/(p - 1) \tag{3.1}$$

is also a prime.

Note that Deng, 2004, called $R(k,p) = (p^k - 1)/(p - 1)$ a *Generalized Mersenne Prime(GMP)*; this is different from a Mersenne prime *[named after the French Monk, Marin Mersenne (1588 - 1648)]*, which is a prime of the form $2^k - 1$. The popular Mersenne Twister (MT19937) generator proposed by Matsumoto and Nishimura, 1998, is based on a particular Mersenne prime with $k = 19937$. Presently, there are only 51 known Mersenne primes. For details, see, the Great Internet Mesrsenne Prime search (GIMPS) at https://www.mersenne.org/. Although GMPs are extensions of Mersenne primes, the goal of GMPs is different as we have discussed above.

With this new pairs of (k,p) that we have found via GMPs, we will now use algorithm GMP

to search for efficient and portable maximum period MRGs, within the DX/DL/DS/DT classes in the next section.
As Deng, 2008, argued, *"Once $R(k, p)$ is claimed as a probable prime, it is fairly safe to claim that the primitive polynomial found subsequently by algorithm GMP, is indeed a primitive polynomial and hence a maximum period MRG is found".*

3.2 Super-order Multiple Recursive Generators

Although MRGs have an increased maximum period of $p^k - 1$, MRGs are less efficient than LCG because of the several multiplications that are involved. To improve the speed of generation, Grube, 1973, L'Ecuyer and Blouin, 1988, and L'Ecuyer, Blouin, and Couture, 1993, suggested using two nonzero terms α_j and α_k $(1 \leq j < k)$ of the MRG in equation (2.3) and provided portable implementations of MRGs satisfying these conditions. Deng and Lin, 2000 proposed to set as many coefficients of α_i in an MRG to be 0 and/or ± 1 as possible. In particular, they proposed a fast multiple recursive generator (FMRG) which is a special MRG with maximum period, $p^k - 1$. What we can infer from this is that, not all MRGs are efficient in generating random numbers or having good empirical/theoretical performances. Therefore, in using algorithm GMP to search for good maximum period MRGs, it is important that we restrict the search within some special classes of MRGs. It is in this regard that we considered extending the DX/DL/DS/DT MRGs in this Chapter.

Finding Maximum Period MRGs

We used an efficient search algorithm known as algorithm GMP proposed by Deng, 2004, to find maximal period MRGs of super-orders. As mentioned earlier, this algorithm provides an early exit strategy to overcome a second bottleneck in the *algorithm AK* proposed by D. Knuth, 1998.

Algorithm GMP

Let p be a prime, $R = (p^k - 1)/(p-1)$ and $f(x)$ be as in equation (2.4)

1. $(-1)^{k-1}\alpha_k$ must be a primitive element modulo p.

2. Initially, let $g(x) = x$. For $i = 1, 2, 3, \cdots, \lfloor \frac{k}{2} \rfloor$, where $\lfloor x \rfloor$ is the largest integer $\leq x$, do:

   (a) $g(x) = g(x)^p \, mod \, f(x)$;

   (b) $d(x) = gcd(f(x), g(x) - x)$;

   (c) if $d(x) \neq 1$, then $f(x)$ cannot be a primitive polynomial. Exit.

3. For each prime factor q of R, the degree of $x^{R/q}\ mod\ (f(x), p)$ is positive.

If all three steps have been passed with no early exit, then $f(x)$ is a primitive polynomial. For details see, Deng, 2005 and Deng and J.-J. H. Shiau, 2015.

The chance of finding one primitive polynomial is less than $1/k$. Algorithm GMP can greatly reduce the search time to a small fraction. In fact, as mentioned earlier in Section 3.1, this early exit strategy saves a huge amount of time, especially when k is large. According to Deng, 2004, algorithm GMP is $1000+$ folds efficient over algorithm AK in D. Knuth, 1998. Furthermore, Deng and Bowman, 2017, illustrated this early exit strategy with a concrete example. In this example, a typical search result for an efficient MRG, called DX-k, with $k = 3803$, shows that the number of iterations for the search is 2858. For the 2857 failed searches, 90% will exit at $i \leq 6$ iterations.
Thus, algorithm GMP has two main advantages. The first advantage is that it avoids the factorization $(p^k - 1)$ by searching for p such that $R(k, p) = (p^k - 1)/(p - 1)$ is a prime. It is a well known fact that problem primality testing is much easier than factorization, see, for example, Agrawal, Kayal, and Saxena, 2004. The second advantage is that it provides an early exit strategy for failed search, which allows for a quick exit in most cases, except for a successful search.
Using algorithm GMP and the various values of k as given in Table 1, we found many primitive polynomials of degrees up to $40751, 50551, 50873$ corresponding to super-order MRGs of periods

up to approximately $10^{380278.1}, 10^{471730.6}, 10^{474729.3}$, respectively. More on these are discussed in the following sections.

3.3 Super-order DX generators

According to L'Ecuyer, 1997, a necessary (but not sufficient) condition for a "good" MRG is that the sum of squares of coefficients, $\sum_{i=1}^{k} \alpha_i^2$, should be large. Therefore MRGs with large $\sum_{i=1}^{k} \alpha_i^2$, should normally be preferable. For DX generators, this entails that s and B should be as large as possible, while maintaining the efficiency and portability property.
For portability, Deng, 2005, discuss some common approaches that impose certain limits on the size of B so that the exact results of the multiplication can be produced with all computing platforms. In particular, Deng, 2005, proposed;

- $B \leq 2^{30}$, for $s \leq 4$, for a system-dependent $64-$bit integer data type which is available in many popular compilers in $32-$bit computer systems.
- $B \leq 2^d$, where $d = 20$, when $s = 1, 2$; $d = 19$, when $s = 3, 4$, following Institute for Electrical and Electronics Engineers (IEEE) double precision standard.

Details on these choices of B can be seen in Deng, 2005.

For $DX - 40751$ and $DX - 50873$ generators and $B \leq 2^{30}$, we took intervals of approximately 10^8 and searched for maximum B in each interval. With these B values, we listed several maximum period $DX - k - s - 1$ generators.

The average search time(in seconds) for each $DX-40751-s-1$ generator, using algorithm GMP is 34696. The summary results for the 40, $DX - 40751 - s - 1$ generators that we found, are given in Table 2.

Illustrative Example

We randomly selected $DX - 40751 - 4 - 1$ for illustration with $B = 1075553705$.

Table 2. List of 40 $DX - 40751 - s - 1$ generators, for $v = 890301, p = 2^{31} - v = 2146593347$, $B < 2^{30}$.

$B \leq$	**s=1**	**s=2**	**s=3**	**s=4**
10^8	99999187	99943616	99989294	99908826
$2x10^8$	199882798	199976817	199757194	199908630
$3x10^8$	299989859	299810510	299947826	299926324
$4x10^8$	399941527	399782020	399837375	399929553
$5x10^8$	499891165	499815925	499995545	499949260
$6x10^8$	599888895	599948335	599973994	599977028
$7x10^8$	699982493	699939982	699997061	699918402
$8x10^8$	799960812	799982007	799994827	799948997
$9x10^8$	899996464	899971880	899903209	899946239
1075741824	1075651623	1075529026	1075474258	1075553705

The primitive polynomial found after a search time of 37533 seconds, is:

$$f(x) = x^{40751} - 1075553705 * x^{40750} - 1075553705 * x^{27167} - 1075553705 * x^{13583} - 1075553705$$

The super-order maximum period $DX - 40751 - 4 - 1$ is:

$$X_i = 1075553705\,(X_{i-1} + X_{i-13584} + X_{i-27168} + X_{i-40751})\; mod\; 2146593347, \quad i \geq 40751 \quad (3.2)$$

The average search time(in seconds) for each $DX - 50873 - s - 1$ generator, using algorithm `GMP` is 54508.6. The summary results for the 40, $DX - 50873 - s - 1$ generators that we found, are given in Table 3.

Table 3. List of 40 $DX - 50873 - s - 1$, generators for, $v = 1359861, p = 2^{31} - v = 2146123787$, $B < 2^{30}$.

$B \leq$	**s=1**	**s=2**	**s=3**	**s=4**
10^8	99759194	99896807	99752885	99972212
$2x10^8$	199989793	199987588	199977331	199975550
$3x10^8$	299854494	299931435	299925758	299691408
$4x10^8$	399881831	399949293	399823824	399652737
$5x10^8$	499941247	499896845	499909502	499968105
$6x10^8$	599893817	599945685	599946254	599981116
$7x10^8$	699842887	699790261	699953754	699890205
$8x10^8$	799948441	799905440	799816150	799855408
$9x10^8$	899602700	899949236	899870427	899987637
1075741824	1075644962	1075133221	1075699644	1075505328

Illustrative Example

Similarly, we used $DX - 50873 - 3 - 1$ for illustration with $B = 1075699644$.

The primitive polynomial found after a search time of 52789 seconds, is:

$$f(x) = x^{50873} - 1075699644 * x^{50872} - 1075699644 * x^{25436} - 1075699644$$

The super-order maximum period $DX - 50873 - 3 - 1$ is:

$$X_i = 1075699644\,(X_{i-1} + X_{i-25437} + X_{i-50873})\ mod\ 2146123787, \quad i \geq 50873 \qquad (3.3)$$

After generating several super-order $DX - 50873 - s - 1$ generators, and considering the fact that $k = 50551$ is closer to $k = 50873$ (both are in the $50,000$ range), we limited our search for maximum period super-order $DX - 50551 - s - 1$ to $B < 2^{30}$.

We also consider the case where $B < 2^d,\ d = 19$, 20 as discussed in Deng, 2005, to generate some super-order $DX - 40751 - s - 1$, $DX - 50551 - s - 1$, and $DX - 50873 - s - 1$ generators as well. The summary results of the generators we found are listed in Table 4.

Table 4. List of $DX - k - s - 1$ generators with $B < 2^d (d = 20$ for $s = 1, 2$ and $d = 19$ for $s = 3, 4)$ and $B < 2^{30}$.

$DX - k - s - 1$	$B < 2^{19}/B < 2^{20}$	$B < 2^{30}$
$DX - 50551 - 1 - 1$	998201	1073390951
$DX - 50551 - 2 - 1$	1044469	1073724894
$DX - 50551 - 3 - 1$	515561	1073646955
$DX - 50551 - 4 - 1$	461111	1073646756
$DX - 40751 - 1 - 1$	949211	
$DX - 40751 - 2 - 1$	973351	
$DX - 40751 - 3 - 1$	433849	
$DX - 40751 - 4 - 1$	509184	
$DX - 50873 - 1 - 1$	1004567	
$DX - 50873 - 2 - 1$	1016882	
$DX - 50873 - 3 - 1$	470516	
$DX - 50873 - 4 - 1$	370676	

Illustrative Examples

We randomly selected some super-order maximum period generators from those listed in Table 4 for illustration.

1. We used $DX - 50551 - 2 - 1$ to illustrate an example with $B = 1073724894$.

   The primitive polynomial found after a search time of 48118 seconds, is:

$$f(x) = x^{50551} - 1073724894 * x^{50550} - 1073724894$$

   The super-order maximum period $DX - 50551 - 2 - 1$ is:

$$X_i = 1073724894 \, (X_{i-1} + X_{i-50551}) \; mod \; 216725226, \quad i \geq 50551 \qquad (3.4)$$

2. As another example, we took $B < 2^{20} (B = 1004567)$ and $DX - 50873 - 1 - 1$ for illustration.

The primitive polynomial found after a search time of 55091 seconds, is:

$$f(x) = x^{50873} - 1 * x^{50872} - 1004567$$

The super-order maximum period $DX - 50551 - 2 - 1$ is:

$$X_i = 1X_{i-1} + 1004567X_{i-50873} \; mod \; 2146123787, \quad i \geq 50551 \tag{3.5}$$

Using algorithm `GMP`, we have successfully found and listed several efficient and portable super-order `DX-k-s-1` generators. We now tend our attention to `DL/DS/DT` class of generators.

3.4 DL/DS/DT Generators

Using the same B values that we used for the `DX` generators, as discussed in Section 3.3, we found some super-order, maximum period, efficient and portable `DL/DS/DT` generators. These found generators are listed in Table 5.

Table 5. List of k, p with $B < 2^{30}$ and $B < 2^d (d = 19, 20)$ for $DL/DS/DT$ generators.

k	**p**	**DL**		**DS**		**DT**	
		$B < 2^{30}$	$B < 2^{20}$	$B < 2^{30}$	$B < 2^{19}$	$B < 2^{30}$	$B < 2^{20}$
40751	2146593347	1073688686	1031270	1073726060	1008189	1073568775	947002
50551	2146725226	1073693006	951636	1073589974	852713	1073716997	876442
50873	2146123787	1073626564	1049100	1073681776	841776	1073547854	998884

Illustrative Examples

We took $k = 40751$, $k = 50551$ and $k = 50873$ to illustrate the super-order `DL`, `DS` and `DT` generators respectively, for $B < 2^{30}$.

