Research Journal of Applied Sciences, Engineering and Technology 5(6): 2174-2179, 2013
DOI:10.19026/rjaset.5.4769
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: August 02, 2012 Accepted: September 08, 2012 Published: February 21, 2013

Research Article

Two Improved Pseudo-Random Number Generation Algorithms Based on the Logistic Map

[1, 2]Chunyan Han, [1]Yunxiao Wang, [1]Yixian Liu and [1]Dancheng Li
[1]Department of Software College,
[2]Department of Information Science and Engineering, Northeastern University, Shenyang, 110819, China

Abstract: In this study, we improve the pseudo-random number generation algorithm of Java based on the logistic map. We replace the seed of the random generation algorithm with a sequence of numbers which is generated by the original logistic map and the improved logistic map. And then we describe the two improved algorithms in detail including the essence of two algorithms and the processes. Also we display a series of experiments to prove the improved algorithms have a good randomness and they are efficient. Eventually we conclude our study and list our future study. The two improved pseudo-random number generation algorithms take advantage of the chaotic characteristics of the logistic map which makes it become much more efficient.

Keywords: Pseudo-random number generation, randomness tests, the logistic map

INTRODUCTION

Nowadays, random numbers are widely used in many fields, like in the cryptography field. Random numbers are generated by the random numbers generator. A random numbers generator is a computational or physical device designed to generate a sequence of numbers or symbols that lack any pattern, i.e., appear random (Wagenaar, 1972; Kanter *et al*., 2010). The generation process of random bits is becoming increasingly important. Since the use of a truly random generator is very difficult as they are generally slow and may consist of special hardware, the importance of designing efficient pseudo-random generators arise (Zarei *et al*., 2010). In the following parts, random numbers mean pseudo-random numbers. There are 2 kinds of generation methods, physical methods and computational methods. Physical random number generators always use hardware or some special devices to generate random numbers like dice, coin flipping, roulette wheels. While pseudo-random number generators with computational methods are algorithms that can automatically create long runs of numbers with good random properties but eventually the sequence repeats. The string of values generated by such algorithms is generally determined by a fixed number called a seed (Heam and Nicaud, 2011). In this study we only concern computational pseudo-random number generation algorithms.

In many computer languages, there are pseudo-random number generation algorithms, like the random () in Java and rand () in C++; Even though the randomness of these methods is good, there still a problem. That is efficiency. In other words, the process of generating a pseudo-random number is relatively slow. In this study, we study the random method of Java language. And we figures out how it generates random numbers. Then we improved this pseudo-random number generation algorithm of Java language by using the logistic map. Here the logistic map includes the original logistic map and the improved logistic map. In this study, we firstly improve the pseudo-random number generation algorithm and describe the 2 algorithms in detail. Then we list some experiments to show the randomness and the efficiency of these 2 algorithms. At last, we conclude our study.

TWO IMPROVED PSEUDO-RANDOM NUMBER GENERATION ALGORITHMS

Random method of Java-RMOJ: There are many random number generation methods of many languages including Java language. We study the process of the random method of the Java. In Java language, random method is for generating random numbers. There are many random methods in Java with different parameters and different ranges. But the cores of these random methods are the same. After studying the source code of the random method in Java, we eventually know how it generates random numbers. We call this random method of Java RMOJ in short. Here are the 3 key steps of RMOJ:

Corresponding Author: Chunyan Han, Department of Software College, Northeastern University, Shenyang, 110819, China

