A Study of Pseudo-Random Number Generators in Software Systems

Vandan Amin
Student ID: 44006979

Department of Computer Science, Central Washington University
CS465/565 - Scientific Computing

Dr. Donald Davendra
October 21, 2024

Contents

Abstract

Pseudo-random number generators (PRNGs) are important tools for introducing randomness in software systems. This report provides an overview of common PRNG algorithms, their properties, and applications. The analysis includes a comparison of their performance in terms of speed, randomness quality, and complexity, offering insights into their use in software systems.

0.1 Introduction

Pseudo-random number generators (PRNGs) are important in optimization and modeling by adding randomness to systems. This report explores different PRNGs used in software, detailing their advantages, disadvantages, mathematical descriptions, and where they can be used.

0.2 Common PRNGs

0.2.1 Linear Congruential Generator (LCG)

The Linear Congruential Generator (LCG) is one of the oldest and simplest methods for generating pseudo-random numbers. It uses a linear equation to produce a sequence of values that appear random based on an initial seed. The sequence is deterministic, meaning the same seed always produces the same series of numbers (GeeksforGeeks, n.d.).

Mathematical Description:

$$X_{n+1} = (aX_n + c) \mod m$$

where a, c, and m are constants, and X is the sequence of pseudo-random values (GeeksforGeeks, n.d.).

Advantages:

- Simple to implement and fast.
- Consistent, reproducible results, useful for testing.
- Widely supported in many programming languages.

Disadvantages:

- Short period if parameters are not chosen well, leading to predictability.
- Patterns can appear, reducing randomness quality.
- Not suitable for secure applications.

Platforms: LCGs are widely used in general-purpose applications and simulations, especially where simplicity and speed are required. They are implemented in languages like C, Python, Java, and MATLAB, where they are often the default or standard PRNG for non-secure uses (GeeksforGeeks, n.d.).

0.2.2 Mersenne Twister

The Mersenne Twister is a widely used PRNG known for its long period and high-quality randomness. Developed in 1997, it is a standard choice for applications needing reliable random numbers (Matsumoto & Nishimura, 1998).

Mathematical Description: The Mersenne Twister uses a linear recurrence relation over a finite field (GF(2)), involving bitwise operations such as shifts and XORs applied to an internal state vector. The standard version, MT19937, has a period of $2^{19937} - 1$, ensuring a long cycle before repeating (Matsumoto & Nishimura, 1998).

Advantages:

- Very long period, reducing repetition.
- High-quality randomness.
- Suitable for simulations and statistical applications.

Disadvantages:

- More complex than simpler PRNGs.
- Slower compared to LCG in some cases.

Platforms: The Mersenne Twister is used in a wide range of applications, from simulations to gaming, where a high-quality random sequence is necessary. It is implemented in popular languages such as Python (in the `random` module), C++ (as part of the standard library), and R, making it a default choice for many scientific computing tasks (Matsumoto & Nishimura, 1998).

0.2.3 XORShift

XORShift is a fast PRNG that uses bitwise operations to generate random sequences. Introduced by George Marsaglia, it is well-suited for environments where speed is critical (Marsaglia, 2003).

Mathematical Description: Uses bitwise XOR and shift operations:

$$X \oplus = X \ll a; \quad X \oplus = X \gg b; \quad X \oplus = X \ll c$$

Advantages:

- Very fast and simple to implement.
- Good randomness for many applications.

Disadvantages:

- Shorter period compared to Mersenne Twister.
- Not suitable for secure purposes.

Platforms: XORShift is commonly used in performance-critical applications, such as video games and simulations, where speed is a priority over cryptographic security. It is implemented in languages like C, Python, and JavaScript, providing an efficient solution for scenarios where moderate randomness is acceptable (Marsaglia, 2003).

0.3 Comparison of PRNGs

PRNG	**Period**	**Speed**	**Quality of Randomness**	**Complexity**
Linear Congruential	Short	Fast	Moderate	Low
Mersenne Twister	Very long	Moderate	High	Moderate
XORShift	Moderate	Very fast	Good	Low

Table 1: Comparison of Common PRNGs

0.4 Conclusion

PRNGs are important tools in modeling and optimization, providing the randomness needed for effective design and control. Different PRNGs have different strengths and weaknesses, and their use depends on the specific requirements of the application. For example, LCGs are simple and fast, suitable for basic uses, while the Mersenne Twister provides high-quality randomness for more demanding tasks. XORShift is ideal for environments requiring high speed with moderate randomness. Choosing the right PRNG is important for ensuring system efficiency and reliability.

0.5 References

1. GeeksforGeeks. (n.d.). Linear congruence method for generating pseudo-random numbers. GeeksforGeeks. https://tinyurl.com/3dpummpn

2. Marsaglia, G. (2003). Xorshift RNGs. *Journal of Statistical Software, 8*(14). https://www.jstatsoft.org/article/view/v008i14

3. Matsumoto, M., & Nishimura, T. (1998). The Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Mathematical Software, 24*(1), 3-30. https://doi.org/10.1145/272991.272995