An Empirical Study on the Power of the Overlapping Serial Test

Xiaoke Xu and Wai Wan Tsang

Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
Email: {xkxu, tsang}@cs.hku.hk

July 3, 2007

Abstract

Random number generators (RNGs) are commonly used in simulations. The overlapping serial test is an important test that examines the randomness of RNGs. Its theory has been well-developed but its true ability for rejecting poor RNGs is not known. In this paper, we study the power of the test against the RNGs included in the widely spread *GNU Scientific Library*. By systematically varying the choices of parameters of the test, we find a fine-tuned version that rejects 29 RNGs out of the total of 57 in the library. We will like to warn users not to use these 29 RNGs.

Keyword: random number generator, test of randomness, overlapping serial test

1 Introduction

Nowadays, random number generators (RNGs) are routinely used in simulations and key generation in cryptographic applications. As poor generators will lead to biased results in simulations and successful guesses on cryptographic keys, the quality assurance of generators has drawn much attention ever since computers began to thrive. A basic requirement of a good generator is that it passes all commonly known statistical tests of randomness. The *overlapping serial test* is an important test of randomness developed by Good in 1951 [2]. It is recommended by Knuth [3] and Marsaglia [4] [5]. It is also called the *overlapping m-tuple test* by Marsaglia.

Consider an experiment whose outcome is $0, 1, ..., d-1$ with probability $P(0), P(1), ..., P(d-1)$, respectively. For the efficiency in representing

samples and in generating samples in our experiments, d is restricted to be powers of 2 and k =$\log_2 d$ is an integer. The experiment is repeated n times and the outcomes are $Y_1, Y_2, ..., Y_n$. The overlapping serial test treats the outcomes as a cyclic string and examines the distributions of two kinds of overlapped samples. The first kind of samples consist of t outcomes. They are $S_1 = (Y_1, Y_2, ..., Y_t)$, $S_2 = (Y_2, Y_3, ..., Y_{t+1})$, ..., and $S_n = (Y_n, Y_1, ..., Y_{t-1})$. The second kind of samples consist of $t-1$ outcomes. They are $S'_1 = (Y_1, Y_2, ..., Y_{t-1})$, $S'_2 = (Y_2, Y_3, ..., Y_t)$, ..., and $S'_n = (Y_n, Y_1, ..., Y_{t-2})$. The number of samples in both kinds are equal to n. The test statistic is the difference between two *Pearson* forms [9] [10].

$$V = \sum_{|\alpha|=t} \frac{(N(\alpha) - nP(\alpha))^2}{nP(\alpha)} - \sum_{|\alpha|=t-1} \frac{(N(\alpha) - nP(\alpha))^2}{nP(\alpha)}. \qquad (1)$$

In the first summation of the formula, $\alpha = a_1 a_2 \dots a_t$ with $0 \leq a_i < d$. $N(\alpha)$ is the number of times that $S_i = \alpha$, for $i = 1, \dots, n$. $P(\alpha) = P(a_1)P(a_2)\dots P(a_t)$ is the probability that a particular sample, say, S_1, is equal to α. The significances of the symbols appearing in the second summation are similar, except that each sample only consists of $t-1$ outcomes. Asymptotically, V follows the chi-square distribution of $d^t - d^{t-1}$ degrees of freedom. Note that when $t = 1$, the test degenerates to the standard chi-square test.

The most important feature of a test of randomness is its power, i.e., its ability to reject poor RNGs. However, because of the difficulty in theory, the powers of these tests are seldom studied analytically. In this paper, we take an empirical approach to explore the power of the overlapping serial test. We first choose a pool consisting of RNGs of different types and of various quality. We then define the stringency of a test as the number of RNGs it fails in the pool. By systematically varying the choices of k and t of the overlapping serial test, we fine-tune the test for maximum stringency. We also take the same approach to compare the stringency of the test with the gorilla test. The gorilla test is chosen because it is one of the most powerful tests of randomness [7]. Moreover, it examines the Y_i's in a way very similar to the overlapping serial test.