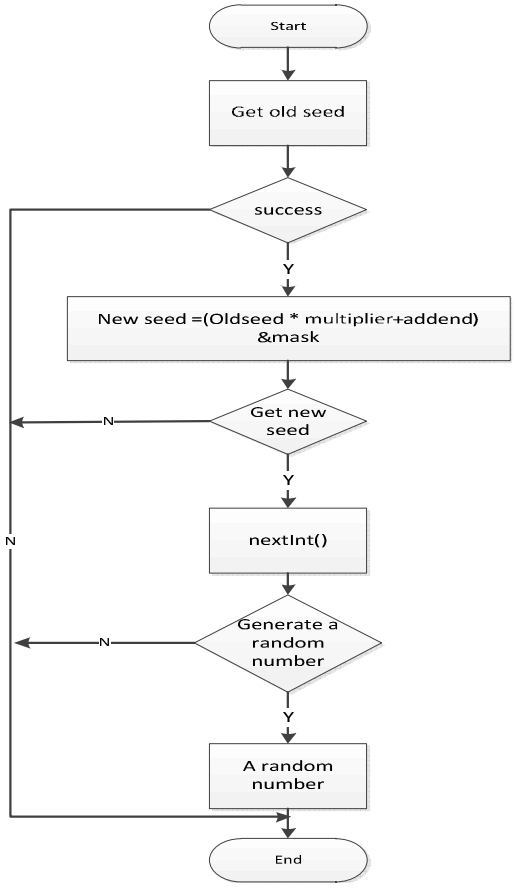

Fig. 1: Process of RMOJ

Step 1: Get a long type seed according to the current time of the system as old seed.

Step 2: Process this old seed with arithmetic operations and a bitwise operation into a new seed.

Step 3: Next Int method uses the new seed to generate a random number according to some specific operations including bitwise operations.

Take the method of generating an integer number as an example. The process of RMOJ is shown in Fig. 1.

In the process, multiplier, addend and mask are three variables. Multiplier is 0x5DEECE66DL and addend is 0xBL. Mask is (1L<<48) -1.

In the process, through step 1 and 2, RMOJ gets a new seed and by using the seed, it can generate a random number. In this 2 steps, an old seed would do a multiply operation, a plus operation and a bitwise and operation. These 2 steps cost much time because of the getting old seed according to the current time and arithmetic operations. There is no doubt that this is the bottleneck of the efficiency problem of RMOJ. If we can find some other faster method to generate a new seed, it would cost less time.

Improved random method of Java by using the logistic map-LRMOJ and ILRMOJ: We call the improved random method of Java by using the original logistic map the LRMOJ and the improved random method of Java by using the improved logistic map the ILRMOJ.

We improve this RMOJ and try to make it more efficient. The core of our idea is to make the sequence of numbers which is generated by the logistic map to be the seeds of generating random numbers.

Chaos is a random movement in definite system. Chaotic system has following characteristics: definite, boundary, sensitivity to initial condition, topology transmission and so on. Chaos structure is complex and difficult to analyze or forecast (Xiao-Jun *et al*., 2006). The logistic map is a polynomial mapping of degree 2, often used in the study of chaos systems. Through the process of RMOJ, we conclude that generating a seed is the most important phase of the method. And in the 1st 2 steps of RMOJ is to get a new seed by changing a long type old seed into a new long type seed. Especially in step 2, the old seed must do bitwise and operation and some arithmetic operations which actually would cost much time. If we want to make it much more efficient, we have to find a way to solve this time consuming problem of 1st 2 steps. Meanwhile we must make the method as random as RMOJ. Based on the chaotic characteristics of the logistic map, we decide to use the logistic map sequence as seeds.

The logistic map can mathematically be described as Eq. (1):

$$X_{n+1} = r*X_n* (1 - X_n) \quad (1)$$

This equation is a recursive equation. Where X_n or X_{n+1} are decimal ranges between 0 and 1, r is a coefficient which is between 0 and 4. When the r is beyond 3.57, most values appear chaotic behaviors (Persohn and Povinelli, 2012). Here we let $X_0 = 0.75$, r = 3.935. We replace the new seeds with the sequence of numbers that this logistic map generates. That is to say, we substitute the step 1 and 2 of RMOJ with one single step. In this new step, we use the sequence of the logistic map as new seeds. Obviously the numbers that this logistic map generates are decimals between 0 and 1; we process these decimals into long type integer by amplifying the decimals. And the step 3 and the following processes are the same with RMOJ. And we can get the steps of LRMOJ:

Step 1: Get a decimal from the logistic map sequence numbers and amplify the decimal into a long type number as a new seed.

Step 2: Next Int method uses the new seed to generate a random number according to some specific operations including bitwise operations.

In step 2, Next Int method is the method to adapt the new seed to a proper range like integral number which ranges from 0 to 10. In this method, the new seed would be processed by shifts operations. This is a same method RMOJ, LRMOJ and ILRMOJ would use.