1. $DL-40751$ *generator with* $B=1073688686$

   The primitive polynomial found after a search time of 35142 seconds, is:

$$f(x)=x^{40751}-1073688686*x^{40750}-1073688686$$

   The super-order maximum period $DL-40751$ is:

$$X_i=1073688686\,(X_{i-1}+...+X_{i-40751})\;mod\;2146593347$$

   which can be implemented efficiently as

$$X_i=X_{i-1}+1073688686\,(X_{i-1}-X_{i-40752})\;mod\;2146593347,\quad i\geq 40752 \qquad (3.6)$$

2. $DS-50551$ *generator with* $B=1073589974$

   The primitive polynomial found after a search time of 56461 seconds, is:

$$f(x)=x^{50551}-1073589974*x^{50550}-0*x^{25275}-1073589974*x^{25274}-1073589974$$

   The super-order maximum period $DS-50551$ is:

$$X_i=1073589974X_{i-1}+.....+0X_{i-25276}+1073589974X_{i-25277}+....+1073589974X_{i-50551}\;mod\;214672522$$

   which can be implemented efficiently as

$$X_i=X_{i-1}+1073589974\,(X_{i-1}-X_{i-25276}+X_{i-25277}-X_{i-50552})\;mod\;2146725226,\;i\geq 50552 \qquad (3.7)$$

3. $DT - 50873$ *generator with* $B = 1073547854$

The primitive polynomial found after a search time of 52690 seconds, is:

$$\begin{aligned}f(x) = x^{50873} &- 1571578769 * x^{50872} - 1387673364 * x^{50871} - 1544646437 * x^{50870} \\ &- 862963011 * x^{50869} - 567428266 * x^{50868} - 514856600 * x^{50867} - 1830446773 * x^{50866} \\ &- 131303340 * x^{50865} - 1458765425 * x^{50864} - \ldots\ldots - 340965073 * x^{10} - 236641122 * x^{9} \\ &- 1206613138 * x^{8} - 1407971913 * x^{7} - 49919788 * x^{6} - 121166587 * x^{5} - 1435531505 * x^{4} \\ &- 1170757851 * x^{3} - 516639799 * x^{2} - 620521937 * x^{1} - 1073547854\end{aligned}$$

The super-order maximum period $DT - 50873$ generator is:

$$\begin{aligned}X_i = 1571578769X_{i-1} &+ 1387673364X_{i-2} + 1544646437X_{i-3} + 862963011X_{i-4} + 567428266 \\ X_{i-5} &+ 514856600X_{i-6} + 1830446773X_{i-7} + 131303340X_{i-8} + 1458765425X_{i-9} + \ldots\ldots \\ &+ 340965073X_{i-50863} + 236641122X_{i-50864} + 1206613138X_{i-50865} + 1407971913X_{i-50866} \\ &+ 49919788X_{i-50867} + 121166587X_{i-50868} + 1435531505X_{i-50869} + 1170757851X_{i-50870} \\ &+ 516639799X_{i-50871} + 620521937X_{i-50872} + 1073547854X_{i-50873} \; mod\ 2146123787\end{aligned}$$

which can be implemented efficiently as

$$X_i = 1849091597X_{i-1} - X_{i-50874} \; mod\ 2146123787, \quad i \geq 50874 \tag{3.8}$$

3.5 Empirical Evaluation

Once PRNGs have been designed and implemented, their empirical performance needs to be evaluated. There are several well known empirical test packages for testing a PRNG. Some of the best known are: DIEHARD proposed by Marsaglia, 1996, the test suite implemented by the National Institute of Standards and Technology(NIST) of the USA, Rukhin, 2000, and TestU01 test package which was developed by Professor L'Ecuyer with source code from http://www.iro.umontreal.ca/ lecuyer/. See, L'Ecuyer and Simard, 2007. It is the most comprehensive test package. The TestU01 test package has three predefined test modules:

- *Small Crush:* It has 10 tests and computes 15 test statistics and p-values.
- *Crush:* It has 96 tests and computes 144 test statistics and p-values.
- *Big crush:* It is the most comprehensive with 106 tests and computes 160 test statistics and p-values.

We evaluated the generators listed in Tables 2-5 with the Small Crush and Crush batteries in version $1.2.3$ of TestU01 with five different starting seeds. Each seed vector consists of k initial seeds generated by an $LCG : X_i = BX_{i-1}\ mod\ p$, where the multiplier B and the modulus p are the same as that of the `MRG` under consideration. We use an `LCG` whose multiplier is the same as B to generate the required k initial seeds. For details, see Deng, J.-J. H. Shiau, and Lu, 2012b. The five different starting seeds used in this study are $1, 12, 123, 1234$ and 12345 following the proposal in Deng, J.-J. H. Shiau, and Lu, 2012b.

The size of a p-value represents the probability of observing a test statistic more extreme than the one observed when the null hypothesis is true. The smaller the p-value is, the more significant the test result gets, and this normally indicates the generator fails the particular test more severely. One the other hand, when the p-value is too close to 1, it is considered as "too good to be truly random." For details, see L'Ecuyer and Simard, 2007. According to L'Ecuyer and Simard, 2007, to pass all the test, no p-value should be outside the range $[10^{-10}, 1 - 10^{-10}]$. In the empirical evaluation for this study, the number of tests with p-values less than α or greater than $1 - \alpha$, for $\alpha = 10^{-3}, 10^{-4}$ and 10^{-5} are tabulated for the `DX, DL, DS` and `DT` generators under testing.

We evaluated the forty $DX-40751-s-1$ and forty $DX-50873-s-1$ (for $s = 1, 2, 3, 4$) generators listed in Table 2 and Table 3 respectively. For the Small Crush battery of tests, we obtained 750 (=10 x 15 x 5) p-values for each class of generators listed and a total of $3,000$(=750 x 4) p-values for all the forty $DX - 40751 - s - 1$ and forty $DX - 50873 - s - 1$ generators. Similarly, for the Crush battery of tests, we obtained $7,200$ (=10 x 144 x 5) p-values for each class of generators listed and a total of $28,800$(=7,200 x 4) p-values for all the forty $DX - 40751 - s - 1$ and forty $DX - 50873 - s - 1$ generators.

For the forty $DX - 40751 - s - 1$ generators listed in Table 2, the p-values for the Small Crush battery of tests are tabulated in Table 6 and that of Crush in Table 7. As we can see from the two Tables 6-7, none of these tests produces a p-value less than 10^{-5} or greater than $1 - 10^{-5}$. For the Small Crush tests in Table 6, the proportion of p-values below 10^{-3} is 0.00067 and that for the Crush tests in Table 7 is 0.00069. Besides, none of these tests produces a p-value very close to 0 or 1 or better still, none of these p-values is outside the range $[10^{-10}, 1 - 10^{-10}]$. Thus, we can conclude that each of the forty $DX - 40751 - s - 1$ generators listed in Table 2 passed both the Small Crush and Crush batteries of tests in the TestU01 suite.

Table 6. Results of Small Crush tests on DX-40751-s-1 with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DX-40751-s-1 (750 p-values each)							
DX-40751-1-1	Count	1	0	0	1	1	0
Table2	Proportion	0.00133	0	0	0.00133	0.00133	0
DX-40751-2-1	Count	1	0	0	3	0	0
Table2	Proportion	0.00133	0	0	0.004	0	0
DX-40751-3-1	Count	0	0	0	1	1	0
Table2	Proportion	0	0	0	0.00133	0.00133	0
DX-40751-4-1	Count	0	0	0	0	1	0
Table2	Proportion	0	0	0	0	0.00133	0
DX-40751-s-1 (3,000 p-values in total)							
DX-40751-s-1	Count	2	0	0	5	3	0
Table2	Proportion	0.00067	0	0	0.00167	0.00100	0

For the forty $DX - 50873 - s - 1$ generators listed in Table 3, the p-values for the Small Crush battery of tests are tabulated in Table 8 and that of Crush in Table 9. As we can see from the two Tables 8-9, none of these tests produces a p-value less than 10^{-5} and only one p-value is greater than $1 - 10^{-5}$. For the Small Crush tests in Table 8, the proportion of p-values below 10^{-3}

Table 7. Results of Crush tests on DX-40751-s-1 with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DX-40751-s-1 (7200 p-values each)							
DX-40751-1-1	Count	4	0	0	10	0	0
Table2	Proportion	0.00056	0	0	0.00139	0	0
DX-40751-2-1	Count	3	0	0	11	0	0
Table2	Proportion	0.00042	0	0	0.00153	0	0
DX-40751-3-1	Count	8	0	0	3	1	0
Table2	Proportion	0.00111	0	0	0.00042	0.00014	0
DX-40751-4-1	Count	5	0	0	5	3	0
Table2	Proportion	0.00069	0	0	0.00069	0.00042	0
DX-40751-s-1 (28,800 p-values in total)							
DX-40751-s-1	Count	20	0	0	36	4	0
Table2	Proportion	0.00069	0	0	0.00125	0.00014	0

is 0.00100, and that for the Crush tests in Table 9 is 0.00125. Besides, none of these tests produces a p-value very close to 0 or 1. Similarly, we can conclude that each of the forty $DX-50873-s-1$ generators listed in Table 3 passed both the Small Crush and Crush batteries of tests in the TestU01 suite.

Table 8. Results of Small Crush tests on DX-50873-s-1 with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DX-50873-s-1 (750 p-values each)							
DX-50873-1-1	Count	1	0	0	1	0	0
Table3	Proportion	0.00133	0	0	0.00133	0	0
DX-50873-2-1	Count	0	0	0	0	0	0
Table3	Proportion	0	0	0	0	0	0
DX-50873-3-1	Count	2	0	0	2	0	0
Table3	Proportion	0.00267	0	0	0.00267	0	0
DX-50873-4-1	Count	0	0	0	1	0	0
Table3	Proportion	0	0	0	0.00133	0	0
DX-50873-s-1 (3,000 p-values in total)							
DX-50873-s-1	Count	3	0	0	4	0	0
Table3	Proportion	0.00100	0	0	0.00133	0	0

Table 9. Results of Crush tests on DX-50873-s-1 with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DX-50873-s-1 (7,200 p-values each)							
DX-50873-1-1	Count	8	0	0	9	1	1
Table3	Proportion	0.00111	0	0	0.00125	0.00014	0.00014
DX-50873-2-1	Count	7	0	0	8	0	0
Table3	Proportion	0.00097	0	0	0.00111	0	0
DX-50873-3-1	Count	9	0	0	9	0	0
Table3	Proportion	0.00125	0	0	0.00125	0	0
DX-50873-4-1	Count	12	0	0	9	0	0
Table3	Proportion	0.00167	0	0	0.00125	0	0
DX-50873-s-1 (28,800 p-values in total)							
DX-50873-s-1	Count	36	0	0	35	1	1
Table3	Proportion	0.00125	0	0	0.00122	0.00003	0.00003

We now evaluate the sixteen $DX - k - s - 1$ generators listed in Table 4. Unlike the ones above, we have not reported the results of the Small Crush battery of tests because the results are similar and the table is a bit larger. However, all the generators listed in Table 4 passed these tests. For the Crush battery of tests, we obtained $1,440$ (=2 x 144 x 5) p-values for each class of $DX - 50551 - s - 1$ generators listed and a total of $5,760$(=1,440 x 4) p-values for all the eight $DX - 50551 - s - 1$ generators. For the $DX - 40751 - s - 1$ and $DX - 50873 - s - 1$ generators, we obtained 720 (=1 x 144 x 5) p-values for each class of generators listed and a total of $2,880$(=720 x 4) p-values for all the generators listed.

For the sixteen $DX - k - s - 1$ generators listed in Table 4, the p-values for the Crush battery of tests are tabulated in Table 10. As we can see from Table 10, none of these tests produces a p-value less than 10^{-5} and only one p-value is greater than $1 - 10^{-5}$. For the Crush tests in Table 10, the proportion of p-values below 10^{-3} is 0.00139 for the $DX - 50551 - s - 1$ generators, 0.00139 for the $DX - 40751 - s - 1$ generators, and 0.00208 for the $DX - 50873 - s - 1$ generators. Besides, none of these tests produces a p-value very close to 0 or 1. We can therefore conclude that each of the sixteen $DX - k - s - 1$ generators listed in Table 4 passed the Crush battery of tests in the TestU01 suite.

According to L'Ecuyer and Simard, 2007, `DX` generators are among very few generators that can pass their stringent test suite. Their assertion is just inline with the results we have obtained above.

Finally, we evaluated the eighteen $DL/DS/DT$ generators listed in Table 5. We have not reported the results of the Small Crush battery of tests but all the generators listed in Table 5 passed these tests. For the Crush battery of tests, we obtained $1,440$ (=2 x 144 x 5) p-values for each class of $DL-k/DS-k/DT-k$ generators listed.
For the eighteen $DL/DS/DT$ generators listed in Table 5, the p-values for the Crush battery of tests are tabulated in Table 11. As we can see from this table, none of these tests produces a p-value less than 10^{-5} or greater than $1-10^{-5}$. None of these tests produce a p-value very close to 0 or 1. We can therefore conclude that each of the eighteen $DL/DS/DT$ generators listed in Table 5 passed the Crush battery of tests in the TestU01 suite.

