METHODS FOR GENERATION OF RANDOM NUMBERS IN PARALLEL STOCHASTIC ALGORITHMS FOR GLOBAL OPTIMIZATION

Algirdas Lančinskas, Julius Žilinskas
Institute of Mathematics and Informatics

1. Introduction

Generation of random numbers plays an important role in stochastic optimization algorithms where sequences of random numbers with particular distribution must be generated. There are many techniques to gain random numbers but they vary in quality and properties of produced numbers. In stochastic algorithms for global optimization important properties of random numbers can be the following: fitness for required probability distribution, repetition of numbers, correlation between sequences of generated numbers, and speed of generation.

There are two principal methods to generate random numbers. One measures a physical phenomenon that is expected to be random. Usually it is called hardware random number generation. The other method uses deterministic computational algorithms that produce long sequences of apparently random numbers. The latter type is often called pseudo random number generator (PRNG). Most of PRNGs must have initial number that is used to start generation of sequence and called a *"seed"*. The results are the same each time when the same seed is used. In parallel computing seeds should be chosen so that the sequences of random numbers in different processes are independent and uncorrelated.

2. Pseudo random number generators

Linear congruential generator (LCG) is a classic random number generator. It generates a sequence of nonnegative integers. It was introduced by Lehmer D. H. (1949) and has become the basis for many random number generators in use today (Moler, 2004). A simple LCG is defined by a recursive relation

$$x_{k+1} = (ax_k + c) \bmod m . \qquad (1)$$

Like most generators it requires an initial seed value x_0. The constant a is called a multiplier, c is an increment and m is a modulus. LCG has at most m distinct values in its sequence thus LCG can generate no more different numbers than its modulus. LCG generates the maximum quantity of different random numbers if and only if (Wehrwein, 2007) c and m are relatively prime, a–1 is divisible by all prime factors of m, and a–1 is multiple of 4 if m is multiple of 4.

If uniformly distributed pseudo random integers from 0 to m–1 are divided by m–1, the resulting floating-point numbers are uniformly distributed in the interval [0,1].

LCG algorithm is used in programming environments such as Visual C/C++, Borland C/C++, Borland Delphi, etc. Crucial differences of usage of LCG are choice of parameters and utilization of bits of produced result, i.e. Borland C/C++ built-in function *rand()* uses parameters $a = 214013$, $c = 2531011$, $m = 2^{32}$ and bits from 16 to 30 of produced number.

Good results are achieved by experimentally testing LCG with parameters $a = 7777$, $c = 101$, $m = 32771$. Description of experiments and their results is given below.

Another algorithm for generation of random numbers is *Mersenne twister*. It is developed by Matsumoto M. and Nishimura T. (1998) and based on a matrix linear recurrence over a finite binary field. Its name derives from the fact that period length is chosen to be a Mersenne prime. Implementation of the algorithm was downloaded from Agner.org.

Marsaglia G. (2005) invented *multiply with carry* method for generating sequences of random integers. Suppose the digits x, y, z and 'carry' c are represented in the form x, y, $^c z$. The iteration rule is as follows (Marsaglia, 2005):

$t = 7(x + y) + c$,
$x \leftarrow y$, $y \leftarrow z$, $c \leftarrow \lfloor t/10 \rfloor$, $z \leftarrow t \bmod 10$,
output z.

For example, given initial $x = 9$, $y = 2$, $z = 1$ and $c = 13$ we have a sequence

$$9, 2, {}^{13}1, {}^{9}0, {}^{3}0, {}^{1}0, {}^{0}1, {}^{0}0, {}^{0}7, {}^{0}7, {}^{4}9, \ldots$$

and an output sequence

000107792296033647649985249751614400438121835342573981850840 07...

A multiply-with-carry generator with more than one factor is called the Mother-Of-All random number generator that is also invented by Marsaglia G. (Agner. org). Implementation of Mother Of All algorithm was downloaded from Agner.org.

3. Investigation of randomly generated numbers

One criterion of random number generator is fitness for required distribution of randomly generated numbers. We use *Kolmogorov-Smirnov (K-S) statistics* to test this property. It quantifies the maximum distance between the empirical distribution function of the sample and the cumulative function of the expected distribution.