Comparing with the 2 steps of RMOJ, the new step of LRMOJ only access one random number in the sequence of the logistic map rather than do a series of operations. The sequence is generated when the method is used first time and the same time the method initiates. And what is important to efficiency, the sequence only is generated once and it would be stored in the memory. When the method needs a new seed, it would access the sequence to get a decimal.

There are many operations to guarantee the chaotic characteristics of the sequence:

- First, after studying the logistic map, we find that at the first phase of the recursive process, X_n doesn't have a good chaotic characteristic. That is to say when the equation starts, the 1st amount values of X_n are not so random. In order to guarantee the randomness, we cut the 1st part of sequence, in the experiments; we generated a sequence of 100000 random numbers and started from the 2000th number.
- Second, in the method, we defined a variable to be the index of the sequence. From above we can find that when the method initiates, the index points the 2000th number of the random numbers sequence. And when we access the sequence and get a random number of the sequence, the index would point to the next number of the sequence. When it comes to the end of the sequence, the index would start from the head. Clearly we used the modulo operation to implement this
- Third, we use the current time of the system as an initial variable of X_n in the equation. After we get the current time, we adapt this long type number into a decimal ranges from 0 to 1 as the X_n. and because the X_ns are different, the sequences of random numbers which is generated by the logistic map are different. Here we defined a time-out time. Consequently, when the method is not used, after the time-out time period the method would get an X_n based on the current time of the system and generate a brand new sequence of random numbers of the logistic map. This new sequence would substitute the old one. This can be described as the sequence is timely updated. By using these 3 operations, the randomness of the sequence is guaranteed.

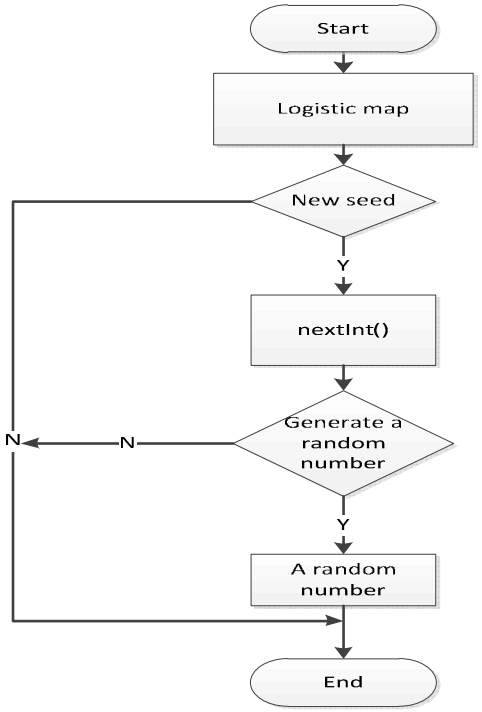

Fig. 2: Process of LRMOJ

Time cost of the LRMOJ mainly focuses on the generation of the sequence of random numbers; the access process and the sequence update process. From the detailed description we can find the generation of the sequence is done when the method initiates and it is before the random number generation process. Also it is done just once rather than every time. When it comes to the access process, the method would access the sequence in the memory. It is faster comparing with the arithmetic operations and bitwise and operations. The sequence update process is almost the same with the generation of the sequence and it would be used less if the user doesn't want to generate too many numbers. Comparing with the RMOJ, we deduce that LRMOJ is faster and much more efficient. The process of LRMOJ is described in Fig. 2.

The logistic map we mentioned is the original logistic map. Even though it is widely used but the original logistic map also has many drawbacks. It mainly has 2 obvious drawbacks. The 1st is the islands of stability problem. In the logistic map, when r is greater than 3.57, there are still certain isolated ranges of r that show non-chaotic behaviors; these are sometimes called islands of stability. Obviously if the numbers are generated in the islands of stability, these numbers are not chaotic or not random. This drawback sometimes can affect the chaotic features of the logistic map. The 2nd problem is the values of generated X_{n+1} is

no uniform. Recording the values of generated X_{n+1} and statistics the frequency of values, we can find they are no uniform. The frequency of 0.6 to 0.8 is much higher than others. In order to solve these 2 problems, there are many kinds of improved logistic map. Here we adopted one. It is Eq. (2) (Jianquan and Qing, 2010). Here the k is another coefficient. It must greater than 668.7 according to the study in the study:

$$X_{n+1} = (k * r * X_n * (1 - X_n)) \bmod 1 \quad (2)$$

Through a series of experiments, it is proved that this improved logistic map solves the problems of islands of stability and no uniform. It is more chaotic and uniform than the original logistic map. And in Eq. (2), mod 1 operation makes the result range from 0 to 1, actually the same range with the original logistic map.