Table 10. Results of Crush tests on DX-k-s-1 with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DX-50551-s-1 (1,440 p-values each)							
DX-50551-1-1	Count	4	0	0	1	0	1
Table4	Proportion	0.00278	0	0	0.00069	0	0.00069
DX-50551-2-1	Count	1	0	0	1	0	0
Table4	Proportion	0.00069	0	0	0.00069	0	0
DX-50551-3-1	Count	3	0	0	4	0	0
Table4	Proportion	0.00208	0	0	0.00278	0	0
DX-50551-4-1	Count	0	0	0	1	0	0
Table4	Proportion	0	0	0	0	0	0
DX-50551-s-1 (5,760 p-values in total)							
DX-50551-s-1	Count	8	0	0	7	0	0
Table4	Proportion	0.00139	0	0	0.00122	0	0
DX-40751-s-1 (720 p-values each)							
DX-40751-1-1	Count	0	0	0	2	0	0
Table4	Proportion	0	0	0	0.00278	0	0
DX-40751-2-1	Count	0	0	0	2	0	0
Table4	Proportion	0	0	0	0.00278	0	0
DX-40751-3-1	Count	0	0	0	1	0	0
Table4	Proportion	0	0	0	0.00139	0	0
DX-40751-4-1	Count	1	0	0	0	0	0
Table4	Proportion	0.00139	0	0	0	0	0
DX-40751-s-1 (2,880 p-values in total)							
DX-40751-s-1	Count	1	0	0	5	0	0
Table4	Proportion	0.00035	0	0	0.00174	0	0
DX-50873-s-1 (720 p-values each)							
DX-50873-1-1	Count	3	0	0	2	1	0
Table4	Proportion	0.00417	0	0	0.00278	0.00139	0
DX-50873-2-1	Count	1	0	0	0	0	0
Table4	Proportion	0.00139	0	0	0	0	0
DX-50873-3-1	Count	0	0	0	3	0	0
Table4	Proportion	0	0	0	0.00417	0	0
DX-50873-4-1	Count	2	0	0	0	0	0
Table4	Proportion	0.00278	0	0	0	0	0
DX-50873-s-1 (2,880 p-values in total)							
DX-50873-s-1	Count	6	0	0	5	1	0
Table4	Proportion	0.00208	0	0	0.00174	0.00035	0

Table 11. Results of Crush tests on DL-k/DS-k/DT-k generators with five starting seeds.

PRNG	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1-10^{-3}$	$> 1-10^{-4}$	$> 1-10^{-5}$
DL-k (1,440 p-values each)							
DL-40751	Count	0	0	0	2	0	0
Table5	Proportion	0	0	0	0.00139	0	0
DL-50551	Count	3	0	0	2	0	0
Table5	Proportion	0.00208	0	0	0.00139	0	0
DL-50873	Count	1	0	0	0	0	0
Table5	Proportion	0.00069	0	0	0	0	0
DS-k (1,440 p-values each)							
DS-40751	Count	3	0	0	1	0	0
Table5	Proportion	0.00208	0	0	0.00069	0	0
DS-50551	Count	2	0	0	0	0	0
Table5	Proportion	0.00139	0	0	0	0	0
DS-50873	Count	0	0	0	1	1	0
Table5	Proportion	0	0	0	0.00069	0.00069	0
DT-k (1,440 p-values each)							
DT-40751	Count	0	0	0	0	1	0
Table5	Proportion	0	0	0	0	0.00069	0
DT-50551	Count	3	0	0	3	0	0
Table5	Proportion	0.00208	0	0	0.00208	0	0
DT-50873	Count	1	0	0	1	0	0
Table5	Proportion	0.00069	0	0	0.00069	0	0

3.6 Summary

In this Chapter, we used some results from number theory as discussed in Deng, J.-J. H. Shiau, and Lu, 2012b, to proposed an efficient method to accelerate the computer searches of super-order maximum period multiple recursive generators (`MRGs`). After this extensive computer searched, and for the 32-bit `PRNGs`, with the skipping strategy described earlier, we were able to find some values of k and p for which $R(k,p)$ is a prime (`GMP`). Specifically, we found some super large values of $k's$ ($k = 40751, k = 50551$, and $k = 50873$). We used these values of $k's$ and the efficient algorithm `GMP` as proposed by Deng, 2004, to extend some existing results for some special classes of `MRGs`. These super-order `MRGs`, have orders $40751, 50551$ and 50873 and, approximate period lengths of $10^{380278.1}$, $10^{471730.6}$, and $10^{474729.3}$, respectively. In particular, after extensive computer searches, we were able to identify 114 `DX/DL/DS/DT` generators that are portable, efficient, have equi-distribution in super high dimensions, super long period lengths, and superior empirical performances. The generators we listed here are far better than some popular `MRGs` such as the `MT19937`. `MT19937` is a popular generator proposed by Matsumoto and Nishimura, 1998, it has a period length of $2^{19937} - 1 \approx 10^{6001}$ and equi-distribution up to 623 dimensions. The property of equi-distribution in dimensions up to $40751, 50551$, and 50873, can be very good for super-scale simulation studies. All the 114 generators found in this study passed the stringent Small Crush and Crush batteries of TestU01 suite.

Chapter 4

Search for "Better" Super-order Multiple Recursive Generators using Spectral Test

4.1 Introduction

In Chapter 3, we searched and implemented super-order, maximum period multiple recursive generators(`MRGs`), which have become popular in scientific research fields such as simulation and computer modeling. Among these super-order maximum period special generators was the `DX-k-s-t` generators proposed by Deng and H. Xu, 2003 and Deng, 2005. These `DX-k-s-t` generators are efficient, portable, have a long-period, and have the nice property of equi-distribution in high dimension. These generators are also special because they have very few nonzero terms. According to L'Ecuyer and Simard, 2014, the points produced by this generator have a poor lattice structure, that is, they don't perform very well on the spectral test, a theoretical test that provides some measure of uniformity in dimensions farther than the `DX-k-s-t` generator's order k.

The performance of `DX-k-s-t` generators on the spectral test could be improved by choosing multipliers that yield a "better" spectral test value. D. Knuth, 1998 notes that *"Not only do all good generators pass this test, all generators now known to be bad fail it. Thus, it is by far the most powerful test known, and it deserves particular attention"*. A major drawback of the spectral test is its computational complexity. Some traditional methods of computing this test have been proposed in the literature, see for example, Kao and Tang, 1997, D. Knuth, 1998 . Most of these proposed procedures are quite tedious and inefficient for large order (`MRGs`). Winter, 2014, propose a new method which is simple, intuitive, and efficient for some special classes of generators with few nonzero terms such as the `DX-k-s-t` generators. In this Chapter, we use this method to extend the search for ”better” `DX-k-s-t` generators beyond $k = 25013$.

In Section 4.2, we examined some of the intuitive and geometrical methods for computing the spectral test, a figure of merit for evaluating `MRGs` as expalined in Section 2.3. There are many traditional ways of calculating the spectral distance in the literature, for example, see Deng, J.-J. H. Shiau, and Lu, 2012b, L'Ecuyer, 1997, and L'Ecuyer and Couture, 1997. In Section 4.3, we briefly look at the new and simple way of computing spectral distance as proposed by Winter, 2014. In Section 4.4, we use this method to extend the search for ”better” `DX-k-s-t` generators beyond $k = 25013$ as proposed in Winter, 2014. We use the figure of merit of $d_{k+1}(k)$, the maximum ”gap” between adjacent hyperplanes in the $(k+1)$-dimensional space to compare `DX-k-s-t` super-order generators of order k. In particular, we found and listed ”better” `DX-k-s-t` super-order generators for $k = 40751, k = 50551$, and $k = 50873$. It should be noted that these are the k values that we searched and obtained in Chapter 3 of this study.

4.2 Evaluating the Spectral Test for MRGs

Definition:

For any integer x, we define $(x)_p$ by

$$(x)_p = \begin{cases} (x \; mod \, p), & if \; (x \; mod \, p) < p/2 \\ (x \; mod \, p) - p, & otherwise. \end{cases}$$

$(x)_p$ denotes the symmetric modulus operation with $-p/2 < (x)_p < p/2$.

For an r-dimensional integer vector $\mathbf{x} = (x_1, x_2, \cdots, x_r)$, we define $(\mathbf{x})_p = ((x_1)_p, (x_2)_p, \cdots, (x_r)_p)$ and $\|\mathbf{x}\|^2 = \sum_{i=1}^{r} x_i^2$.

Let

$$\Omega_{k+1}(I) = \{[X_i/p, X_{i+1}/p, \cdots, X_{i+k}/p], i = 0, 1, 2, \cdots\} \tag{4.1}$$

Spectral Test for LCGs

For the `LCG` (or `MRG` with $k = 1$), $X_i = BX_{i-1}\, mod\, p$, the maximum period is $p - 1$. Let

$$\Omega_2(I) = \{[X_{i-1}/p, X_i/p], i = 1, 2, \cdots, (p-1)\} \tag{4.2}$$

be the set of all overlapping pairs $(X_{i-1}/p, X_i/p), i = 1, 2, \cdots, (p-1)$. The ordered pair $\Omega_2(I)$ in (4.2) can be covered by several parallel lines $Bx - y = \tau, \tau = 0, \pm 1, \pm 2 \cdots$ where x corresponds to X_{i-1}/p and y corresponds to X_i/p. Therefore $BX_{i-1} - X_i = \tau p$. Its corresponding "normal vector" is $\mathbf{V} = [B, -1]$. The distance or gap between the two adjacent parallel lines is $d_2 = \frac{1}{\|\mathbf{V}\|}$ $=\frac{1}{\sqrt{1+B^2}}$. The smaller the value of d_2, the better the generator.

There are other ways to cover $\Omega_2(I)$ in (4.2) by other sets of parallel lines. For an integer c, any point in equation (4.2) will also satisfy the equation

$$cX_i = (cB)_p X_{i-1}\, mod\, p \tag{4.3}$$

The points in the lattice $\Omega_2(I)$ in (4.2) can also be covered by several parallel lines of $(cB)_p x - cy = \tau$, with the corresponding "normal vector" $\mathbf{N}_c = (c\mathbf{V})_p = [(cB)_p, -c]$. The distance or gap between the two parallel lines is $d_2 = \frac{1}{\|\mathbf{N}_c\|}$ $=\frac{1}{\sqrt{c^2+(cB)_p^2}}$. We will restrict the range of c to $0 < c < p/2$ because $\|(-c\mathbf{V})_p\| = \|(c\mathbf{V})_p\|$, that is, because of the symmetric property. Each c in this range, which need not be unique, will define family of parallel lines that cover all the points in $\Omega_2(I)$ as in (4.2). Our objective is to find the value of c such that $\|\mathbf{N}_c\|$ is the smallest among all the families of parallel lines. The smallest $\|\mathbf{N}_c\|$ will give the largest distance or gap, and hence the worst generator.

Illustrative Examples

For illustration, let's consider the `LCG[B,23]`: $X_i = BX_{i-1}\ mod\ 23$ generators. The ten primitive roots mod 23, which guarantee a maximum period of 22, are $B = (5, 7, 10, 11, 14, 15, 17, 19, 20, 21)$. We shall use a starting seed of 4 and generate 22 numbers (22 is the maximum period) for each `LCG[B,23]` generator in our illustration.

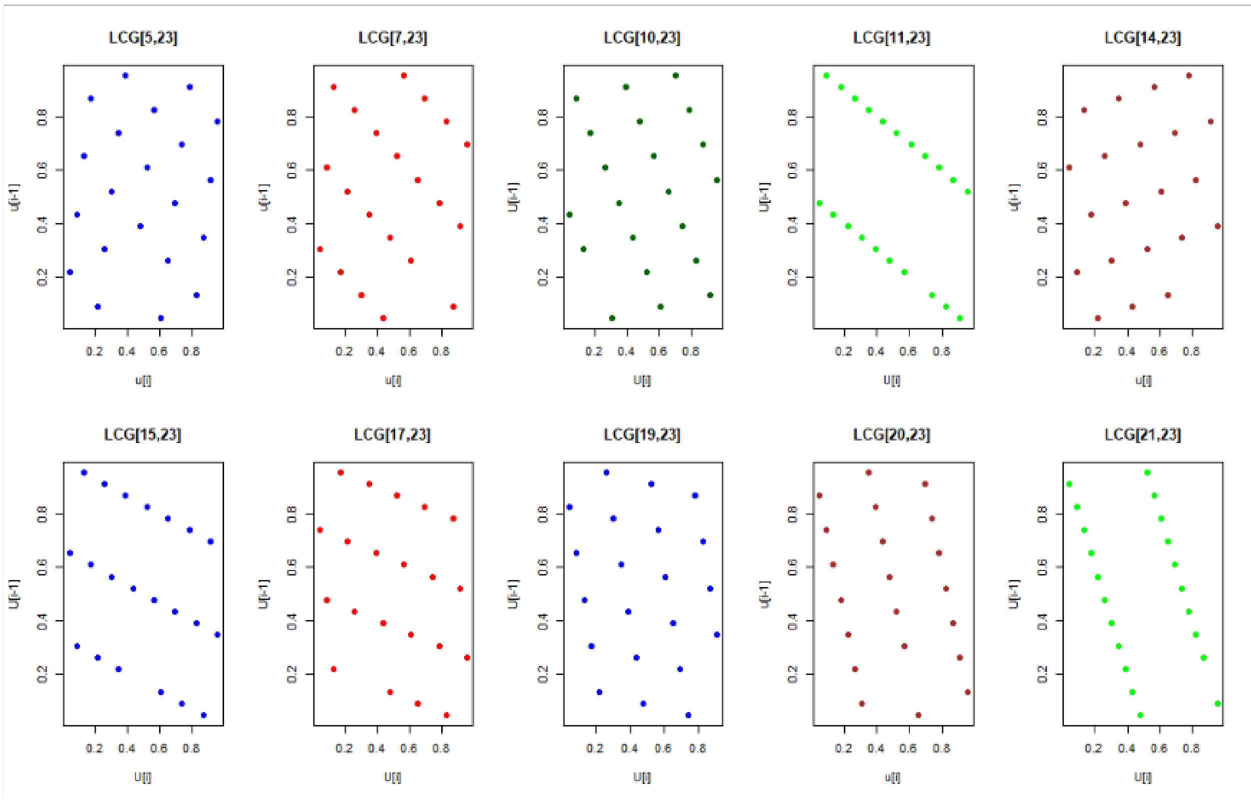

Figure 1. Lattice structure of ten maximum period `LCG[B, 23]` generators.

