Programmatic Entropy:
Exploring the Most Prominent PRNGs

Jared Anderson, Evan Bause, Nick Pappas, Joshua Oberlin

A Recap of Randomness

Random number generating algorithms are rated by two primary measures: entropy - the measure of disorder in the numbers created and period - how long it takes before the PRNG begins to inevitably cycle its pattern. While high entropy and a long period are desirable traits, it is sometimes necessary to settle for a less intense method of random number generation to not sacrifice performance of the product the PRNG is required for. However, in the real world PRNGs must also be evaluated for memory footprint, CPU requirements, and speed.

In this poster we will explore three of the major types of PRNGs, their history, their inner workings, and their uses.

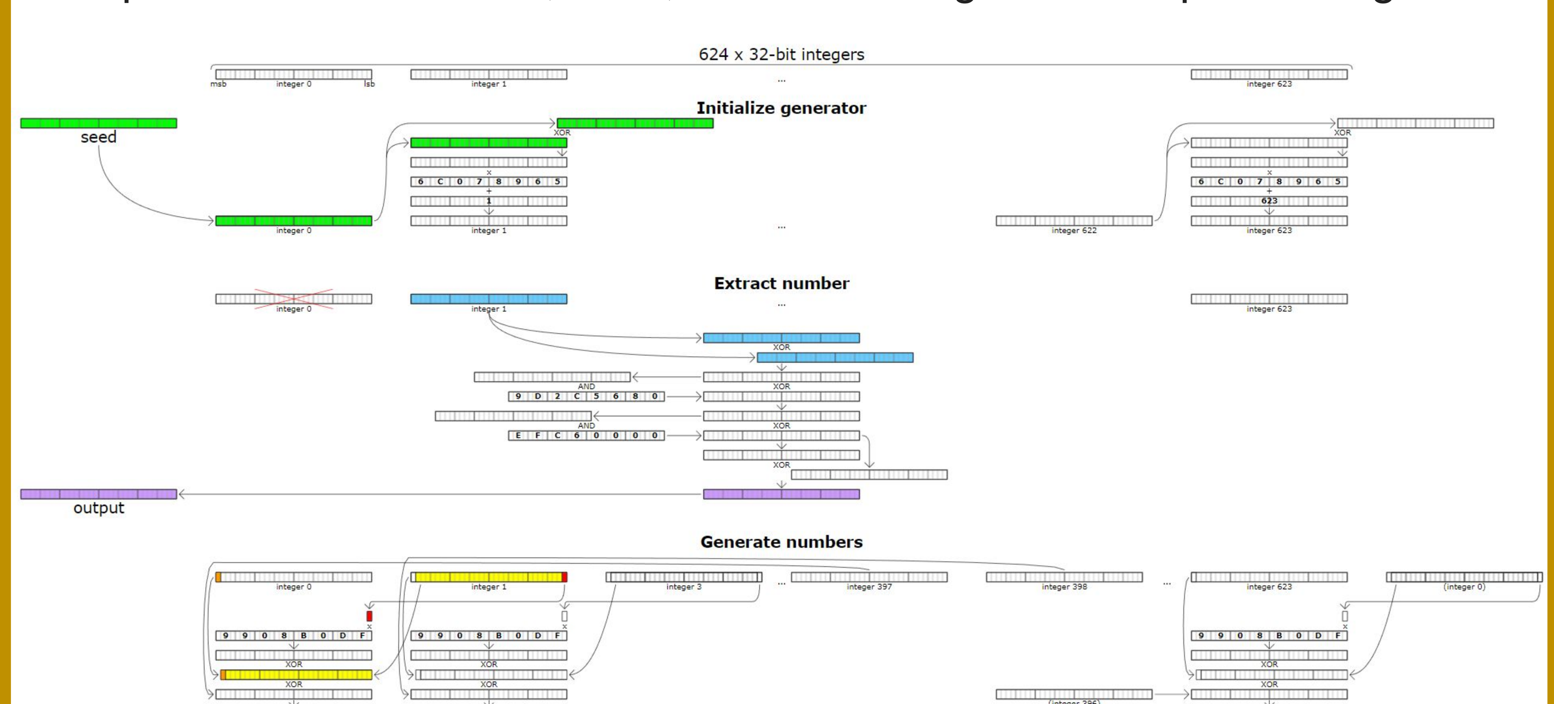
The Mersene Twister