We call this improved LRMOJ the ILRMOJ. In ILRMOJ, we replace the logistic map with the improved logistic map. Thus the steps of ILRMOJ are as followed:

Step 1: Get a decimal from the improved logistic map sequence numbers and amplify the decimal into a long type number as a new seed.

Step 2: Next Int method uses the new seed to generate a random number according to some specific operations including bitwise operations.

In the next chapter, we would demonstrate the comparison experiments.

We can tell from the RMOJ process, every time the old seed must do arithmetic operations including multiply and plus operations and a bitwise and operation. It costs much time. While by using the logistic map to generate new seeds, we only need to put the sequence of numbers into memory and access them when in need. This reduces much time.

Because the logistic map has many chaotic characteristics, we deduce that the sequences of numbers which are generated by the logistic map are of good randomness.

EXPERIMENTS

We did many experiments to assess the LRMOJ and ILRMOJ. When assessing a random number generation algorithm, we mainly focus on two aspects. They are randomness and efficiency. We compared RMOJ, LRMOJ and ILRMOJ in these two aspects respectively.

Randomness: There is no doubt that randomness is the main consideration of a random number generation algorithm. There are many kinds of tests to test the randomness, like the Kendall and Smith's tests, frequency test, serial test, poker test, gap test and so on (Wolfram, 2002). Here we used a basic test method frequency test to test RMOJ, LRMOJ and ILRMOJ. We let the three random number generation algorithms generate 10000 integral numbers and these numbers are