As we can see from Figure 1, the lattice structures for the ten maximum period `LCG[B,23]` generators are not the same. This simply means that it does not just suffices for the primitive root mod 23 to give a maximum period generator, we need to further evaluate the generators in order to get a ”better” one.

Example 1: Intuitively comparing the lattice structures of `LCG[5,23]`, `LCG[20,23]`, and `LCG[21,23]` generators, represented in Figure 2 by Figures a, b and c respectively.

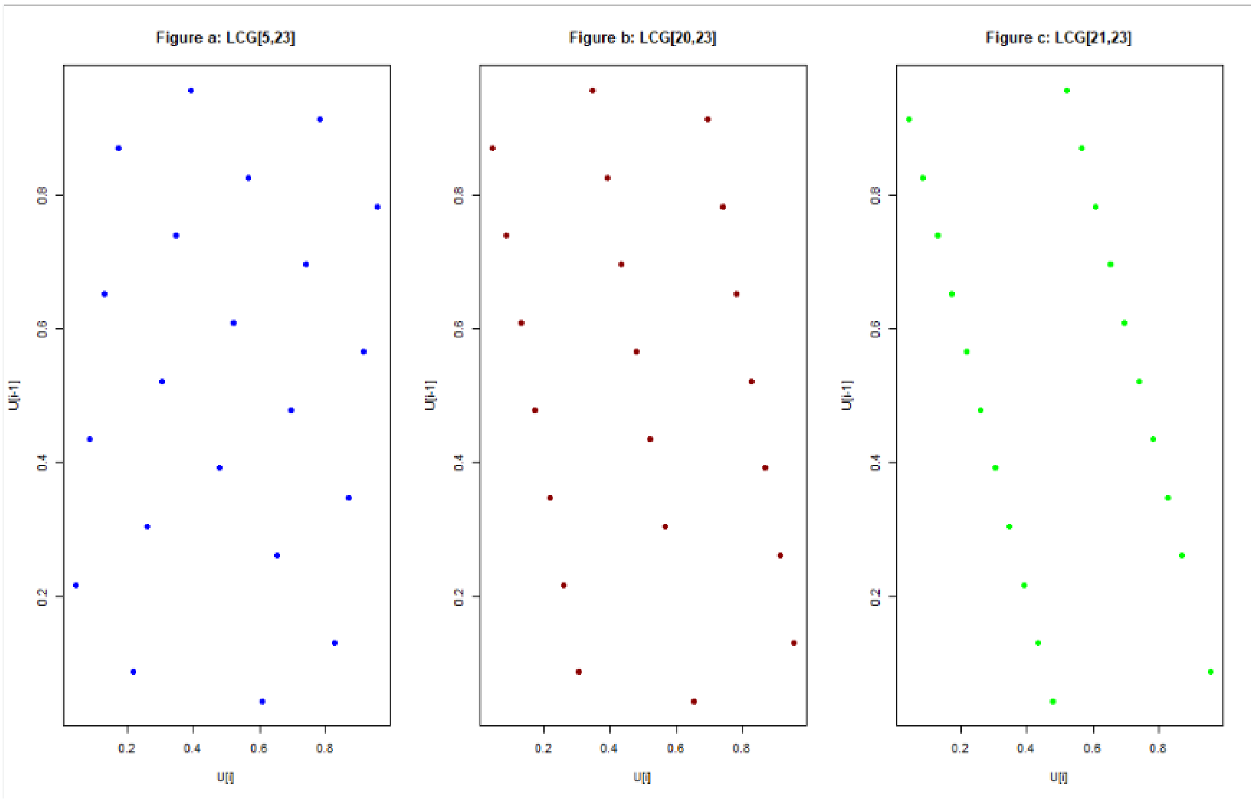

Figure 2. Lattice structure of LCG[5, 23], LCG[20, 23] and LCG[21, 23] generators.

From these lattice structures, the `LCG[5,23]` generator, Figure a (top-left), is the best of the three generators because it is more evenly distributed. Similarly, the `LCG[21,23]` generator, figure c(top-right) is the worst of the three generators.

Example 2: Comparing the lattice structures of `LCG[5,23]` and `LCG[14,23]` generators, represented in Figure 3 by Figures a and d respectively by calculating their spectral distance.

Using specific pairs of parallel lines for the overlapping pairs in each generator, we can clearly see that the generator in Figure d (top-right) is ”better”. However, we want to calculate

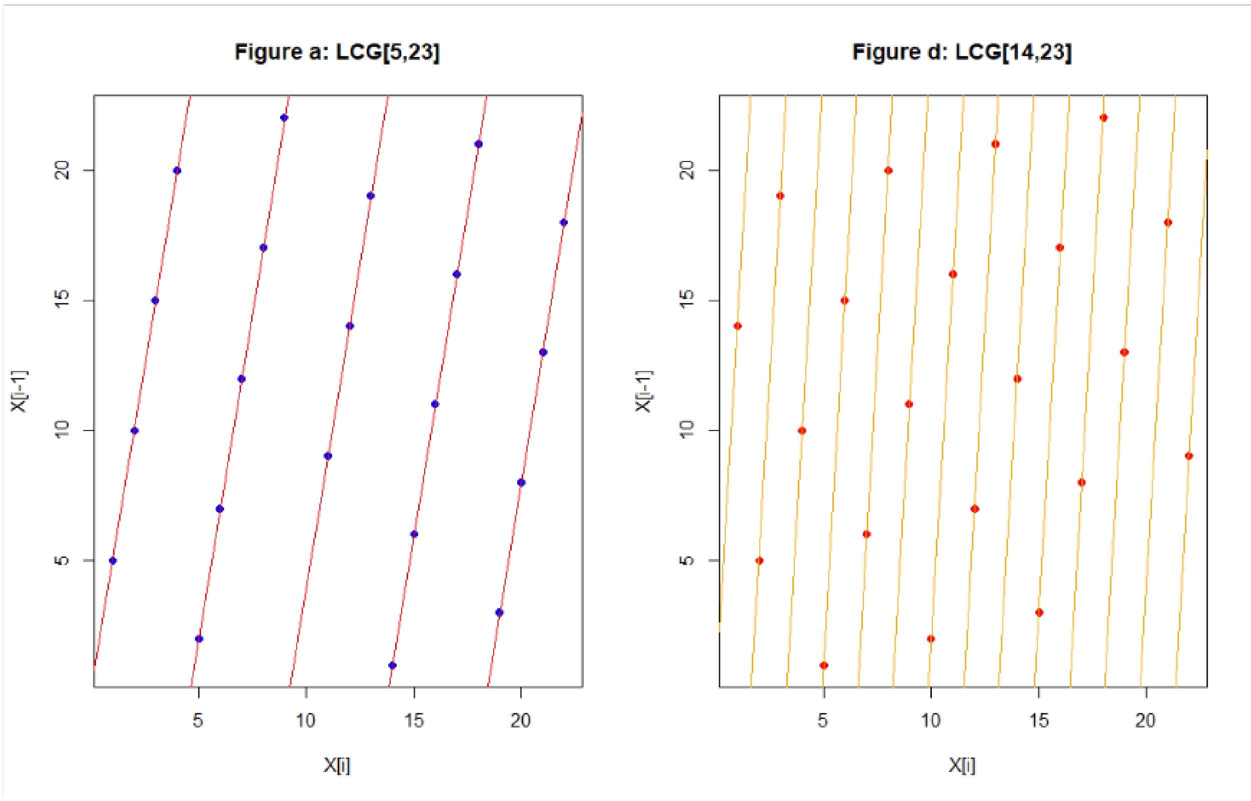

Figure 3. Lattice structure of LCG[5, 23] and LCG[14, 23] generators.

the actual spectral distance between these pairs of parallel lines in order to confirm this assertion. For `LCG[5,23]` the "normal vector" for the pairs of parallel lines is $\mathbf{N} = [5, -1]'$. The spectral distance

$$d_2 = \frac{1}{\sqrt{1 + 5^2}} = 0.19612.$$

For `LCG[14,23]` the "normal vector" for the pairs of parallel lines is $\mathbf{N} = [14, -1]'$. The spectral distance

$$d_2 = \frac{1}{\sqrt{1 + 14^2}} = 0.071247.$$

From their spectral distance, the `LCG[14,23]` generator, indicated in Figure d (top-right), is better than the `LCG[5,23]` generator, indicated in Figure a (top-left).

Example 3: Consider the families of parallel lines for $cX_i = (c14)_{23} X_{i-1} \, mod \, 23$ generators, for $0 < c < 23/2$. Our objective is to find the value of c for which the distance or gap, d_2 is the largest (Worst case of `LCG[14,23]`). The families of parallel lines are shown in Figure 4, based on the calculation of the "normal vectors" below.

$$\mathbf{N}_1 = (1\mathbf{V})_{23} = (-9,-1), \mathbf{N}_2 = (2\mathbf{V})_{23} = (5,-2), \mathbf{N}_3 = (3\mathbf{V})_{23} = (-4,-3)$$

$$\mathbf{N}_5 = (5\mathbf{V})_{23} = (1,-5), \mathbf{N}_7 = (7\mathbf{V})_{23} = (6,-7), \mathbf{N}_8 = (8\mathbf{V})_{23} = (-3,-8),$$

$$\mathbf{N}_9 = (9\mathbf{V})_{23} = (11,-9), \mathbf{N}_{11} = (11\mathbf{V})_{23} = (-7,-11)$$

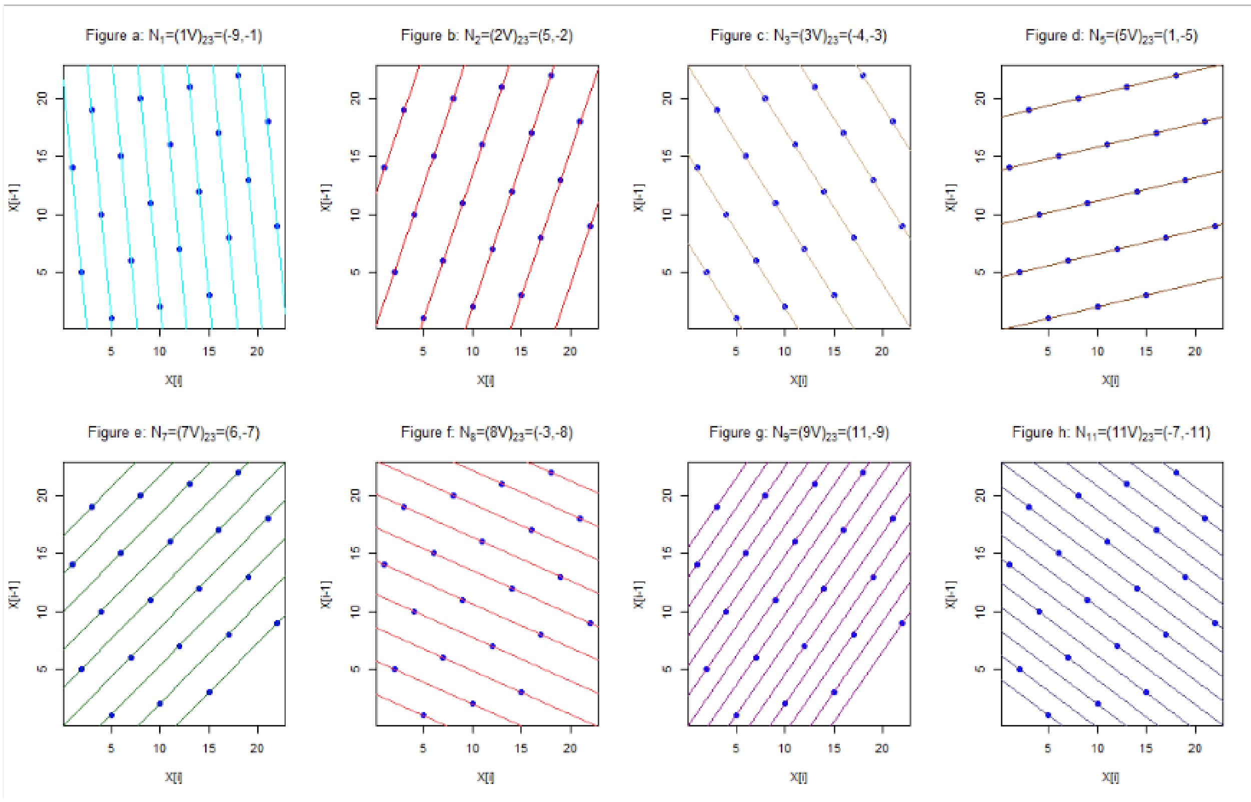

Figure 4. Families of parallel lines covering successive overlapping pairs from `LCG[14,23]` generator.