Our study reveals that the overlapping serial test is most stringent when $k = 1$. Moreover, t shall be set to the largest possible value, subject to a constraint imposed by the size of the main memory in the platform computer used for testing. Finally, we find that the power of the fine-tuned test is at par with the gorilla test. The overlapping serial test rejects 29 out of the 57 RNGs in the pool while the gorilla test rejects 28. The two sets of rejected RNGs are mostly overlapping. We would like to alert users not to use these RNGs in simulations.

2 The GSL Pool for Gauging Stringency

We would like to have a pool of RNGs such that a powerful test will fail more RNGs in the pool than a weaker one. The pool shall contain RNGs of different types and of various levels of randomness. One adequate candidate is the RNGs contained in the *GNU Scientific Library (GSL)*. GSL is a free and widely used library for scientific applications. It contains 57 RNGs suggested by experts in random number generation, including 17 *linear congruential generators* (LCG) and 18 *linear feedback shift-register generators* (LFSR). Figure 1 lists the names of these generators and the numbers of bits in the words they generate.

num	name	bits	num	name	bits	num	name	bits	num	name	bits
1	borosh13	32	16	ran1	31	31	random64-glibc2	31	46	ranlxs1	24
2	coveyou	32	17	ran2	31	32	random64-libc5	31	47	ranlxs2	24
3	cmrg	31	18	ran3	29	33	random8-bsd	31	48	ranmar	24
4	fishman18	31	19	rand	31	34	random8-glibc2	31	49	slatec	22
5	fishman20	31	20	rand48	32	35	random8-libc5	31	50	taus	32
6	fishman2x	31	21	random128-bsd	31	36	random-bsd	31	51	transputer	32
7	gfsr4	32	22	random128-glibc2	31	37	random-glibc2	31	52	tt800	32
8	knuthran	30	23	random128-libc5	31	38	random-libc5	31	53	uni	15
9	knuthran2	31	24	random256-bsd	31	39	randu	31	54	uni32	31
10	lecuyer21	31	25	random256-glibc2	31	40	ranf	32	55	vax	32
11	minstd	31	26	random256-libc5	31	41	ranlux	24	56	waterman14	32
12	mrg	31	27	random32-bsd	31	42	ranlux389	24	57	zuf	24
13	mt19937	32	28	random32-glibc2	31	43	ranlxd1	32			
14	r250	32	29	random32-libc5	31	44	ranlxd2	32			
15	ran0	31	30	random64-bsd	31	45	ranlxs0	24			

Figure 1: The names and word lengths of the RNGs in GSL.

The quality of the RNGs in GSL varies in a large range. A good RNG passes all known tests whereas a poor one passes only a few. This phenomenon can be used to distinguish powerful tests from weaker ones. Namely, a weak test can reject only few RNGs in GSL whereas a powerful test can reject many. For examples, the serial correlation test in [3] fails only 1 RNG in GSL but the birthday spacing test [3] [7] fails up to 31.

Let us define the *stringency* of a test as the number of GSL RNGs failed by the test. Suppose the stringency of Test A is higher than Test B and the RNGs failed by Test A is a superset of the RNGs failed by Test B, we conclude that Test A is more powerful than Test B. The superset condition does not necessarily hold between any two tests as they may check different deficiencies of RNGs. For example, the GSL RNGs rejected by the gorilla test does not have substantial overlap with that of the birthday spacing test. However, in the cases where two tests inspect similar features of samples, the superset condition often holds. Take, for example, the overlapping serial test and the gorilla test. These two tests extract same samples from RNG

outcomes but compute different statistics. The RNGs rejected by the two tests are mostly overlapped.

As test outcomes are probabilistic in nature, the superset condition may not hold even if Test A is truely more powerful than Test B. Thus, we will only use this condition as an auxiliary reference instead of a necessary requirement in determining whether a test is more powerful than another.

The GSL pool is a useful tool for tuning the parameters of a test because different versions of the test inspect samples in the same way. The RNGs failed by a low stringency version are likely a subset of the RNGs failed by a higher stringency version.

