Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 8; April-June, 2015 pp. 34-36

http://www.krishisanskriti.org/acsit.html

A New Algorithm for Generating Pseudo Random Number Sequence

Vikas Pareek[1], Harshita Gupta[2], Jhilmil Gupta[3], Shreya Chatterjee[4] and Astha Jain[5]

[1,2,3,4,5]Banasthali University, Newai, Tonk
E-mail: [1]er_pareekvikas@yahoo.co.in, [2]gupta.harshita12@gmail.com, [3]jhilmil271994@gmail.com, [4]chatterjeeshreya.felix@gmail.com, [5]jainastha1994@gmail.com

Abstract—*Pseudo random number generation is an area of study vital to cryptography, gaming, modeling and numerous other fields. Taking inspiration from the phenomenon of ball-collision, which is an excellent example of random movement, we have proposed a novel algorithm for generating pseudo random bits using the ball-collision model proposed by us. In the end, we have discussed about a standard testing suite, the NIST test suite, which needs to be passed, for the generated sequence to exhibit good quality randomness.*

1. INTRODUCTION

For that which may seem random to us, is actually determined by Him…..

Even the most unordered things in nature have an underlying orderliness. The challenge lies in finding order from this chaos. To defeat this challenge in the technical world, various deterministic algorithms have been put forward, which generate seemingly random numbers.

Deterministic algorithms are those which, given a particular input, will always produce the same output. But the beauty of these algorithms (speaking in context of random number generation), is that even if one has the algorithm, one cannot determine the output, unless the key/seed is available, thus making the output seem random to user.

A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequently reliably reproduced. In most cases, the determination of randomness is done, not for one particular number, but for a sequence of independent random numbers with a specified distribution. Random numbers can be generated using computer, by two approaches: True Random Number Generator(TRNG) and Pseudorandom Number Generator(PRNG).

TRNG is a generator which produces true random numbers by extracting randomness from any physical phenomenon and introducing it into a computer[19].

PRNG is also known as quasi-random number generator[2] .The basic idea of a PRNG is to take a few truly random bits as input(also known as the 'seed' value), and generate a sequence of n bits that appear to be random.

2. PROPOSED ALGORITHM

I) CONCEPT

On observing and pondering upon the various natural phenomena occurring around us, one can be marveled at finding underlying orderliness in those which seem random. The phenomenon which inspired this algorithm is the collision of a ball, when moved inside a container, with the walls of the container. This ball, if given an initial push, keeps moving around in the container, hitting its walls one by one, until stopped by an external force. In this process, the ball does not follow any predefined trajectory. Its movement is random and chaotic.

The key idea behind our algorithm is to capture the random movement of a ball inside a 2-dimensional frame. We give the ball an initial push, by providing it with seed values (i.e., the initial angle along which the ball will move and the initial distance from where the ball will start moving). By keeping a track on the ball's movement, we will be able to capture the exact points at which the ball hits the walls one by one. These random points will be outputted as random numbers/bits. Thus at the end of 'n' iterations, a sequence of random numbers is generated (where 'n' denotes the number of random numbers/bits requested by the user).

Further, to increase the randomness of the numbers, we have introduced 8 such containers, with varying sizes. There will be a ball in each container. A container is chosen randomly, the ball inside it is moved, and its position is captured. Using that position, the next container is selected. Thus, with each pass, a different container is selected and its respective ball movement captured. Since the size of each container is kept different, the numbers generated are evenly spread over the range (from very small numbers to very large numbers).

Following is an algorithm explaining the working of our code:-

3. ALGORITHM

/* Frames taken will be 2-dimensional and each frame will be of different parameters (length and breadth).

Seeds are taken as inputs from a True Random Number Generator(from Random.org). Among the seeds are:-

Angle, at which the ball will be inclined for its movement; this angle will be with respect to the axis parallel to the wall on which it is positioned.

Distance, from where the ball will start its movement; this distance will be along the axis parallel to the wall on which it is positioned.

Frame Initiator, a number selecting one of the 8 frames, in which the next ball movement will take place.

All the distances are captured either along the x-axis or the y-axis, according to where the ball is positioned. The frames are set such that the origin of the frame is the lower limit of the range provided by user and the other 3 corners are plotted as per the highest limit of the range provided.

*/

Step 1: Input the following from the user:-

(i) Whether user wants a sequence of **random numbers**, or sequence of **random bits**.
(ii) If random numbers then maximum range of the sequence of numbers to be generated.
(iii) How many numbers/bits to be generated?

Step 2: Convert the **frame initiator** into a binary string.

Step 3: From the binary string, extract the last 3 bits. Convert them into a decimal number.

Step 4: According to the decimal number generated, a frame is selected.

Step 5: Ball is moved inside the selected frame according to the previously stored values of **seeds** of this particular frame (at the **intended angle** and from the **intended distance**).

Step 6: The position where the ball hits a wall inside the frame, after its movement, is captured. The angle with which the ball hits the wall is also captured.

/* Position here indicates the distance of the ball from the set origin, along the axis parallel to the wall on which the ball is positioned. */

Step 7: The position and angle are stored for future movement of the ball inside this particular frame.

Step 8: If user wants random numbers, go to Step 9. If user wants random bits, go to Step 10.

Step 9: Output the captured position as the next random number.

Step 10: Convert the position into a bit and output this bit as the next random bit.

Step 11: Concatenate the generated bit with the previous binary sequence generated in Step 2.

Step 12: Go to step 3.

Step 13: Exit when the required number of bits are generated.