The family corresponding to $c = 2$ is the same as that of $c = 4$, the family corresponding to $c = 3$ is the same as that of $c = 6$ and the family corresponding to $c = 5$ is the same as that of $c = 10$. Thus $\mathbf{N}_4, \mathbf{N}_6$ and $\mathbf{N}_{10}$ are omitted from these calculations and from Figure 4.

The family corresponding to $c = 3$ (Figure c of the top row of Figure 4) is the one with the largest spectral distance of 0.2 between adjacent, parallel lines. As a result, it is also the family with the shortest "normal vector", $\mathbf{N}_3 = (3\mathbf{V})_{23} = (-4, -3)$, with a value of $\|\mathbf{N}_3\| = 5$. This family will produce the "worst" generator.

Spectral test for r=k+1

Points in $\Omega_{k+1}(I)$ as in (4.1), form a $(k+1)$-dimensional lattice over $[0,1)^{k+1}$, where all these points can be covered by several parallel k-dimensional hyperplanes whose "normal vector" is

$$V = [\alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1]'.$$

For any integer c, $\Omega_{k+1}(I)$ as in (4.1), will also satisfy the equation

$$cX_i = ((c\alpha_1)_p X_{i-1} + \cdots + (c\alpha_k)_p X_{i-k}) \; mod \; p, \; i \geq k.$$

Consequently, several families of parallel k-dimensional hyperplanes, associated with a chosen c, can cover all the points in $\Omega_{k+1}(I)$ as in (4.1), with "normal vector" $(c\mathbf{V})_p$. We can find the shortest "normal vector" $(c\mathbf{V})_p$, by evaluating

$$v_{k+1}^2(k) = \min_{c \neq 0} \|(c\mathbf{V})_p\|^2 = \min_{0<c<p/2} \left(\sum_{i=1}^{k} (c\alpha_i)_p^2 + c^2 \right) \tag{4.4}$$

The search space for c is limited to $0 < c < p/2$ because of the symmetric property. After computing $v_{k+1}^2(k)$, the spectral distance for the (k+1)-dimension is given by

$$d_{k+1}(k) = \frac{1}{v_{k+1}^2(k)}.$$

For any dimension r, a large $v_r(k)$ corresponds to a small $d_r(k)$, which means more uniform coverage for the r-tuples. Thus for a given MRG of order k, our wish is for $v_r(k)$ to be as large as possible. We simply call $v_r^2(k)$ the spectral test value.

To evaluate spectral test value $v_{k+1}^2(k)$ for an MRG of order k, we use the following algorithm as stated in Winter, 2014, page 35

Algorithm 1:

```
1. Initially, set $v_{min}^2 = 1 + \sum_{i=1}^{k} \alpha_i^2$ with $c = 1$
2. For $c = 2, 3, \cdots ,$    do
    (a) compute $v_c^2 = c^2 + \sum_{i=1}^{k} (c\alpha_i)_p^2$
    (b) if $v_{min}^2 > v_c^2$, then reset $v_{min}^2 = v_c^2$
    (c) if $v_{min}^2 \leq (c+1)^2$, then break; else continue with the next $c$;
3. Deliver $v_{k+1}^2(k) = v_{min}^2$
```

This algorithm is simple to implement when evaluating $v_{k+1}^2(k)$ for an `MRG(k,p)`. Nevertheless, it is not easy to generalize for dimensions greater than $k+1$.

Spectral test for $r > k$

In the previous subsection, we evaluated the spectral test value by finding the ”normal vector” and setting up the minimization problem, which was simple enough to solve with a straight forward algorithm. This method can be extended to evaluate spectral test in dimension $r = k + d$, for some integer d. It should be noted here that d stands for some dimensions farther than k. Unfortunately, we will need a more sophisticated algorithm to solve this minimization problem.

Initially, we will define d "normal vectors" of dimension r:

$$\begin{aligned}
\mathbf{V}_1 &= [\alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1, 0, 0, \cdots, 0]', \\
\mathbf{V}_2 &= [0, \alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1, 0, \cdots, 0]', \\
&\vdots \\
\mathbf{V}_d &= [0, 0, \cdots, 0, \alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1]'
\end{aligned}$$

where $\mathbf{V}_{i+1}$ is merely a simple rotation of $\mathbf{V}_i$ for $i = 1, 2, \cdots, d-1$. We can then solve the minimization problem below:

$$v_{k+d}^2(k) = \min_{(c_1, c_2, \cdots, c_d) \neq (0, 0, \cdots, 0)} \left(\| \sum_{i=1}^{d} (c_i \mathbf{V}_i)_p \|^2 \right) \tag{4.5}$$

The actual value of the minimization, $v_r^2(k)$, depends on the choices of the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ of the MRG in (2.3).
It is obvious that, $v_{k+1}^2(k) \geq v_{k+2}^2(k) \geq \cdots \geq v_{k+d}^2(k)$. It becomes increasingly difficult to solve this minimization problem when the order of the MRG, k, is larger or for large values of d. Searching for the "best" multipliers($\alpha_i's$) such that $v_r^2(k)$ is largest for given order k and modulus p is even more difficult. Winter, 2014, page 38, proposed a new method of computing the spectral test for $r > k$.

4.3 Using the LLL Algorithm to Compute the Spectral Test for $r > k$

The minimization problem of finding $v_r^2(k)$ for some dimensions $r = k+d$ is solve by using the LLL algorithm for lattice basis reduction. This algorithm was proposed by A. K. Lenstra, H. W. Lenstra, and Lovász, 1982. For the lattice reduction process, we start by creating a matrix whose rows correspond to a lattice basis of ”normal vectors”, next, we find another vector whose basis vectors are relatively short and nearly ”orthogonal”, see, for example Cohen, 1993. The vector with the shortest squared length will give the spectral value $v_r^2(k)$.

The `LLL` algorithm is an integer lattice version of Gram-Schmidt orthogonalization. fpLLL is a software for a stand-alone implementation of this algorithm but there are several implementations available in many software packages such as MAPLE, Mathematical, NTL, Python and SageMath. The algorithm proposed by Winter, 2014 is given below.

Algorithm 2: Algorithm for finding $v_{r=k+d}^2(k)$

1. (Create the initial d normal vectors.) Let $\mathbf{V}_1 = [\alpha_k, \alpha_{k-1}, \cdots, \alpha_1, -1, 0, 0, \cdots, 0]'$, where its last $d-1$ entries are all zero. Compute the remaining $d-1$ normal vectors. $\mathbf{V}_i = R^{i-1}(\mathbf{V}_1)$, for $i = 2, 3, \cdots, d$, where $R^{i-1}(\mathbf{V}_1)$ denotes $i-1$ simple rotations of $\mathbf{V}_1$ as earlier discussed.

2. (Creation of initial matrix.) Let $\mathbf{M}_0$ be an initial $d \times r$ matrix whose d rows are $\mathbf{V}_1'$, $\mathbf{V}_2', \cdots, \mathbf{V}_d'$.

3. (Remove the columns of zeros.) Remove any column of zeros from the initial matrix $\mathbf{M}_0$. Call the new matrix $\mathbf{M}_1$ whose dimension will be $d \times r^*$, where r^* is the number of columns left in the matrix. If there is no column of only zeros in $\mathbf{M}_0$, then $r^* = r$ and $\mathbf{M}_1 = \mathbf{M}_0$.

4. (Create final matrix for basis reduction.) Let $\mathbf{M}$ be an $r^* \times r^*$ matrix whose first $(r^* - d)$ rows are $p\mathbf{e}_{i(r^*)}$, where p is the modulus, $\mathbf{e}_{i(r^*)}$ is the i-th unit vector of dimension r^*, and i corresponds to the row number $i = 1, 2, \cdots, (r^* - d)$. Let the remaining d rows correspond to the rows in $\mathbf{M}_1$.

5. (Basis reduction.) Apply the `LLL` algorithm (or some other basis reduction procedure) to matrix $\mathbf{M}$ which yields a reduced matrix $\mathbf{M}^* = LLL(\mathbf{M})$. The spectral test value $v_r^2(k)$ is squared length of the shortest row vector $\mathbf{M}^*$.

Illustrative example for MRG

Consider evaluating the spectral test for five dimensions farther than $k = 7$ for the following small order `MRG`.

$$X_i = \alpha_1 X_{i-1} + \alpha_2 X_{i-2} + \alpha_3 X_{i-3} + \alpha_4 X_{i-4} + \alpha_5 X_{i-5} + \alpha_6 X_{i-6} + \alpha_7 X_{i-7} \, mod \, p, \quad i \geq 7. \quad (4.6)$$

The first corresponding normal vector will be

$$\mathbf{V}_1 = [\alpha_7, \alpha_6, \alpha_5, \alpha_4, \alpha_3, \alpha_2, \alpha_1, -1, 0, 0, 0, 0]' \quad (4.7)$$

$\mathbf{V}_2, \mathbf{V}_3, \mathbf{V}_4$ and $\mathbf{V}_5$ can be easily found by rotating the elements in $\mathbf{V}_1$ once, twice, thrice and four times, respectively. This will give

$$\mathbf{M}_0 = \begin{vmatrix} \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 & 0 \\ 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 \\ 0 & 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 \\ 0 & 0 & 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 \end{vmatrix}$$

There is no column of all zeros in M_0 to remove, thus $M_1 = M_0$. The final matrix to be submitted to `LLL` will be

$$\mathbf{M} = \begin{vmatrix} p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 \\ \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 & 0 \\ 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 & 0 \\ 0 & 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 & 0 \\ 0 & 0 & 0 & 0 & \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & -1 \end{vmatrix}$$

This example shows the simple nature of this algorithm proposed by Winter, 2014, page 38. For super-order `MRGs` (with many nonzero terms), a large matrix of basis vectors will need to be created. This algorithm can create this efficiently. Creating a large matrix with other existing methods is very tedious.

D. E. Knuth, 1981; D. Knuth, 1998, Kao and Tang, 1997, L'Ecuyer, 1997 and L'Ecuyer and Couture, 1997, are some authors who used similar methods for computing the spectral test. For comparison of this method proposed by Winter, 2014 and other existing methods, see, Winter, 2014, page 41, for details.

4.4 List of ”better” DX-k-s-t generators.

In Section 2.2, we discussed the `DX-k-s-t` generators as proposed in Deng and Lin, 2000, for `DX-k-1-1` or `FMRG` and Deng and H. Xu, 2003, for the other `DX-k-s-t` for $s = 2, 3, 4$. As discussed in Chapter 3, this class of generators are portable, efficient, and maximum period `MRGs`. As discussed in Section 4.2, solving the minimization problem as seen in equation (4.5) becomes increasing difficult when the order of k is large or for large values of d. Using Algorithm 2 in Section 4.3 (as proposed by Winter, 2014, page 38), we shall compute spectral values for `DX-k-s-1` generators. We shall use one example to illustrate this fact, with the understanding that, the rest can be done similarly.