3 Two-Phase Testing

In this section, we describe the procedure of examining the outcomes of an RNG using the overlapping serial test. First, we describe how to examine a bit sequence using the test. Second, we explain how to extract bit sequences from the RNG outcomes and how to combine the results from testing the bit sequences into a single statistic, and eventually, how to determine pass or fail.

Consider applying the overlapping serial test on a bit sequence $b_1 b_2 b_3 ... b_m$. m is a multiple of k, and $n = m/k$. Each Y_i in Formula (1) is mapped into k bits. S_i consists of kt bits while S'_i consists of $k(t-1)$ bits. As an example, suppose we examine a bit sequence of length $m = 2^{24}$ with the test $[k = 4, t = 6]$. The sample size then is $n = 2^{22}$, while outcomes are $Y_1 = b_1 b_2 b_3 b_4$, $Y_2 = b_5 b_6 b_7 b_8$, ... and $S_1 = (Y_1, Y_2, Y_3, Y_4, Y_5, Y_6)$, $S_2 = (Y_2, Y_3, Y_4, Y_5, Y_6, Y_7)$, ..., etc. V is computed according to Formula (1) and its value is substituted in the chi-square distribution of $d^t - d^{t-1}$ degrees of freedom. If the bit sequence is truly random, the resulting value, $p = Chisq(V, d^t - d^{t-1})$, is a uniform random number in [0, 1). If the samples, S_i's and S'_i's, are too evenly distributed, p will tend to be close to 0. If the samples are too unevenly distributed, p will tend to be close to 1.

Now, we are ready to describe a two-phase procedure for examining an RNG. Suppose $w_1, w_2, ..., w_m$ is a sequence of words generated by the RNG and each word consists of s bits. With reference to Figure 2, b_{ij} is the j^{th} bit of w_i. $< bi >$, $1 \le i \le s$, is the bit sequence formed by concatenating the i^{th} bits of each word (bits in the i^{th} column).

In Phase 1, we apply the test to each $< bi >$ to obtain p_i. In Phase 2, the *Anderson-Darling Goodness-of-fit Test (AD Test)* [1] [6], is applied on the p's to check whether they are truly uniformly distributed. We reject the RNG if the resulting p-value is less than the significance level of 0.01.

The upper portion of Figure 3 shows the values of the 31 p's obtained in testing the *lecuyer21* RNG using the overlapping serial test of $[k = 12, t = 2]$ in Phase 1. The p-value returned from the AD test on these 31 p_i's in Phase

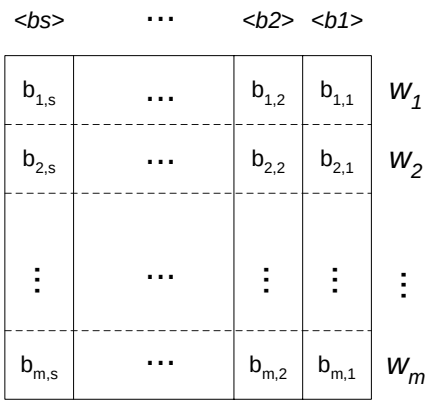

Figure 2: The bit sequences extracted from the outcomes of an RNG.

2 is 0.148. The generator passes this version of the test. The lower portion of the figure shows the p_i's when we test the same generator with the test of $[k = 1, t = 24]$. Note that five of the p_i's are zero or close to zero. The p-value returned by the AD test is 0.000. The generator is rejected.

[k = 12, t = 2]		p-value = 0.148					
p1 - p7	0.315	0.012	0.046	0.227	0.524	0.779	0.726
p8 - p14	0.200	0.559	0.425	0.372	0.554	0.068	0.232
p15 - p21	0.506	0.178	0.835	0.352	0.344	0.541	0.717
p22 - p28	0.124	0.034	0.761	0.451	0.425	0.383	0.645
p29 - p31	0.741	0.146	0.770				
[k = 1, t = 24]		p-value = 0.000					
p1 - p7	0.418	0.251	0.026	0.975	0.048	0.071	0.428
p8 - p14	0.409	0.927	0.655	0.041	0.063	0.269	0.417
p15 - p21	***0.005***	0.041	0.177	***0.000***	***0.002***	0.023	0.024
p22 - p28	0.151	0.095	0.381	0.117	0.181	***0.000***	0.033
p29 - p31	***0.007***	0.025	0.110				

Figure 3: Tables of the p_i's obtained from testing the *lecuyer21* RNG using the overlapping serial test of $[k = 1, t = 24]$ and $[k = 12, t = 2]$.