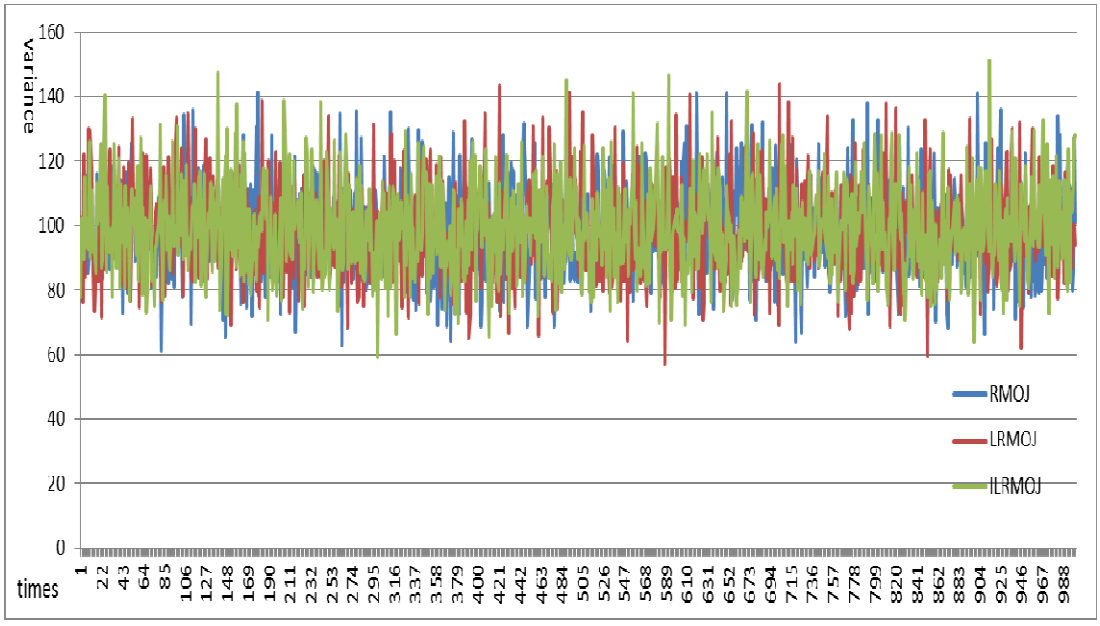

Fig. 3: Randomness comparison in a round of experiments

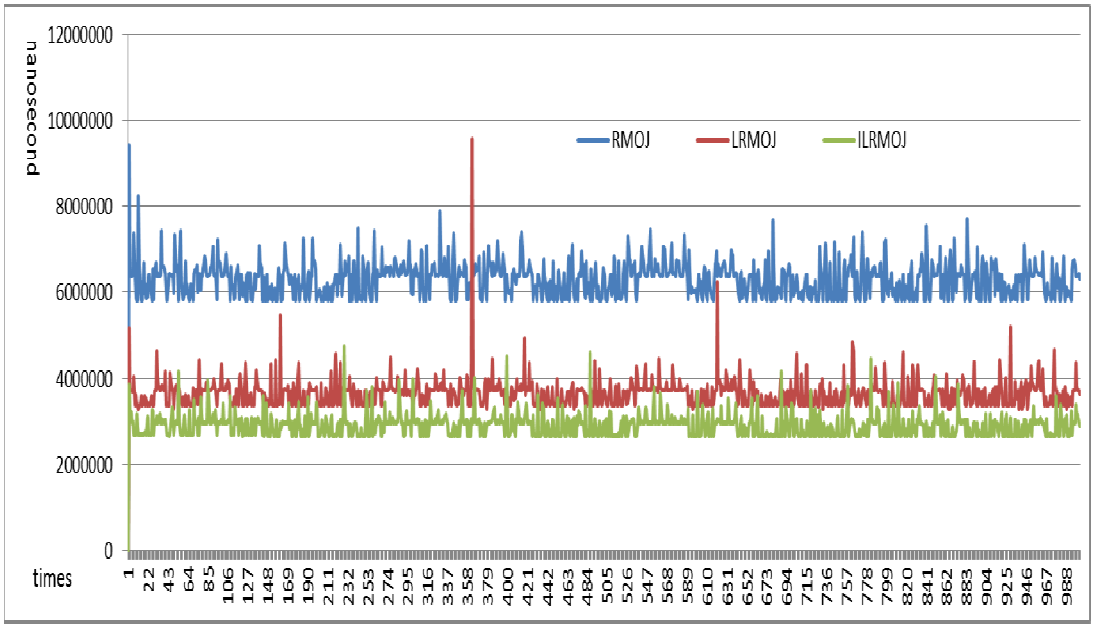

Fig. 4: Comparison of efficiency

from 0 to 99. And then we recorded these 100 numbers' appearance frequency. Obviously, the ideal and average appearance frequency of each number is 100. Actually, we compared each numbers' real appearance frequency with the ideal appearance frequency and calculated the variance of each sequence numbers. We can conclude that the less the variance is, the better the randomness is. We set a round of experiments contained 1000 times experiments.

For one single round of experiments, we displayed the figure of variance of each method respectively in Fig. 3. We can tell that the gap between three kinds of curves is not much and the curves in Fig. 3 are not of much difference. The average variance of RMOJ is 98.94, LRMOJ is 98.91 and the ILRMOJ is 99.13. We did another test to make every algorithm generate 1000000 numbers which are from 0 to 100. And the rest is the same with the former experiment. After getting the curve graph of variances, we can still find the three curves are almost over lap. And the variances range from 80 to 160. That is to say after increasing the number of generated numbers, the three algorithms are still having almost the same randomness.

As is shown in Table 1, after 1000 rounds experiments, we found that the situations, in which the sum of the variance RMOJ is greater than the LRMOJ's, appeared 501.4 times in average. While the opposite situations appeared 498.6 times in average. In other words, the situations in which the RMOJ performed better appeared 498.6 times in average, the LRMOJ performed better appeared 501.4 times in average. From the statistical numbers, we can conclude that the LRMOJ and the RMOJ had almost the same randomness. In the same way, we compared LRMOJ and ILRMOJ. The LRMOJ performed better appeared 489.3 times in average and the ILRMOJ performed better appeared 510.7 times in average.