Illustrative example for DX-k-3-1

Consider a `DX-k-3-1` with dimension $r = k + 3$ for some k. The normal vectors will be

$$\mathbf{V}_1 = [B, 0, 0, \cdots, 0, B, 0, 0, \cdots, 0, B, -1, 0, 0]'$$
$$\mathbf{V}_2 = [0, B, 0, 0, \cdots, 0, B, 0, 0, \cdots, 0, B, -1, 0]'$$
$$\mathbf{V}_3 = [0, 0, B, 0, 0, \cdots, 0, B, 0, 0, \cdots, 0, B, -1]'$$

These ”normal vectors” will form the rows of $\mathbf{M}_0$.

$$\mathbf{M}_0 = \begin{vmatrix} B & 0 & 0 & \cdots & 0 & B & 0 & 0 & \cdots & 0 & B & -1 & 0 & 0 \\ 0 & B & 0 & 0 & \cdots & 0 & B & 0 & 0 & \cdots & 0 & B & -1 & 0 \\ 0 & 0 & B & 0 & 0 & \cdots & 0 & B & 0 & 0 & \cdots & 0 & B & -1 \end{vmatrix}$$

There are many zeros between the $B's$. Therefore, there will be many zero columns in $\mathbf{M}_0$ that we can remove resulting in

$$\mathbf{M}_1 = \begin{vmatrix} B & 0 & 0 & B & 0 & 0 & B & -1 & 0 & 0 \\ 0 & B & 0 & 0 & B & 0 & 0 & B & -1 & 0 \\ 0 & 0 & B & 0 & 0 & B & 0 & 0 & B & -1 \end{vmatrix}$$

The number of columns have been drastically reduced from the initial matrix $\mathbf{M}_0$, especially if k is very large like in our super-order `DX-k-s-t` generators. To compute the minimization problem

$$v_{k+3}^2(k) = \min_{(c_1,c_2,c_3)\neq(0,0,0)} \|(c_1\mathbf{V}_1)_p + (c_2\mathbf{V}_2)_p + (c_3\mathbf{V}_3)_p\|^2 \tag{4.8}$$

we simply apply the final input matrix **M** for `LLL` to obtain our desired result.

$$\mathbf{M} = \begin{vmatrix} p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p & 0 & 0 & 0 \\ B & 0 & 0 & B & 0 & 0 & B & -1 & 0 & 0 \\ 0 & B & 0 & 0 & B & 0 & 0 & B & -1 & 0 \\ 0 & 0 & B & 0 & 0 & B & 0 & 0 & B & -1 \end{vmatrix}$$

DX-k-s-t generators and Consistency

As seen earlier in Section 4.2, $v_{k+1}^2(k) \geq v_{k+2}^2(k) \geq \cdots \geq v_{k+d}^2(k)$. With the exception of a few, most of the spectral values of `DX-k-s-t` generators with multiplier B and k not too small, remains constant for large d,

$$v_{k+1}^2(k) = v_{k+2}^2(k) = \cdots = v_{k+d}^2(k)$$

This property is known as the consistency of the spectral test values for `DX-k-s-t` generators, for details, see Winter, 2014, page 27. According to Winter, 2014, there are (rare) exceptions to this consistency property, and he buttressed this fact with some "intuitive explanations".

Winter, 2014 states that *"the consistency property is expected to hold for $v_{k+d}^2(k)$ for several dimensions beyond k until there is a change in relative relationship on the "angles" (inner-products) among theses vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_d$".*

To further consolidate this fact, we randomly tried $80,000$ $B's$ as possible multipliers for the `DX-k-s-t` generators and got a similar result. That is, $s = 1$ and $s = 4$ had 100% of the consistency property, $s = 2$ had 77.94% of the consistency property (62348 out of 80000 tries) and $s = 3$ had 95.09% of the consistency property (76068 out of 80000 tries). This result is summarize in Table 12 below.

Table 12. Consistency property of DX-k-s-t generators with 80,000 tries.

s	80,000 tries	Percentage of consistency	Percentage of nonconsistency
1	80,000	100	0
2	62,348	77.94	22.06
3	76,068	95.09	4.91
4	80,000	100	0

Since the `MRGs` with the consistency property will have exactly the same spectral value in several dimensions farther than k, it will be easier to rank them for a given order k (and modulus p). For `MRGs` in general, an `MRG` with the best spectral test value in dimension $k + 1$ can have a worst spectral value in dimension $k+2$, for example, see, D. Knuth, 1998. However, this is not the case for `MRGs` that have the consistency property. For `MRGs` having this property, the one with the best spectral test value in dimension $k+1$ will be the best for several dimensions farther than k.

As seen in Chapter 3, we found several super-order `DX-40751-s-1, DX-50551-s-1`, and `DX-50873-s-1` generators. We shall used the algorithm proposed by Winter, 2014, page 38 as stated Section 4.3 to search for "better" super-order `DX-k-s-1` generators for $k = 40751, 50551, 50873$, with reasonably good spectral distances $d_{k+1}(k)$. We start by screening potential multipliers for spectral distance $d_{k+1}(k)$ below a pre-specified bound (for example, $3.0e-05$ or $2.75e-05$) and verify that the generator possessed the consistency property. When this is done, we proceed to check whether the generator had the maximum period as we did in Chapter 3, using the methods proposed by Deng, 2004, Deng, J.-J. H. Shiau, and Lu, 2012b, and Deng, J.-J. H. Shiau, and Lu, 2012a. Table 13 below list "better" super-order `DX-k-s-t` generators. In this study, we multiply the spectral distance by 10^5 for easy visualization.

Illustrative Example with super-order DX-50873-4-1 generator.

As an example, we used $DX-50873-4-1$ for illustration with a "better" B value of 1073544618 and a spectral distance of 1.58374.

The primitive polynomial found after a search time of 95701 seconds, is:

$$f(x) = x^{50873} - 1073544618 * x^{50872} - 1073544618 * x^{33915} - 1073544618 * x^{16957} - 1073544618$$

The super-order "better" maximum period $DX-50873-4-1$ generator is:

$$X_i = 1073544618(X_{i-1} + X_{i-16958} + X_{i-33916} + X_{i-50873})\ mod\ 2146123787, \quad i \geq 50873 \quad (4.9)$$

4.5 Summary

For an MRG(k,p) with maximum period, we consider an r-tuple $X_i, X_{i+1}, \cdots, X_{i+r-1}$ with the index i running through the entire period $(p^k - 1)$. For $r \leq k$, generators in MRG(k,p) are quite close to an ”ideal” generator; only the all-zero tuple is generated one time less than the other r-tuples, by the equi-distribution property. For $r > k$, the r-dimensional points lie on a relatively small family of equidistant parallel hyperplanes in a high dimensional space. In this Chapter, we calculated the spectral distance between these hyperplanes. For LCGs, MRGs and other generators with lattice structures, the spectral test is the most perfect figure of merit. D. Knuth, 1998, notes that *”Not only do all good generators pass this test, all generators now known to be bad fail it. Thus, it is by far the most powerful test known, and it deserves particular attention”.* A drawback of the spectral test is its computational complexity. Using the LLL algorithm, Winter, 2014, proposed a simple and intuitive method for calculating spectral distance. Using this method, we extended the search for ”better” DX-k-s-t farther than $k = 25013$ as proposed in Winter, 2014. In particular, we searched and listed ”better” super-order DX-k-s-t super-order generators for $k = 40751, k = 50551$, and $k = 50873$.

Table 13. List of ”better” super-order `DX-k-s-t` generators with $B < 2^{19}$, $B < 2^{20}$, and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)(\times 10^5)$.

Table 13a

List of ”better” DX-k-1-1 with $B < 2^{20}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)(\times 10^5)$

k	p	$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
40751	2146593347	949211	1.76933	1073724261	1.92831
50551	2146725226	541542	1.88551	1073390951	1.76445
50873	2146123787	1004567	2.00597	1073624018	2.95118

Table 13b

List of ”better” DX-k-2-1 with $B < 2^{20}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)(\times 10^5)$

k	p	$B < 2^{20}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
40751	2146593347	910659	2.46292	1073500698	2.15977
50551	2146725226	536124	2.17392	1073724894	2.4113
50873	2146123787	943659	2.36297	1073653794	2.50025

Table 13c

List of ”better” DX-k-3-1 with $B < 2^{19}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)(\times 10^5)$

k	p	$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
40751	2146593347	433849	2.08802	1073679636	1.86653
50551	2146725226	515561	2.201	1073646955	2.13737
50873	2146123787	470516	1.71238	1073705303	1.91587

Table 13d

List of ”better” DX-k-4-1 with $B < 2^{19}$ and $B < 2^{30}$ and their spectral distance $d_{k+1}(k)(\times 10^5)$

k	p	$B < 2^{19}$	$d_{k+1}(k)$	$B < 2^{30}$	$d_{k+1}(k)$
40751	2146593347	495476	1.67846	1073695069	1.74335
50551	2146725226	461111	1.57481	1073646756	1.93132
50873	2146123787	289642	1.89455	1073544618	1.58374

Chapter 5

Extension of a Special Class of Large order Multiple Recursive Generators (`MRGs`) with Many Nonzero Terms to Super-order Generators.

5.1 Introduction

In Section 2.2, we discussed some special classes of `MRGs`. Among these, were some with many nonzero terms. In particular, we discussed the `DL`, `DS`, and `DT` generators, as special classes of `MRGs` with many nonzero terms as proposed by Deng and his group of co-authors, for details see, Deng, Li, J.-J. Shiau, and Tsai, 2008, Deng, J.-J. Shiau, and Tsai, 2009. These groups of generators are efficient, portable, and maximum order `MRGs` and are implemented efficiently using higher order recurrence with few nonzero terms.

In Chapter 3, using the values of k that we obtained from the primality testing of $R(k,p) = (p^k - 1)/(p-1)$, we extended these `DL`, `DS`, and `DT` generators to super-order generators in super-high dimensions, by applying algorithm `GMP`, proposed by Deng, 2004. In this Chapter, we will discuss and extend another special class of large order `MRGs` with many nonzero terms that have an efficient and parallel implementation. This special class is known as the `DW-k` generator and was proposed by Winter, 2014. It is another class of efficient, portable, and maximum period `MRG`. This class of generators is defined in Section 5.2. Using the values of $k = 40751, k = 50551$, and $k = 50873$, that we obtained in Chapter 3, we will search for new `DW-k` generators. Thus, our results will extend the `DW-k` generators, farther than the order $k = 25013$, to super-order `DW-k` generators.

5.2 `DW-k` generator

Definition 1. The `DW-k` generator is a large order `MRG` with many nonzero terms defined with the following characteristic polynomial modulo p

$$f(x) = (x - B)(x - C)^{k-1} - ABx^{k-2} \, mod \, p \tag{5.1}$$

where A, B, and C are suitably chosen nonzero integers over $\mathbb{Z}_p$, such that $f(x)$ is a k-th degree primitive polynomial modulo p.

The expansion of this polynomial using the binomial theorem yields many nonzero multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$, each of which is a function of the order k and the parameters A, B, or C are suitably chosen nonzero integers over $\mathbb{Z}_p$.

$$\alpha_i = \begin{cases} ((k-1)C + B) \; mod \, p, & \text{for } i = 1 \\ \left(AB - \binom{k-1}{2}C^2 - (k-1)BC\right) \; mod \, p, & \text{for } i = 2 \\ \left((-1)^{i-1}\binom{k-1}{i}C^i + \binom{k-1}{i-1}BC^{i-1}\right) \; mod \, p, & \text{for } i = 3, 4, \cdots, k-1 \\ (-1)^{k-1}BC^{k-1} \, mod \, p & \text{for } i = k \end{cases} \tag{5.2}$$

For details, see Winter, 2014.

When specifying the order k, parameters A, B, C, and prime modulus p, the `DW-k` generator is denoted as `DW(k; A, B, C; p)`, because the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ are fully specified by the order k and the parameters A, B, C. Worthy of note is the fact that A, B, C are parameters of the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$.

The characteristic polynomial in equation (5.1) has three special attributes. The first attribute is that it yields many nonzero multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$, for the `MRG` recursion stated in equation (2.3). The next attribute is that there is an efficient matrix congruential generator (`MCG`) that shares the same characteristic polynomial as the `DW-k` generator. This `MCG` is define in the next Section and its efficient implementation is given. This `MCG's` efficient implementation will be used for the efficient and parallel implementation of the `DW-k` generators. The final attribute is that, only the multiplier α_2 in equation (5.2) is a function of A. To simplify the search method for super-order maximum period `DW-k` generator, we will take advantage of this third attribute.

5.3 Implementation of DW-k generators using MCG

The `MCG` that shares the same characteristic polynomial as the `DW-k` generator, is defined in this Section. Furthermore, we will use this `MCG` to implement the `DW-k` generator, efficiently and in parallel.

Consider the matrix congruential generator with the following multiplier matrix

$$\mathbf{B} = \begin{pmatrix} B & 0 & 0 & \cdots & 0 & A \\ B & C & 0 & \cdots & 0 & 0 \\ B & C & C & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ B & C & C & \cdots & C & 0 \\ B & C & C & \cdots & C & C \end{pmatrix} \tag{5.3}$$

with recursion as defined as in equation (2.25). Its characteristic polynomial is $f_{\mathbf{B}}(x) = det(x\mathbf{I} - \mathbf{B}) \, mod\, p$, as defined in equation (2.26).

Winter, 2014, proved that the `MCG` used with multiplier **B** in equation (5.3) has the same characteristic polynomial $f_{\mathbf{B}}(x)$ as the one that defines the `DW-k` generator in (5.1), for details, see, Winter, 2014, Lemma 2, Theorem 2.

Therefore, this `MCG` with matrix multiplier **B** shares the same characteristic polynomial $f(x) = (x - B)(x - C)^{k-1} - ABx^{k-2}$ as the `DW-k` generator.