4 Tuning the Parameters for Maximum Stringency

Consider examining m words generated by an RNG. We want to determine the values of k and t such that the overlapping serial test reaches its maximum stringency. As the test needs to use a large amount of main memory,

$2^{kt} + 2^{k(t-1)}$ integer variables, for keeping the numbers of the occurrences of all possible samples of lengths kt and $k(t-1)$, the memory size of the platform computer imposes a key constraint on the choices of k and t. As an example, the test of $[k = 4, t = 6]$ uses $2^{24} + 2^{20}$ integer variables. Assuming an integer variable is represented using 4 bytes, the memory needed is 68Mbytes.

4.1 Tuning k

Given m, we want to find a choice for k such that the test reaches its maximum stringency. Since the main memory is a key resource, we only consider those versions of the test that use approximately 2^{kt} integer variables. Considering the cases of $m = 2^{24}$, we gauged the stringencies of the versions of $[k = 1, t = 24], [k = 2, t = 12], [k = 3, t = 8], [k = 4, t = 6], [k = 8, t = 3], [k = 12, t = 2]$, and $[k = 24, t = 1]$. Each of these versions examines all m words being tested. The amount of main memory used lies in $[2^{24}, 2^{24}+2^{23}]$. The results are shown in Figure 4. There is a clear trend that the stringency decreases as k increases. The version of $k = 1$ rejects most RNGs. It rejects all the generators failed by the version of $k = 12$ and three more: *lecuyer21*, a multiplicative RNG (pp.106-108 of [3]); *minst*, Park and Miller's "minimal standard" RNG; and *ran0*, another *minst* RNG with modified seeding procedure.

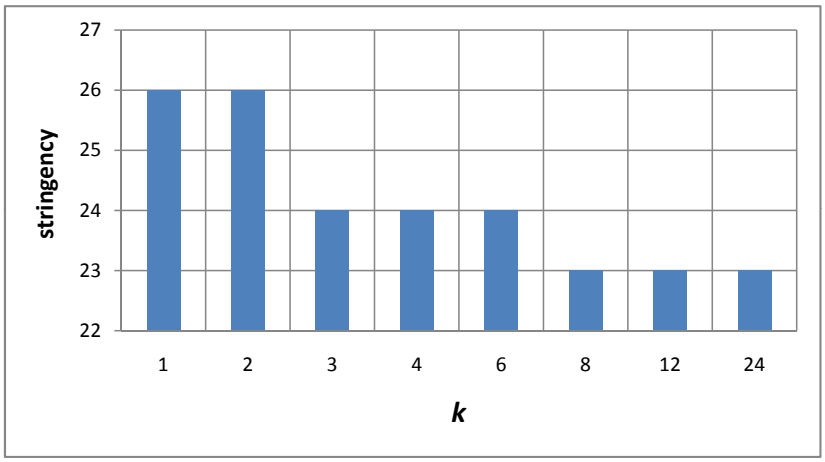

Figure 4: The stringency of the overlapping serial test against k for $m = 2^{24}$, $t = 24/k$.

The experiment of tuning k was repeated for other values of m. In all cases, the version of $k = 1$ always prevails. We conclude that k shall always be set to 1. In such cases, the amount of overlapping bits between adjacent samples, say, S_i and S_{i+1}, reaches maximum.