Empirical function F_n for sequence of numbers x_i, $i = 1, 2, ..., n$ can be defined as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{x_i \le x}, \quad (2)$$

where

$$I_{x_i \le x} = \begin{cases} 1, \text{if } x_i \le x, \\ 0, \text{otherwise.} \end{cases}$$

Expected (theoretical) uniform distribution cumulative function $F(x)$ and empirical cumulative function $F_n(x)$ when numbers are uniformly distributed in [0; 1] are presented in Figure 1.

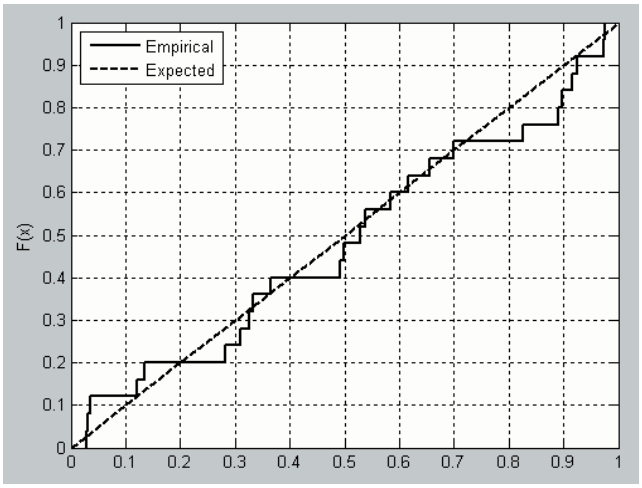

Figure 1. Expected uniform distribution and empirical cumulative functions

Kolmogorov-Smirnov statistics for given cumulative distribution function $F(x)$ are

$$D_n = \sup_x |F_n(x) - F(x)|.$$

Kolmogorov-Smirnov test may be used to reject a hypothesis that the considered sequence follows the required distribution (uniform distribution in our case). Let us define the hypothesis H_0: the sequence follows uniform distribution.

The hypothesis H_0 is rejected if the test statistic D_n is greater than the critical value obtained from a table. The hypothesis can be tested using MATLAB function [h,p,ksstat,cv]=kstest(x,[x;unifcdf(x)]',0.05), where x is a testing sequence, [x;unifcdf(x)] is the required distribution cumulative function (uniform distribution in our case), 0.05 is the significance level, h shows the result of hypothesis testing (0 means that H_0 is not rejected, 1 means that H_0 is rejected), p is p-value, ksstat is Kolmogorov-Smirnov statistic and cv is critical value.

Important criterion is quantity of different numbers in generated sequence which can be expressed in percents. For example, if in a sequence of 100 random numbers one number is repeated once, we can say that 99% of generated numbers are different.

Another criterion is correlation between sequences of randomly generated numbers. Suppose we have k independently generated sequences X_1, X_2, ..., X_k. We evaluate correlations $\rho(X_i, X_j)$, and present either the maximum of absolute value or both the minimum and the maximum.

The last criterion used in this paper is the speed of generation of random numbers. It can be measured by generating huge sequence of random numbers observing time elapsed during the generation. Note that the previous criteria do not depend on computer parameters while speed of generation can vary depending on CPU speed, virtual memory size, operating system, running processes, other conditions. Therefore speed of generation must be measured on identical systems under identical conditions.

4. Techniques for generating seed for random number generator

A seed is a starting value that is necessary for random number generators. If you run a program again with the same seed value then you will get exactly the same sequence of random numbers. If you use a different seed value then you will get a different sequence of random numbers. Computer time is usually used as the seed value to get unpredictable randomness. However this method is not acceptable in parallel computing since different nodes must use different seed values to make sure that each node generates a unique sequence of random numbers. This is complicated when two or more nodes use the same clock, for example in multi-tread systems. Therefore other techniques for generation of seed values are necessary. We have investigated several techniques to generate seed values for parallel computing.

S1. Time in seconds given by computer clock is multiplied by processor's ID:

$$seed_{S1} = time \cdot id\,.$$

S2. The number of clock ticks elapsed since the program was launched is multiplied by processor's ID:

$$seed_{S2} = tics \cdot id$$

S3. The required number of seeds is generated using random number generator on one of the processors and distributed to others.