EXAMPLE:-

Seed values-> angle=56

Maximum value=10000

Minimum value=100

Position=500

No. of numbers/bits=10

Output-> 4259.511

2871.2126

4007.0022

615.7717

4396.685

1139.3693

3043.0383

120.86505

2480.77

3150.6296

4. RESULTS AND DISCUSSIONS

A. Statistical Testing

After the algorithm has been defined, it is supposed to be checked for the correctness of results. A sequence is considered random when it passes a set of statistical tests in polynomial time. The result of these tests is successful if the set of tests cannot successfully distinguish between a sequence of true random numbers and the one generated by us. For convenience, NIST provides a set of predefined tests to ensure the randomness of the generated sequence.

THE NIST TEST SUITE

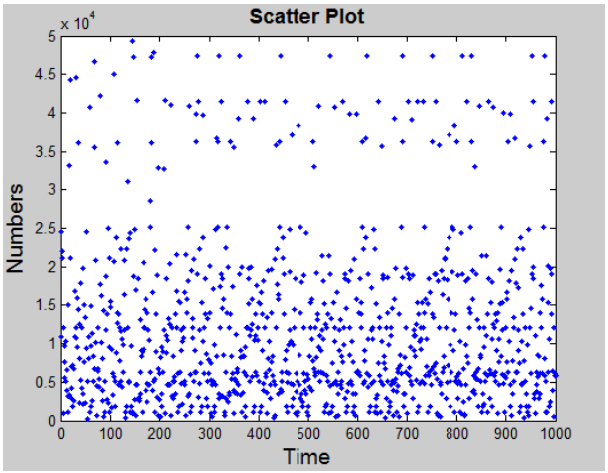
- Frequency (Monobits) test:
- Test for Frequency within a block:
- Runs test:
- Test for the longest run of ones in a block:
- Random Binary Matrix Rank Test:
- Discrete Fourier Transform (Spectral) Test:
- Non-Overlapping (Aperiodic) Template Matching Test:
- Overlapping (Periodic) Template Matching Test:

- Maurer's Universal Statistical Test:
- Linear Complexity Test:
- Serial Test:
- Approximate Entropy Test:
- Cumulative Sum (Cusum) Test:
- Random Excursions Test:
- Random Excursions Variant Test

From the NIST suite, we find that the proposed algorithm passes its test if the length of the sequence is small (1000), but for larger (10^9) lengths, the algorithm becomes slow and does not pass NIST criteria

B. Scatter plot of the numbers generated

A graph of the numbers generated, with respect to time is plotted to see how uniformly the numbers are distributed over a given range.



5. CONCLUSION

As shown in the scatter plot above, and as deciphered from the NIST test suites's results, this algorithm exhibits reasonable randomness, and is hence suitable for simple modeling and simulation uses. However it can be improved further to make it suitable for cryptographically secure applications.

REFERENCES

[1] Ankur Rathi, Divyanjali Sharma, and Vikas Pareek, A New Approach to Pseudo random Number generation, "4th International Conference on Advanced Computing & Communication Technologies (ACCT)", published in IEEE Xplore and Digital Library by CPS, 8-9 February 2014. ISBN: 978-1-4799-4910-6/14 2014 IEEE, DOI: 10.1109/ACCT.2014.26, pp. 290-295.

[2] M.Blum and S.Micali, "How to generate cryptographically strong sequences of pseudo-random bits", November 1984.

[3] D.Knuth, "The Art of Computer Programming: Seminumerical Algorithms", Vol. 2, Addison-Wesley Pub. Co. 1981.

[4] Stephen K.Park and Keith W.Miller, "Random Number Generators: Good Ones Are Hard To Find", October 1988.

[5] Pascal Junod, "Cryptographic Secure Pseudo-Random Bits Generation: The Blum-Blum-Shub Generator", August 1999.

[6] Pat Burns, "Linear, Congruential Random Number Generators", 2004.

[7] Chung Chih Li and Bosum, "Using Linear Congruential Generator for Cryptographic Purposes".

[8] D.T. Downham and F.D.K. Roberts, "Multiplicative Congruential Number Generators".

[9] Lenure Blum, Manuel Blum and Michael Shub, "Comparison of two pseudo-random number generators", 1983.

[10] Marsaglia, "Random Number Generators", 2003.

[11] Brian Gerhardt, "Cryptographic PseudoRandom Numbers"

[12] Bruce Schneier, "Applied Cryptography", 2nd Edition.

[13] J.Plumstead, "Inferring a sequence generated by a Linear Congruence", 1982.

[14] Dromey, "How to solve it by Computer", Pearson Education, India, 2008.

[15] A.Shamir, "On the generation of cryptographically strong pseudo random sequences", February 1978.

[16] Josefin Agerblad, Martin Andersen, "Provably Secure Pseudo-Random Generators: A Literary Study"

[17] William Stallings, "*Cryptography and Network Security: Principles and Practice*", Fifth Edition, Pearson Education, Inc., Prentice Hall, 2011

[18] A.J. Menzes, S.A. Vanstone, P.C.V. Oorschot, "*Handbook of Applied Cryptography*", 1st Edition, CRC Press, Inc., Boca Raton, FL, USA, 1996

[19] https://www.random.org/

[20] http://www.cs.indiana.edu/~kapadia/project2/node11.html

[21] http://crypto.stackexchange.com/questions/3454/blum-blum-shub-vs-aes-ctr-or-other-csprngs

[22] http://www.anomaly.org/Thinair/charactr.html

[23] http://www.pit-claudel.fr

[24] http://www.nist.gov