4.2 Tuning t

Given m and $k = 1$, we want to find a choice for t such that the test reaches its maximum stringency, subject to the constraint imposed by the memory size of the platform computer. Consider the case where $m = 2^{26}$ and the platform computer has up to 512M bytes free memory. The size of memory imposes a constraint of $t \leq 26$ on the test. Thus, we gauged the stringencies of the versions: $[k = 1, t = 2], [k = 1, t = 6], ...$, and $[k = 1, t = 26]$. The results are shown in Figure 5. The stringency of the test rises progressively as t increases. Moreover, an RNG rejected by a version of low stringency is also rejected by the more stringent versions. Figure 6 shows eight generators in the GSL pool which pass the test when $t = 6$ but are gradually rejected by the test when t increases to 26. The same trend was observed in the experiments with other m values. We conclude that t shall be set to the largest value, subject to the constraint imposed by the memory of the platform computer.

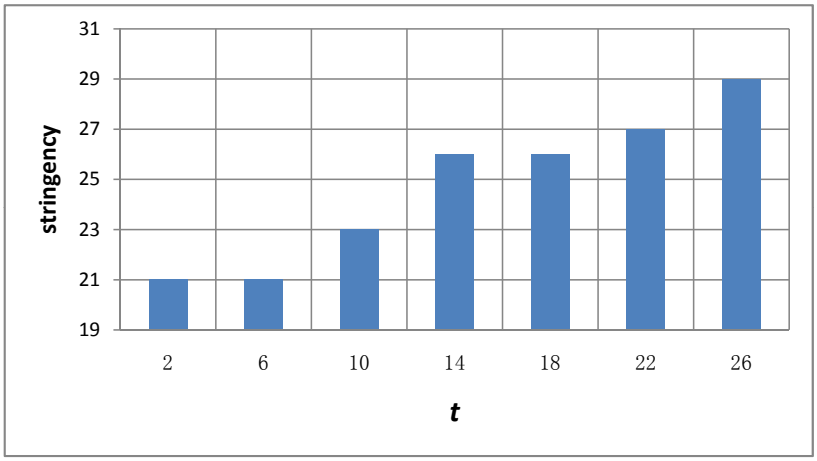

Figure 5: The stringency of the overlapping serial test against t for $m = 2^{26}$ and $k = 1$.

5 Comparison with the Gorilla Test

The gorilla test is the most powerful version of the monkey test and is among the most difficult-to-pass tests for RNGs [7] [8]. It extracts S_i's from $< bi >$ in the same way as the overlapping serial test, but counts the number of all possible S_i's which are missing. Comparing the overlapping serial test with the gorilla test gives us an idea of how powerful the latter really is.

The gorilla test given in [7] examines $m = 2^{26}$ words of an RNG. To be fair, we compare it with the overlapping serial test of $[k = 1, t = 26]$,

	t=2	t=6	t=10	t=14	t=18	t=22	t=26
fishman18	P	P	F	F	F	F	F
fishman20	P	P	F	F	F	F	F
lecuyer21	P	P	P	F	F	F	F
minstd	P	P	P	F	F	F	F
ran0	P	P	P	F	F	F	F
random256-bsd	P	P	P	P	P	F	F
uni	P	P	P	P	P	P	F
uni32	P	P	P	P	P	P	F

Figure 6: Eight GSL generators which pass (P) the test when $t = 6$ but gradually fail (F) when $t = 26$.

which examines same number of words. We apply both tests to check the RNGs in the GSL pool. The results are shown in Figure 7. The gorilla test rejects 28 out of the 57 RNGs in the pool while the overlapping serial test rejects 29. 27 RNGs are rejected by both tests. Some of these RNGs are widely used in standard *C* library, e.g., *rand* and *rand48*. The only RNG that fails the gorilla test but passes the overlapping serial test is *knuthran2*, a second-order multiple recursive generator suggested by Knuth. The two RNGs that pass the gorilla test but fail the overlapping serial test are *ran3* and *random256-bsd*. As the two sets of RNGs rejected by the two tests are mostly overlapped, we conclude that the power of the overlapping serial test is at par with the gorilla test.