S4. The current time, the number of tics and processor ID are composed using the formula:

$$seed_{S4} = (time \bmod 32768) \text{ rotr } (id \bmod 31) +$$
$$+ (tics \bmod 32768) \text{ rotr } (id \bmod 31),$$

where rotr means rotation of bits, i.e. *a* rotr *r* means the bits of *a* rotated by *r* places to the right, (00001111_2 rotr 3 = 11100001_2).

S5. The current time, the number of tics and processor ID are composed using the formula:

$$seed_{S5} = 2 \cdot \left(time \cdot tics \cdot 2^{id \bmod 15} \bmod 2^{30}\right) - 1\,.$$

All techniques have been investigated considering correlation between generated sequences.

5. Experimental investigation of random number generators

Seven methods for generation of random numbers were experimentally investigated considering Kolmogorov-Smirnov statistics (2), repetition of numbers in one sequence (3), correlation between sequences (4) and time of generation. 40 sequences of 10^8 random numbers with uniform distribution in range [0, 1] were generated to measure time and 100 sequences of 10^4 numbers were generated for other experiments. The same set of seeds was used for each method. Parameters $a = 7777$, $c = 101$ and $m = 32771$ (1) were used for LCG. Averaged results are shown in Table 1. In column "K-S statistic" averaged Kolmogorov-Smirnov statistic values are presented. Critical value is 0.0136 in our case. It means that a sequence with K-S statistic larger than the critical value did not pass fitness for uniform distribution test (see Section 3). The numbers of sequences that fail K-S test are presented in column "Rejected". Correlations with the largest absolute value (except 5% largest) are presented in column "Correlation".

Table 1. ***Average values of criteria of investigation of random number generators***

Generator	Different numbers (%)	K-S statistic	Rejected	Correlation	Time of generation
Matlab	99.996	0.00853	3	0.0047	2.105
Visual C++	86.156	0.00879	4	0.0050	4.374
LCG	100.00	0.00809	0	0.0023	3.124
Turbo Pascal	92.725	0.00856	1	0.0030	43.750
Mersenne Twister	99.995	0.00823	0	0.0098	10.250
Mother Of All	99.994	0.00836	1	0.0032	9.274
FORTRAN	99.983	0.00838	1	0.0158	2.002

As we can see in Table 1 LCG produces the largest quantity of different numbers in generated sequences (100% of produced numbers are different). That means that there are no sequences with repetitive numbers. The smallest Kolmogorov-Smirnov statistics and correlation are given with LCG, too. All of the generators except LCG and Mersenne twister produced at least one sequence that did not pass K-S test. Turbo Pascal generator provides small correlation, but larger repetition of numbers, Kolmogorov-Smirnov statistic and critical time of generation. FORTRAN generator seems

to be the fastest but produces the most correlated sequences.

95% confidence intervals for mean of K-S statistics are calculated and presented in Figure 2. We can see that K-S statistics are less than 0.0095 with probability of 0.95 for all generators. But there are few sequences with K-S statistics larger than critical value of 0.0136 and rejected to be uniformly distributed (see Table 1).

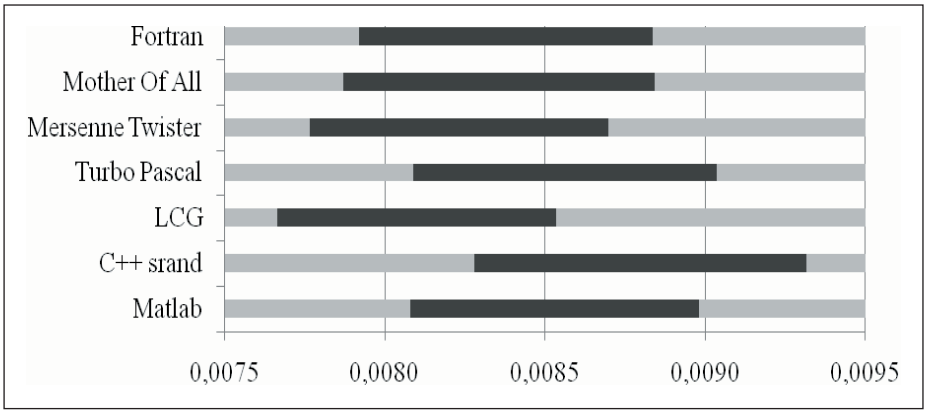

Figure 2. Confidence intervals for mean of K-S statistics of generated sequences

Intervals between minimum and maximum values (excepting 5% largest in absolute value) of correlations of sequences are presented in Figure 3.