We will make use of this fact to show that the `DW-k` generator can be implemented efficiently and in parallel using this `MCG`. Using the multiplier matrix **B**, the iterative equation (2.25) can be rewritten as follows:

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ X_{i,3} \\ X_{i,4} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} BX_{i-1,1} + AX_{i-1,k} \\ BX_{i-1,1} + CX_{i-1,2} \\ BX_{i-1,1} + CX_{i-1,2} + CX_{i-1,3} \\ BX_{i-1,1} + CX_{i-1,2} + CX_{i-1,3} + CX_{i-1,4} \\ \vdots \\ BX_{i-1,1} + CX_{i-1,2} + \cdots + CX_{i-1,k} \end{pmatrix} mod\, p, \quad i \geq 1 \tag{5.4}$$

This can be implemented efficiently as

$$\begin{pmatrix} X_{i,1} \\ X_{i,2} \\ X_{i,3} \\ X_{i,4} \\ \vdots \\ X_{i,k} \end{pmatrix} = \begin{pmatrix} BX_{i-1,1} + AX_{i-1,k} \\ BX_{i-1,1} + CX_{i-1,2} \\ X_{i,2} + CX_{i-1,3} \\ X_{i,3} + CX_{i-1,4} \\ \vdots \\ X_{i,k} + CX_{i-1,k} \end{pmatrix} mod\, p, \quad i \geq 1 \tag{5.5}$$

From equation (5.5), we can see that the generated output $X_{i,1}$ and $X_{i,2}$ are exclusively generated from numbers in the previously generated output vector $\mathbf{X}_{i-1}$ and $X_{i,j}$ for $j = 3, 4, \cdots, k$ is the sum of the previous number just generated in the current output vector $\mathbf{X}_i$ and a multiple of a number in the previous output vector $\mathbf{X}_{i-1}$. It is obvious that, the iteration in (5.5) does not need as many multiplication and addition as those in (5.4). A direct implementation using (5.4) would have been less efficient, because more multiplications and additions are involved. Thus, the implementation using (5.5) is more efficient.

According to Winter, 2014, ”*additional efficiency on some compilers might be gained if we let C be a power of 2, that is, we can let $C = 2^e$ for some positive integer e.*” In this case, empirical evidence suggest that for $2^e, e \leq 4$ should be avoided. For example, Deng, 2016, explained that, *""small" numbers tend to follow "small" numbers which will create a bad run property."* This simply means that, generally, C should not be too small for given A, B, and order k.

In Section 2.4, we discussed the relationships between `MCGs` and `MRGs`. Following this discussion, since the `MCG` defined by **B** in (5.3) and `DW-k` generator shares the same characteristic polynomial, we can confirm that the generated vector sequence ($\mathbf{X}_i, i \geq 0$) from this `MCG` satisfies the iteration for `DW-k` generator,

$$\mathbf{X}_i = (\alpha_1 \mathbf{X}_{i-1} + \alpha_2 \mathbf{X}_{i-2} + \cdots + \alpha_k \mathbf{X}_{i-k}) \, mod \, p, \quad i \geq k \tag{5.6}$$

where the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ are as defined in (5.2). This means that each of the k sequences taken from each of the k rows in (5.6) can be viewed as k copies of the same `DW-k` generator with different starting seeds. Thus, our recommendation is that k numbers should be generated at a time from the efficient iteration in (5.5) and assign each number to one of the k `CPUs`. Empirical performance suggests that the `MCG` defined by **B** in (5.3) has its own advantages as a standalone generator where numbers can be generated one at a time.

Equipped with this background information, we will now focus our attention in finding the values $A, B,$ and C, such that $f(x)$ as defined in (5.1), is a primitive polynomial over $\mathbb{Z}_p$. As mentioned earlier, we will use a similar method to the one we used in Chapter 3.

5.4 Search for Super-order `DW-k` generators

In Chapter 3, we discussed that Alanen and D. E. Knuth, 1964 and D. Knuth, 1998, proposed an algorithm with three sets of necessary and sufficient conditions for $f(x)$ as defined in (2.3) to be a primitive polynomial. One of the major shortcoming of their algorithm is finding the complete factorization of $R(k, p) = (p^k - 1)/(p - 1)$, when k and/or p is large. To bypass this shortcoming of factorizing a huge number like $R(k, p) = (p^k - 1)/(p - 1)$, there are two common ways, either (i) for a given p one can find k such that $R(k,p)$ is relatively easy to factorize, usually because $R(k,p) = n \times M$, where M is a huge prime factor and n is a product of several "small" primes factors (say, 10^9) or (ii) for a known prime k, one can find a prime p such that $R(k,p)$ is a prime (`GMP`), for details, see, Deng and Bowman, 2017.

If we know the complete factorization of $R(k,p)$ for a given order k and modulus p, we proceed to search for multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ such that the rest of the conditions of their algorithm are met. We can simplify this procedure for the `DW-k` generator. As we mentioned earlier, in (5.2) only the multiplier α_2 is a function of order k and the parameters $A, B,$ and C over $\mathbb{Z}_p$. The rest of the multipliers in (5.2) are completely specified by the order k and the parameters B and C. Thus, for a given order k and modulus p, once we know that $\alpha_k = (-1)^{k-1}BC^{k-1}$ is a primitive root modulo p, we can fix multipliers $\alpha_1, \alpha_3, \alpha_4, \cdots, \alpha_k$ and search for α_2 until the characteristic polynomial of the `DW-k` generator is primitive. In other words; once we find B and C such that $\alpha_k = (-1)^{k-1}BC^{k-1}$ is a primitive root modulo p, we can fix B and C and just search for A until $f(x)$ in (5.1) is a primitive polynomial, for details, see Winter, 2014.

Using the first way of factorizing $R(k,p)$ as mentioned earlier, Deng and H. Q. Xu, 2005 and Deng, J.-J. H. Shiau, and Lu, 2012a, found k-th degree primitive polynomial for several k ($k = 47, k = 643, k = 1597, k = 7499, k = 20897$) and modulus $p = 2^{31} - 1$. Winter, 2014, used these values of k, for $p^{31} - 1$ and $2^5 \leq C \leq 2^9$, with four values of B, and search for A such that `DW(k; A, B, C; p)` generator is of maximum period. He obtained 100 `DW-k` generators as listed in Winter, 2014, Pages 80-82.

The second approached of bypassing the factorization of $R(k,p)$ is what we adopted in this study. That is, for a prime k, we find p for which $R(k,p) = (p^k - 1)/(p-1)$ is a prime (`GMP`). For a 32-bit `RNG,` for a prime k, we find v such that $p = 2^{31} - v$ and $R(k,p)$ are primes. In Chapter 3, we used this method and found super large values of $k's$ ($k = 40751, k = 50551, k = 50873$) as shown in Table 1.

Winter, 2014, found and listed `DW-k` generators for k up to 25013. Using a similar method to the one we used in Chapter 3, we will use these super large values of k to extend the search for `DW-k` generators to super-order generators. That is, for our super large order k, prime modulus $p = 2^{31} - v$, $2^5 \leq C \leq 2^9$ (for additional efficiency), and 2 values of B, we search for A (with 2 different maximum limits) such that `DW(k; A, B, C; p)` generator achieves the super maximum period. The list of 22 super-order `DW(k; A, B, C;` $p = 2^{31} - v$`)` for 3 values of k and 5 values of C are given in Table 14. There are 3 cases in which there is no A (failed search) for which $f(x)$ as in (5.1) is a primitive polynomial(see, the 3 blank spaces in Table 14).

Table 14. List of A for 22 DW($k; A, B, C = 2^e : p = 2^{31} - v$); k=40751, 50551, 50873 and e=5, 6, 7, 8, 9.

k	v	B	C				
			32	64	128	256	512
40751	890301	20000	75040		41383	38231	43873
40751	890301	30001	85221	51148	80990	39409	57227
50551	758421	30004		61061	98532	94652	61243
50873	1359861	20000	75712	59812		79297	12800
50873	1359861	30001	94637	31031	93060	26174	43496

5.5 Evaluation

Empirical Evaluation

As we mentioned earlier in Section 5.3, efficiently implementing the `DW-k` generators in parallel across k processors requires generating k numbers at a time from the iteration in equation (5.5) and assigning each of these numbers to one of the k processors. As we showed in Section 3.4, for the other special cases of `MRGs,` maximum period `MRGs` have excellent empirical performance when the generated output are taken in successive sequences. These special class of `MRGs` are one of the few kinds of random number generators that are able to pass all the stringent batteries of tests in the TestU01 package proposed by L'Ecuyer and Simard, 2007. As we saw in Section 2.2, D. Knuth, 1998, noted that concerning output from generated large order maximum period `MRGs`, *"all known evidence indicates that the result will be a very satisfactory source of randomness".*

With this in mind, we can conclude that the k sequences of `DW-k` generator has excellent empirical performance.

The above assertion is also satisfactory when generating numbers one at a time from the `MCG` of the iteration in (5.5). Of course we mentioned in Section 5.3 that this `MCG` has its own merits as a standalone generator. For each combination of super-order k, modulus p, and parameters A, B, C, listed in Table 14, we generated numbers one at a time and apply the sequential output to Small Crush and Crush batteries of the stringent empirical tests in the TestU01 package. It should be noted here that this a similar procedure to the one we used in Section 3.4. We will use 5 starting seeds, following the approached used in Deng, J.-J. H. Shiau, and Lu, 2012b, and only the p-values produced outside of $[10^{-5}, 1 - 10^{-5}]$ will be reported.

For Small Crush battery of test, the total number of p-values is $1650(22 \times 5 \times 15)$. The empirical performance for this test is satisfactory, though we did not report it in this study. We will report results for the Crush battery of tests.

For the Crush battery of tests, the total number of p-values produced is $15,840(22 \times 5 \times 144)$. The proportion of these p-values is summarized in Table 15 below.

Only 2 out of $15,840$, p-values, with proportions of 0.00006, respectively, is outside the range $[10^{-5}, 1-10^{-5}]$. Furthermore, none of these p-values is outside the range, $[10^{-10}, 1-10^{-10}]$, as stated in L'Ecuyer and Simard, 2007. Therefore, these results lead us to the same conclusion. That is, we have strong empirical evidence that even generating numbers one at a time from the `MCG` in iteration (5.5) yields a satisfactory source of randomness.

Nevertheless, to implement `DW-k` generators, numbers must be generated k at a time. Thus, generating numbers k at a time is highly recommended.

Theoretical Evaluation

As defined in Section 2.3, the spectral test for an `MRG` is a theoretical test that measures the uniformity of the `MRG` farther than its k dimension. For dimension $r \leq k$, the nice equi-distribution

Table 15. Results of Crush tests on DW-k generators with five starting seeds.

Dw-k	p-value	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$	$> 1 - 10^{-3}$	$> 1 - 10^{-4}$	$> 1 - 10^{-5}$
DW-40751(2,880 p-values each)							
B=20000	count	0	1	0	0	0	0
	proportion	0	0.00035	0	0	0	0
DW-40751(3,600 p-values each)							
B=30001	count	1	1	1	6	0	0
	proportion	0.00028	0.00028	0.00028	0.00167	0	0
DW-50551 (2,880 p-values each)							
B=30004	count	0	2	0	0	1	0
	proportion	0	0.00069	0	0	0.00035	0
DW-50873(2,880 p-values each)							
B=20000	count	3	0	0	0	0	1
	proportion	0.00104	0	0	0	0	0.00035
DW-50873 (3,600 p-values each)							
B=30001	count	1	0	0	5	0	0
	proportion	0.00028	0	0	0.00139	0	0
DW-k (15,840 p-values in total)							
DW-k	count	5	4	1	11	1	1
Total	proportion	0.00032	0.00025	0.00006	0.00069	0.00006	0.00006

property [Lidl and Niederreiter, 1994, Theorem 7.43] for maximum period MRGs guarantee very good uniformity.

In Chapter 4, we discussed the measure of this uniformity for dimensions $r = k + d$. Using the traditional methods, calculation of spectral test becomes quite tedious and very inefficient when the order of k is large. We used the simple and intuitive algorithm proposed by Winter, 2014, Page 38 to bypass this difficulty for MRGs with few nonzero terms, like the DX generators. However, for the DW-k generator with many nonzero terms, it is not likely that this simple and intuitive algorithm used in Chapter 4 will be any more efficient. Thus, computing the spectral test for super-order MRGs with many nonzero terms is computationally demanding.

For maximum period MRGs with good spectral test performance in dimensions beyond k, L'Ecuyer, 1997 stated a necessary (but not sufficient) condition that the sum of squares of the multipliers, $\sum_{i=1}^{k} \alpha_i^2$ should be large. Thus, one would prefer MRGs with large $\sum_{i=1}^{k} \alpha_i^2$. Since the DW-k generator has many nonzero multipliers, it is likely to have excellent spectral test performance. Further more, as stated in Section 2.2, there is strong statistical justification for MRGs with many nonzero terms as well. An MRG will become ”more and more uniform” with larger number of nonzero terms in the summation, for details, see, for example, Deng, 2016, and Deng and George, 1990.

Concerning timing, Winter, 2014, showed that DW-k generator is very efficient by comparing it with the Mersenne-Twister (MT19937), Matsumoto and Nishimura, 1998, and the combined MRG, (MRG32k3a), L'Ecuyer, 1999.

5.6 Summary

In this Chapter, we examined the DW-k generator as proposed by Winter, 2014, as another special class of MRG with many nonzero terms whose iteration can be implemented efficiently and in parallel, using a k-th order MCG sharing the same characteristic polynomial. We extended the work-done in Winter, 2014, by searching for super-order DW-k generators using our super large k values ($k = 40751, k = 50551, k = 50873$) that we obtained in Chapter 3. Using extensive computer searches, we found and listed 22 super-order maximum period DW(k; A, B, C, $p = 2^{31} - v$) generators. We evaluated the empirical performance of these found generators using the Small Crush and Crush batteries of tests in the TestU01 suite, though we reported only the Crush results. Our super-order generators are found to have excellent empirical performance. It is likely that their theoretical performance on the spectral test are good as well.