Table 1: Randomness comparison

Times	RMOJ	LRMOJ
1	490	510
2	520	480
3	519	481
4	498	502
5	473	527
...	...	...
1000	497	503
Average	498.6	501.4

Efficiency: Efficiency of a method always indicates the availability. Time is a good standard to measure efficiency. As we mentioned above, according to the process of two improved random methods, we deduced that the two improved methods could reduce generating time. Here we recorded the time of generation of 100000 integral random numbers to measure the efficiency. The integral numbers ranged from 0 to 100. As is shown in Fig. 4, after 1000 times of experiments, the time of generating 100000 random integer numbers by RMOJ is around 6299910.72 ns. In average, while the LRMOJ is around 3647619.76 ns in average and the ILRMOJ is 2906039.53 ns in average.

And we can tell from Fig. 4, the LRMOJ is around 900000 ns around 358th experiment and it is around 600000 ns around 610th experiment. That means this method is not stable in some experiments. Sometimes the time is way too much longer than the average value. While the ILRMOJ and RMOJ don't have this situations. That means these 2 methods are much more stable than the LRMOJ.

Experiments summary: We improved the random method of Java with using the logistic map. Briefly, we change the seed of the random method of Java to the sequence numbers which are generated by the logistic map. Here the logistic map includes the original logistic map and the improved logistic map. In this way, we simplify the process of generating new seed and substitute the arithmetic operations and bitwise operations with access the sequence of numbers which are generated by the logistic map.

Obviously we concluded that the efficiency of the LRMOJ and ILRMOJ is way much better than the RMOJ. And the efficiency of ILRMOJ is better than LRMOJ's.

In conclusion, the LRMOJ, RMOJ and ILRMOJ almost have the same randomness. But the efficiency of the LRMOJ and ILRMOJ is much better. And ILRMOJ is the best.

CONCLUSION

Through the experiments, we concluded that the RMOJ, LRMOJ and the ILRMOJ almost have the same randomness. But the efficiency of the LRMOJ and ILRMOJ is much better; the efficiency of the ILRMOJ is the best. According to the results of experiments, these two improved random methods are of good quality and good efficiency. As for generating random numbers by algorithm, these 2 improved methods have many advantages. Because there are more and more researches and experiments which need to use random method to generate random numbers like in cryptography field, image processing and so on, we are sure that they can be widely used and make contributions.

On one hand, after generating the sequence of numbers with the logistic map, we store the sequence of numbers to the memory, which takes relatively much memory. In the future, we are about to find a better solution. And no matter which method of these three methods, after generating the new seed, a same method Next Int would be used. And we are about to study this shifts operations method deeply and find whether we can find a better method to replace it in order to make this method much more efficient.

On the other hand, we only improved the random method of Java. Because of the chaotic properties of the logistic map, we may apply the logistic map to other more random number generation algorithms. Even though through the frequency test, we concluded that the 3 algorithms had almost the same randomness, we need a series of deliberate tests to make the conclusion much more persuasive.

REFERENCES

Heam, P.C. and C. Nicaud, 2011. Seed: An easy-to-use random generator of recursive data structures for testing. IEEE 4th International Conference on Software Testing, Verification and Validation (ICST), France, pp: 60-69.

Jianquan, X. and X. Qing, 2010. Security analysis and improvement of an encryption algorithm based on logistic map. J. Chinese Comput. Syst., 31(6).

Kanter, I., Y. Aviad, I. Reidler, E. Cohen and M. Rosenbluh, 2010. An optical ultrafast random bit generator. Nature Photonic., 4(1): 58-61.

Persohn, K.J. and R.J. Povinelli, 2012. Analyzing logistic map pseudorandom number generators for periodicity induced by finite precision floating-point representation. Chaos Solitons Fract., 45(3): 238-245.

Wagenaar, W.A., 1972. Generation of random sequences by human subjects: A critical survey of the literature. Psychol. Bull., 77(1): 65-72.

Wolfram, S., 2002. A New Kind of Science. Wolfram Media, pp: 975-976, ISBN: 1-57955-008-8.

Xiao-Jun, T., C. Ming-Gen and J. Wei, 2006. The production algorithm of pseudo-random number generator based on compound non-linear chaos system. International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP '06, China, pp: 685-688.

Zarei, M.I., A.S. Rostami and M.R. Tanhatalab, 2010. Designing a random number generator with novel parallel lfsr substructure for key stream ciphers. International Conference on Computer Design and Applications (ICCDA), Iran, 5: V5-598-V5-601.