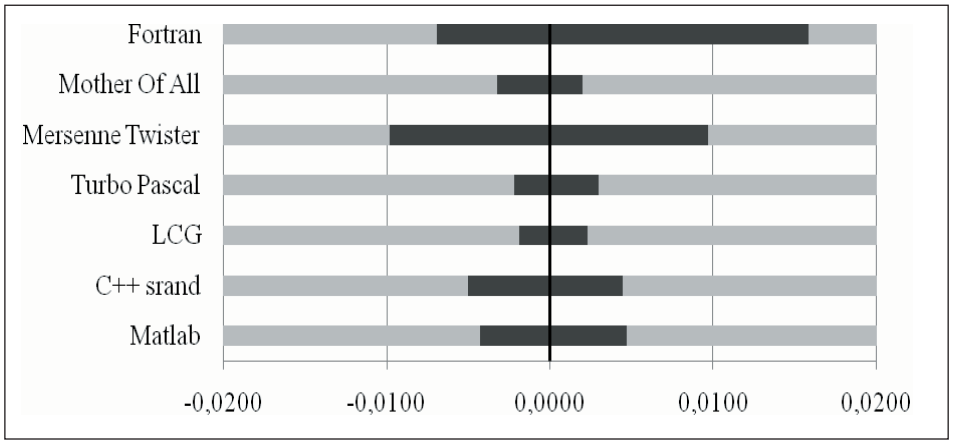

Figure 3. Minimum and maximum values of correlations of generated sequences

As we can see in Figure 3, correlations of sequences generated with LCG are the smallest. Correlations of sequences generated with FORTRAN and Mersenne Twister generators are most scattered.

Five techniques for generating seed values for random number generation in parallel computing (see previous section) have also been tested considering correlation between sequences of generated numbers. 100 sequences of 10^3 random numbers with uniform distribution in the range [0, 1] have been generated using 100 parallel processes. Averaged correlations of sequences of 10 experiments are presented in Table 2.

Table 2. ***Average correlation of sequences when seeds are generated using techniques S1-S5***

Technique	S1	S2	S3	S4	S5
Averaged correlation	-0.0002512	0.0001007	0.0000388	-0.0000034	-0.0000022

95% confidence intervals for mean of correlations are calculated and presented in Figure 4.

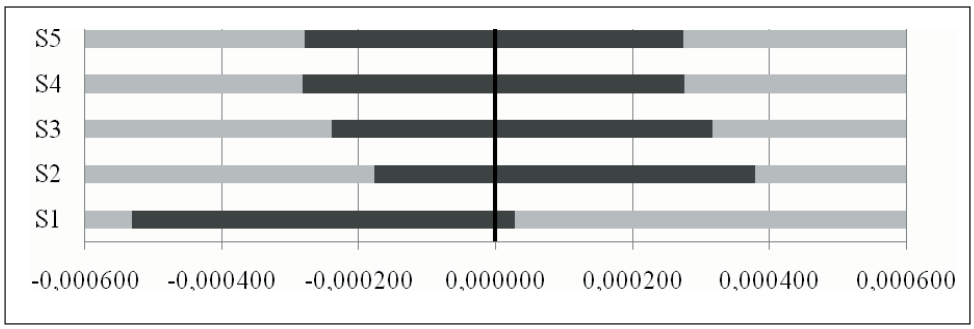

Figure 4. Confidence intervals for results of tests of techniques for generating seeds

As we can see from the results techniques involving larger number of parameters such as processor id, time in seconds, processor tics leads to generation of less correlated sequences.

