MA252 - Design and Analysis of Algorithms

Mini Project Report

Mohammad Humam Khan (180123057)

Kartikeya Singh (180123021)

PSEUDO-RANDOM NUMBER GENERATION

Introduction

The problem of random number generation has been of utmost importance since the evolution of computers. Due to various applications in cryptography, random number generation has become a hot topic for research among computer scientists. The question is whether we can use computational algorithms to generate truly random numbers or the sequence will start repeating itself after a period. Since it is very difficult to get a computer to do something by chance because a computer follows its instructions blindly and is therefore completely predictable. Hence to deal with this problem of random number generation, computer scientists came up with the concept of pseudo-random number generators which would for most of our practical purposes behave as random number sequences.

A random number generator (RNG) is a computational mathematical algorithm that is designed to generate a random sequence of numbers that should not display any distinguishable patterns in their appearance or generation.

A pseudorandom number generator, also known as a deterministic random bit generator, is an algorithm for generating a sequence of numbers whose properties resemble the properties of sequences of random numbers. The PRNG-generated sequence cannot be said to be truly random, because it is determined by an initial value, called the PRNG's seed. The seed may be any number, but it usually comes from seconds on a computer system's clock. Several computational methods for pseudorandom number generation exist, but all of these methods fall short of the goal of true randomness.

Our report mainly focuses on studying the state of art of the problem of random number generation from the point of view of efficiency of various algorithms i.e. how random is the sequence generated by the PRNG.

Background

One of the earliest approaches to this problem was made by John Von Neumann in 1946 when he suggested the Middle-Square method. This method works as follows :

1. Take a random number.
2. Square it.
3. Take the middle digits of the resulting number as our random number.
4. Use this number as seed for the next iteration.

However, the middle-square method had a drawback known as “zero mechanism” i.e. once the middle digits of the resulting number became zero, further numbers of the sequence became zero.

In 1949, Mathematician D.H Lehmer made significant progress in this field with an algorithm called the linear congruence generator. The generator is defined by the recurrence relation :-

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

Where a, c, m are fixed constants, X is the sequence of randomly generated numbers, with X_0 as the seed. This method is popularly known as multiplicative congruential generator (MCG), or Lehmer RNG. This algorithm was fast and required minimal memory to retain state. However it had several drawbacks and could not pass Spectral test which is a simple test of LCG’s quality and randomness.

In 1958, G. J. Mitchell and D. P. Moore came up with the Lagged Fibonacci Generator(LFG) based on the Fibonacci sequence which was an improvement of standard LCG. The mathematical formula that describes this generator is:-

$$S_n \equiv S_{n-j} \star S_{n-k} \pmod{m}, 0 < j < k$$

Where m is generally a power of 2. * is a general binary operation such as addition, subtraction, multiplication, or bitwise arithmetic XOR. j and k are chosen randomly such that 0 < j < k . This generator achieves maximum period when j and k satisfy a condition given by:

$X^j + x^k + 1$ is primitive over integers mod 2

All values S_0, S_1 , ... , S_{k-1} should also be provided prior to computation.

In 1997, Makoto Matsumoto and Takuji Nishimura developed the Mersenne Twister Algorithm for the generation of pseudo-random numbers. The period for this algorithm is chosen to be a Mersenne Prime (primes of the form 2^n - 1). The most commonly used version of the Mersenne Twister algorithm, known as MT19937 uses the Mersenne prime 2^{19937}−1. The Mersenne Twister is the default random number generating algorithm in the standard libraries of most modern-day programming languages such as C++ and Java.

Middle - Square Weyl Sequence PRNG

The drawbacks, especially zero mechanism, associated with the original middle square method by Neumann were overcome by Bernard Widynski in 2019 by using the middle square method with a Weyl sequence resulting in a PRNG generator which was faster than all previous Pseudo-Random Number Generators and passed all statistical tests. A Weyl sequence is an integer stepping sequence *(0,k,2k, ...)* with all multiplications modulo m which is also known as the period. Most commonly , a period of 2^{64} is used in the PRNG. An integer *k* is chosen, relatively prime to the integer modulo *m*. In the common case that *m* is a power of 2, this amounts to requiring that *k* is odd. The sequence of all multiples of such an integer *k*, is equidistributed modulo *m* (Refer Theorem 1*)*.

The Algorithm is as follows:-

1. A 64-bit seed x is taken.
2. The seed is squared to produce a 128 bit number but only the least significant 64 bits are stored in x (due to memory constraints) .
3. A Weyl sequence number w is added to the squared seed.
4. The middle 32 bits are extracted as our result and the resultant is used as the seed for the next iteration.