Chapter 6

Conclusion

In this study, we used some results from number theory as discussed in Deng, J.-J. H. Shiau, and Lu, 2012b, to proposed an efficient method to accelerate the computer search of super-order maximum period multiple recursive generators (MRGs). After this extensive computer searched, and for the 32-bit PRNGs, we were able to find some values of k and p for which $R(k, p)$ is a prime (GMP). In particular, we found super-order MRGs, with orders $40751, 50551$, and 50873 and, approximate period lengths of $10^{380278.1}$, $10^{471730.6}$, and $10^{474729.3}$, respectively. We used these values of $k's$ and the efficient algorithm GMP as proposed by Deng, 2004, to extend some existing results for some special classes of MRGs. In particular, after extensive computer searches, we were able to identify 114, DX/DL/DS/DT generators that are portable, efficient, have equi-distribution in super high dimensions, super long period lengths, and superior empirical performances. The generators we listed here are far better than some popular MRGs such as the MT19937. MT19937 is a popular generator proposed by Matsumoto and Nishimura, 1998, it has a period length of $2^{19937} - 1 \approx 10^{6001}$ and equi-distribution up to 623 dimensions. The property of equi-distribution in dimensions up to $40751, 50551$, and 50873, can be very good for super-scale simulation studies. All the 114 generators found in this study passed the stringent Small Crush and Crush batteries of TestU01 suite.

For an MRG(k,p) with maximum period, we consider an r-tuple $X_i, X_{i+1}, \cdots, X_{i+r-1}$ with the index i running through the entire period $(p^k - 1)$. For $r \leq k$, generators in MRG(k,p) are quite close to an ”ideal” generator; only the all-zero tuple is generated one time less than the other r-tuples, by the equi-distribution property. For $r > k$, the r-dimensional points lie on a relatively small family of equidistant parallel hyperplanes in a high dimensional space. In this study, we calculated the spectral distance between these hyperplanes. For `LCGs, MRGs,` and other generators with lattice structures, the spectral test is the most perfect figure of merit. D. Knuth, 1998, notes that *”Not only do all good generators pass this test, all generators now known to be bad fail it. Thus, it is by far the most powerful test known, and it deserves particular attention”.* A drawback of the spectral test is its computational complexity. Using the `LLL` algorithm, Winter, 2014, proposed a simple and intuitive method for calculating spectral distance. Using this method, we extended the search for ”better” `DX-k-s-t` beyond $k = 25013$ as proposed in Winter, 2014. In particular, we searched and listed, 24, ”better” super-order `DX-k-s-t` generators for $k = 40751, k = 50551,$ and $k = 50873$.

Finally, we discussed the `DW-k` generator as proposed by Winter, 2014, as another special class of `MRG` with many nonzero terms whose iteration can be implemented efficiently and in parallel, using a k-th order `MCG` sharing the same characteristic polynomial. We extended the work-done in Winter, 2014, by searching for super-order `DW-k` generators using our super large k values ($k = 40751, k = 50551, k = 50873$) that we obtained in Chapter 3. Using extensive computer searches, we found and listed, 22, super-order maximum period `DW(k; A, B, C,` $p = 2^{31} - v$`)` generators. We evaluated the empirical performance of these found generators using the Small Crush and Crush batteries of tests in the TestU01 suite. Our super-order generators are found to have excellent empirical performance. It is likely that their theoretical performance on the spectral test are good as well.

Bibliography

[1] J. D. Alanen and D. E. Knuth, "Tables of finite fields," *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, vol. 26, no. 4, pp. 305–328, 1964, ISSN: 0581572X. [Online]. Available: `http://www.jstor.org/stable/25049338`.

[2] D. Knuth, "Art of computer programming, volume 2: Seminumerical algorithms," vol. 2, 1998.

[3] L.-Y. Deng and H. Q. Xu, "Design, search and implementation of high-dimension, efficient, long-cycle and portable uniform random variate generator," 2005.

[4] L.-Y. Deng, J.-J. H. Shiau, and H. H.-S. Lu, "Large-order multiple recursive generators with modulus 231- 1," *INFORMS Journal on Computing*, vol. 24, no. 4, pp. 636–647, 2012.

[5] P. L'Ecuyer, F. Blouin, and R. Couture, "A search for good multiple recursive random number generators," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 3, no. 2, pp. 87–98, 1993.

[6] L.-Y. Deng, "Generalized mersenne prime number and its application to random number generation," *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pp. 167–180, 2004.

[7] L.-Y. Deng, J.-J. H. Shiau, and H. H.-S. Lu, "Efficient computer search of large-order multiple recursive pseudo-random number generators," *Journal of Computational and Applied Mathematics*, vol. 236, no. 13, pp. 3228–3237, 2012, ISSN: 0377-0427. DOI: `https://doi.org/10.1016/j.cam.2012.02.023`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0377042712000805`.

[8] B. R. Winter, *Design, Search and Implementation of Improved Large Order Multiple Recursive Generators and Matrix Congruential Generators.* The University of Memphis, 2014.

[9] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*. Cambridge university press, 1994.

[10] D. H. Lehmer, "Mathematical methods in large-scale computing units," *Annu. Comput. Lab. Harvard Univ.*, vol. 26, pp. 141–146, 1951.

[11] G. MARSAGLIA, "The structure of linear congruential sequences," in *Applications of Number Theory to Numerical Analysis*, S. Zaremba, Ed., Academic Press, 1972, pp. 249–285, ISBN: 978-0-12-775950-0. DOI: `https://doi.org/10.1016/B978-0-12-775950-0.50013-3`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/B9780127759500500133`.

[12] P. L'Ecuyer and R. Simard, "Testu01: A c library for empirical testing of random number generators," *ACM Trans. Math. Softw.*, vol. 33, no. 4, Aug. 2007, ISSN: 0098-3500. DOI: `10.1145/1268776.1268777`. [Online]. Available: `https://doi.org/10.1145/1268776.1268777`.

[13] L.-Y. Deng and H. Xu, "A system of high-dimensional, efficient, long-cycle and portable uniform random number generators," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 13, no. 4, pp. 299–309, 2003.

[14] L.-Y. Deng, "Recent developments on pseudo-random number generators and their theoretical justifications," *J Chin Stat Assoc*, vol. 54, pp. 154–179, 2016.

[15] L.-Y. Deng and E. O. George, "Generation of uniform variates from several nearly uniformly distributed variables," *Communications in Statistics-Simulation and Computation*, vol. 19, no. 1, pp. 145–154, 1990.

[16] A. Grube, "Mehrfach rekursiv-erzeugte pseudo-zufallszahlen," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 53, no. 12, T223–T225, 1973.

[17] P. L'Ecuyer and F. Blouin, "Linear congruential generators of order k¿ 1," pp. 432–439, 1988.

[18] L.-Y. Deng, "Efficient and portable multiple recursive generators of large order," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 15, no. 1, pp. 1–13, 2005.

[19] L.-Y. Deng and D. K. Lin, "Random number generation for the new century," *The American Statistician*, vol. 54, no. 2, pp. 145–150, 2000.

[20] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Trans. Math. Softw.*, vol. 32, no. 1, pp. 1–16, Mar. 2006, ISSN: 0098-3500. DOI: 10.1145/1132973.1132974. [Online]. Available: https://doi.org/10.1145/1132973.1132974.

[21] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[22] L.-Y. Deng and J.-J. H. Shiau, "Uniform random numbers," *In:Wiley StatsRef: Statistics Reference online. Hoboken, NJ: Wiley*, pp. 1–14, Dec. 2015.

[23] L.-Y. Deng and D. Bowman, "Developments in pseudo-random number generators: Pseudo-random number generators," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 9, e1404, Aug. 2017. DOI: 10.1002/wics.1404.

[24] L.-Y. Deng, H. Li, J.-J. Shiau, and G.-H. Tsai, "Design and implementation of efficient and portable multiple recursive generators with few zero coefficients," *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pp. 263–273, Jan. 2008. DOI: 10.1007/978-3-540-74496-2_15.

[25] L.-Y. Deng, J.-J. Shiau, and G.-H. Tsai, “Parallel random number generators based on large order multiple recursive generators,” *Monte Carlo and Quasi-Monte Carlo Methods 2008*, pp. 289–296, Jan. 2009. DOI: 10.1007/978-3-642-04107-5_17.

[26] G. Marsaglia, “Random numbers fall mainly in the planes,” *Proceedings of the National Academy of Sciences*, vol. 61, no. 1, pp. 25–28, 1968, ISSN: 0027-8424. DOI: 10.1073/pnas.61.1.25. eprint: https://www.pnas.org/content/61/1/25.full.pdf. [Online]. Available: https://www.pnas.org/content/61/1/25.

[27] P. L’Ecuyer, “Bad lattice structures for vectors of nonsuccessive values produced by some linear recurrences,” *INFORMS Journal on Computing*, vol. 9, no. 1, pp. 57–60, 1997.

[28] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. 3rd ed. Addison-Wesley Professional, 2014.

[29] P. L’ecuyer, “Tables of linear congruential generators of different sizes and good lattice structure,” *Mathematics of Computation*, vol. 68, no. 225, pp. 249–260, 1999.

[30] C. Kao and H.-C. Tang, “Upper bounds in spectral test for multiple recursive random number generators with missing terms,” *Computers & Mathematics with Applications*, vol. 33, no. 4, pp. 119–125, 1997.

[31] G. S. Fishman and L. R. Moore III, “An exhaustive analysis of multiplicative congruential random number generators with modulus 2̂31-1,” *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 1, pp. 24–45, 1986.

[32] H. Niederreiter, “A pseudorandom vector generator based on finite field arithmetic,” *Math. Japonica*, vol. 31, no. 5, pp. 759–774, 1986.

[33] P. L’Ecuyer, “Random numbers for simulation,” *Communications of the ACM*, vol. 33, no. 10, pp. 85–97, 1990.

[34] J. N. Franklin, "Equidistribution of matrix-power residues modulo one," *Mathematics of Computation*, vol. 18, no. 88, pp. 560–568, 1964.

[35] H. Grothe, "Matrix generators for pseudo-random vector generation," *Statistische Hefte*, vol. 28, no. 1, pp. 233–238, 1987.

[36] H. C. Williams and E. Seah, "Some primes of the form $(a^n - 1)/(a - 1)$," *Mathematics of Computation*, vol. 33, no. 148, pp. 1337–1342, 1979, ISSN: 00255718, 10886842. [Online]. Available: `http://www.jstor.org/stable/2006470`.

[37] J. Brillhart, D. Lehmer, J. Selfridge, B. Tuckerman, and J. Wagstaff S.S., *Factorizations of $bn\pm1$, b = 2, 3, 5, 6, 7, 10, 11, 12, up to high powers*, Third, ser. Contemporary Mathematics Series. Providence, RI, USA: American Mathematical Society, 2002. [Online]. Available: `http://www.ams.org/online_books/conm22`.

[38] H. C. Williams, *Édouard Lucas and primality testing*. John Wiley & Sons, 1998, vol. 23.

[39] L.-Y. Deng, "Issues on computer search for large order multiple recursive generators," in *Monte Carlo and Quasi-Monte Carlo Methods 2006*, S. Keller Alexander and Heinrich and H. Niederreiter, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 251–261.

[40] P. L'Ecuyer, "Good parameters and implementations for combined multiple recursive random number generators," *Oper. Res.*, vol. 47, pp. 159–164, 1999.

[41] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in p," *Annals of mathematics*, pp. 781–793, 2004.

[42] G. Marsaglia, "The marsaglia random number cdrom including the diehard: A battery of tests of randomness. see http://stat.fsu.edu/pub/diehard," Tech. Rep., 1996. [Online]. Available: `http://stat.fsu.edu/pub/diehard`.

[43] A. Rukhin, "Testing randomness: A suite of statistical procedures," *Theory of Probability and Its Applications*, vol. 45, pp. 111–132, Apr. 2000. DOI: `10.1137/S0040585X97978087`.

[44] P. L'Ecuyer and R. Simard, "On the lattice structure of a special class of multiple recursive random number generators," *INFORMS Journal on Computing*, vol. 26, no. 3, pp. 449–460, 2014. DOI: `10.1287/ijoc.2013.0576`. eprint: `https://doi.org/10.1287/ijoc.2013.0576`. [Online]. Available: `https://doi.org/10.1287/ijoc.2013.0576`.

[45] P. L'Ecuyer and R. Couture, "An implementation of the lattice and spectral tests for multiple recursive linear random number generators," *INFORMS Journal on Computing*, vol. 9, no. 2, pp. 206–217, 1997.

[46] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische annalen*, vol. 261, no. ARTICLE, pp. 515–534, 1982.

[47] H. Cohen, *A Course in Computational Algebraic Number Theory*. 1993, vol. 138.

[48] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. 2nd ed. Addison-Wesley, Reading, MA, 1981.