6. Conclusions

The paper presents an overview of widely used methods for generation of random numbers and their experimental investigation. Seven methods have been investigated considering four criteria important in stochastic algorithms for parallel optimization. The best results have been achieved using Linear Congruential Generator with parameters $a = 7777$, $c = 101$ and $m = 32771$. Turbo Pascal internal generator produces sequences with the smallest correlation but it is the slowest. The fastest generator seems to be FORTRAN internal generator but it generates most correlated sequences.

Five techniques of seed construction for parallel generation of sequences of random numbers have also been experimentally tested. Results show that technique involving larger number of parameters (processor id, time in seconds, processor tics) leads to generation of less correlated sequences.

References

1. Moler C., 2004, *Numerical Computing with MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia.
2. Wehrwein J., 2007, *Random Number Generation*. Senior Thesis, Middlebury College.
3. Matsumoto M., Nishimura T., 1998, Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*. Vol. 8. No. 1. P. 3–30.
4. Marsaglia G., 2005, *On the Randomness of Pi and Other Decimal Expansions*. Florida State University. Available online at <http://interstat.statjournals.net/YEAR/2005/articles/0510005.pdf>.
5. Agner.org. Available online at <http://www.agner.org/random/>. [Accessed on 04-03-2010].
6. Lehmer D. H., 1949, Mathematical methods in large-scale computing units. *Proc. 2nd Sympos. on Large-Scale Digital Calculating Machinery*. P. 141–146. Harvard University Press.

METHODS FOR GENERATION OF RANDOM NUMBERS IN PARALLEL STOCHASTIC ALGORITHMS FOR GLOBAL OPTIMIZATION

Algirdas Lančinskas, Julius Žilinskas

Summary

Performance of stochastic algorithms for global optimization crucially depends on generation of random numbers. Random number generation methods may vary on features as independence of the generated random numbers, fit to the required distribution, and speed of generation. This paper reviews the main idea and several algorithms for generation of pseudo random numbers. Evaluation criteria of pseudo random numbers generators are also reviewed. Seven widely used random numbers generators (Linear Congruential Generator, Mersenne Twister, Mother At All, C++, Pascal, Matlab and Fortran) are experimentally compared evaluating the distribution of random numbers, correlation of sequences and speed of generation. In parallel computations correlation of sequences may depend on the seed of pseudo random numbers generators. Therefore several ways for construction of the seeds are compared considering correlation of generated sequences of random numbers when computations are performed in parallel computers.

Keywords: random numbers, random number generators, parallel stochastic algorithms, global optimization.

ATSITIKTINIŲ SKAIČIŲ GENERAVIMO PARALELINIUOSE STOCHASTINIUOSE ALGORITMUOSE BENDRAJAM OPTIMIZAVIMUI METODAI

Algirdas Lančinskas, Julius Žilinskas

Santrauka

Bendrojo optimizavimo stochastinių algoritmų efektyvumas itin priklauso nuo atsitiktinių skaičių generavimo. Atsitiktinių skaičių generavimo metodai gali skirtis tokiais bruožais, kaip sugeneruotų atsitiktinių skaičių nepriklausomumas, tinkamumas reikiamam skirstiniui ir generavimo greitis. Šiame straipsnyje apžvelgiama pagrindinė idėja ir keletas pseudoatsitiktinių skaičių generavimo algoritmų. Taip pat apžvelgiami pseudoatsitiktinių skaičių generatorių vertinimo kriterijai. Eksperimentiškai lyginami septyni plačiai naudojami atsitiktinių skaičių generatoriai (Linear Congruential Generator, Mersenne Twister, Mother At All, C++, Pascal, Matlab ir Fortran) vertinant atsitiktinių skaičių distribuciją, sekų koreliaciją ir generavimo greitį. Lygiagrečiuosiuose skaičiavimuose sekų koreliacija gali priklausyti nuo pseudoatsitiktinių skaičių generatorių pradinių skaičių. Todėl lyginami keli būdai sudaryti pradinius skaičius atsižvelgiant į sugeneruotų atsitiktinių skaičių sekų koreliaciją skaičiuojant paraleliais kompiuteriais.

Prasminiai žodžiai: atsitiktiniai skaičiai, atsitiktinių skaičių generatoriai, paraleliniai stochastiniai algoritmai, bendrasis optimizavimas.

Įteikta 2010 06 15