Algorithm

```
#include <stdint.h>
uint64_t x = 0, w = 0, s = 0xb5ad4eceda1ce2a9;
inline static uint32_t msws() {
    x *= x; // Square the seed
    w += s; // Next number in the weyl sequence
    x += w; // Add the Weyl Sequence Number
    return x = (x>>32) | (x<<32); // Extract Middle 32 bits
}
```

The state vector of this PRNG consists of three 64-bit words: x, w, and s where x and w are computed while s is a constant that provides for distinct initialization. If x or w are used for initialization then there are chances of overlapping.

The Weyl sequence overcomes the zero mechanism and keeps the generator running for long periods by preventing repeating cycles. Now we will establish this fact by proving that no two of the first 2^{64} elements of the Weyl sequence w can be equal which will imply that there can be no repeating cycles in our sequence of random numbers.

Theorem 1

For the Weyl sequence that is generated by the formula w += s, where w is an unsigned 64-bit integer and s is an odd constant, no two of the first 2^{64} generated elements can be equal.

Proof:

We will prove this theorem by contradiction.

The mathematical equivalent of w += s in C is as follows:

For i = 0 **to** $2^{64} - 1$, and some odd constant $s < 2^{64}$

$w(i) = (i * s) \bmod 2^{64}$

Now, assume that there exist an m and n with $m < n < 2^{64}$ such that w(m) = w(n).

$\Rightarrow n * s - m * s = k * 2^{64}$ for some integer k

$\Rightarrow (n - m) * s = k * 2^{64}$

Then there are two cases:

Case-I

Here we have odd times odd equal to some k times even, which implies that odd is equal to even which is not possible.

Case-II

Let (2^j)∗ p be the factorization of (n - m) for some integer j and odd integer p. Then we have :

$$p*s*2^j = k*2^{64}$$
$$\Rightarrow p*s = k*2^{64-j}$$

Now we had $0 < j < 64$ because (n-m) is even and $(n-m) < 2^{64}$.

This implies odd is equal to even which is impossible.

Since we arrived at a contradiction which means our assumption was wrong.

Hence no two of the first 2^{64} numbers can be equal.

Theorem 2

For the first 2^{64} iterations of the PRNG, there will be no repeating cycles in x.

Proof:

Consider two different x values in the first 2^{64} iterations that are equal.
Now, the RNG performs three operations viz square, add Weyl sequence, and shift.

1. The square operation will yield the same result.
2. Adding the Weyl sequence value w (which we know from Theorem 1 is different) will give different results.
3. After performing circular shift operation also, the numbers will be different.

This implies that the next x value after each of the equal x values will be different.
Hence there can be no repeating cycles in x for the first 2^{64} iterations of the RNG which imply that period of x will be at least 2^{64}

Using Weyl sequence provided an additional advantage i.e. a basis for uniformity in output. The Weyl sequence is uniform and hence adding it to the square of a random number produces a uniform output.

This PRNG is cryptographically secure. It is nonlinear and hence the only possible attack would probably be brute force. The state vector has three 64-bit elements: x, w, and s. Using brute force, one could crack the state with $(2^{64})^3$ or 2^{192} combinations of x, w, and s.

Though the middle square was invented at the very beginning of computer science, modern 64-bit computing architecture made it possible to create a usable version which

has a sufficiently long period (2^{64} per stream) with processing speed comparable with the fastest RNGs.

Conclusion

In this report, We have tried to discuss the state of art of the problem of pseudo random number generation with special emphasis on the middle square weyl sequence algorithm. We also briefly discussed various other algorithms for generation of pseudo random numbers. In a talk in 1949, Von Neumann quoted that, "Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin." What he meant was that there were no true "random numbers", just means to produce them, and "a strict arithmetic procedure", like the middle-square method, is not such a method. Although middle square weyl sequence is the fastest PRNG till date that has passed all statistical tests but the scope of improvement is still present and the generation of true random numbers remains an unsolved problem in Computer Science till date. Due to vast applications of PRNG algorithms in cryptography, this area is of much interest to many computer scientists and research still goes on.

References

1) https://en.wikipedia.org/wiki/Pseudorandom_number_generator (Definitions only)
2) https://arxiv.org/pdf/1704.00358v4.pdf (Main work focussed on this paper --- Middle Square Weyl Sequence)
3) https://mcnp.lanl.gov/pdf_files/nbs_vonneumann.pdf (Middle square method)
4) https://dl.acm.org/doi/pdf/10.1145/272991.272995 (Mersenne Twister Algorithm)
5) https://www.math.cmu.edu/~af1p/Texfiles/LINEARCONGRU.pdf (Linear Congruential Generator)