The overlapping serial test uses an integer variable (4 bytes) for keeping the count of the occurrence of a possible sample, whereas the gorilla test uses only 1 bit for keeping track whether the sample has occurred. Thus, the gorilla test uses far less memory. On the other hand, the gorilla test suffers from a major drawback. The cumulative distribution function (CDF) of its statistic cannot be derived analytically. The p-value of the test is computed from an empirical cumulative distribution obtained via extensive simulation.

6 Future Work

In deriving the CDF of the V defined in Formula (1), the $N(\alpha)$'s are assumed following a joint normal distribution. This assumption is severely violated in the extreme sparse versions of the test, i.e., when m is small and t is very large. So far, we have not yet encountered any noticeable discrepancy between the CDF derived and the true CDF. However, we shall be aware of this possible hazard in practice, and find out how sparse we can go before the theory breaks down.

Num	Name	OS	Gorilla	Num	Name	OS	Gorilla	Num	Name	OS	Gorilla
1	borosh13	F	F	21	random128-bsd	P	P	41	ranlux	P	P
2	coveyou	F	F	22	random128-glibc2	P	P	42	ranlux389	P	P
3	cmrg	P	P	23	random128-libc5	P	P	43	ranlxd1	P	P
4	fishman18	F	F	24	random256-bsd	***F***	***P***	44	ranlxd2	P	P
5	fishman20	F	F	25	random256-glibc2	P	P	45	ranlxs0	P	P
6	fishman2x	P	P	26	random256-libc5	P	P	46	ranlxs1	P	P
7	gfsr4	P	P	27	random32-bsd	F	F	47	ranlxs2	P	P
8	knuthran	P	P	28	random32-glibc2	F	F	48	ranmar	P	P
9	knuthran2	***P***	***F***	29	random32-libc5	F	F	49	slatec	F	F
10	lecuyer21	F	F	30	random64-bsd	F	F	50	taus	P	P
11	minstd	F	F	31	random64-glibc2	F	F	51	transputer	F	F
12	mrg	P	P	32	random64-libc5	F	F	52	tt800	P	P
13	mt19937	P	P	33	random8-bsd	F	F	53	uni	F	F
14	r250	F	F	34	random8-glibc2	F	F	54	uni32	F	F
15	ran0	F	F	35	random8-libc5	F	F	55	vax	F	F
16	ran1	P	P	36	random-bsd	P	P	56	waterman14	F	F
17	ran2	P	P	37	random-glibc2	P	P	57	zuf	P	P
18	ran3	***F***	***P***	38	random-libc5	P	P				
19	rand	F	F	39	randu	F	F				
20	rand48	F	F	40	ranf	F	F				

Figure 7: The results of testing the GSL generators using the overlapping serial test and the gorilla test.

Acknowledgment

This research in supported by the Hong Kong Research Grants Council HKU7143/04E.

References

[1] T.W. Anderson and D.A. Darling. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *The Annals of Mathematical Statistics*, 23(2):193–212, 1952.

[2] I.J. Good. The serial test for sampling numbers and other tests for randomness. In *Proceedings of Cambridge Philosophical Society*, pages 49:276–284, 1953.

[3] D.E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Welsley, third edition, 1998.

[4] G. Marsaglia. A current view of random number generators. In *Computer Science and Statistics: Proceedings of the 16th Symposium on the Interface*, pages 151–158, Atlanta, Georgia, 1984.

[5] G. Marsaglia. Monkeying with the Goodness-of-Fit Test. *Journal of Statistical Software*, 2005.

[6] G. Marsaglia and J. Marsaglia. Evaluating the Anderson-Darling distribution. *Journal of Statistical Software*, 9(2):1–5, 2004.

[7] G. Marsaglia and W.W. Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–8, 2002.

[8] G. Marsaglia and A. Zaman. Monkey tests for random number generators. *Computers and Mathematics with Applications*, 26(9):1–10, 1993.

[9] K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50(5):157–175, 1900.

[10] W.W. Tsang and Chi yin Pang. The Mathematics of the Overlapping Chi-square Test. Technical Report TR-2006-12, Department of Computer Science, The University of Hong Kong, September, 2006.