The Mersenne Twister is the most widely used general purpose pseudorandom number generator today. A Mersenne prime is a prime number that is one less than a power of two, and in the case of a mersenne twister, is its chosen period length (most commonly $2^{19937}-1$). There are only 50 such numbers known to exist today, with the most recent, $2^{77,232,917}$-1, only being found on January 3rd of this year. Makoto Matsumoto and Takuji Nishimura developed the algorithm in 1997 to overcome the flaws found in older PRNGs, being the first PRNG to provide high entropy, long period random number generation in little time. Because of this, the Mersenne Twister is the default pseudorandom number generator for software systems such as Microsoft Excel, GAUSS, GNU, IDL, MATLAB, Python, R, and Ruby.

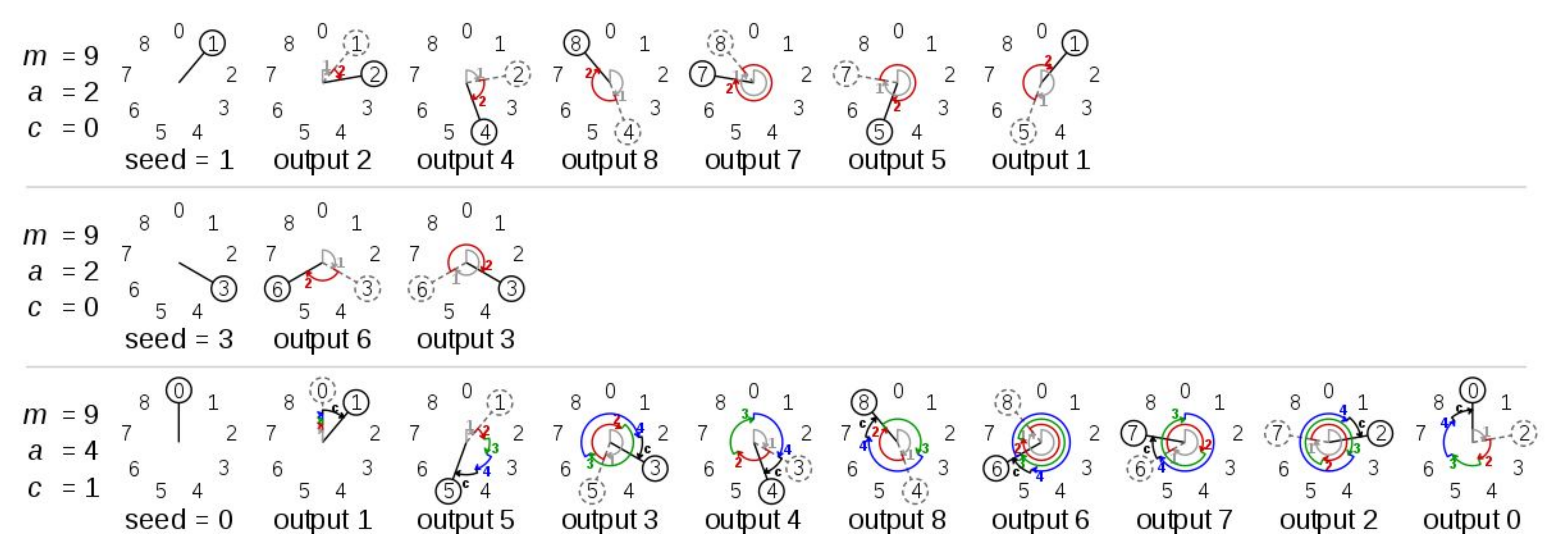
There are many reasons to choose the mersenne twister over other PRNGs. As stated, it has a truly impressive speed and quality combination, producing even 64-bit floats 20x faster than hardware based solutions, while also passing statistical tests for randomness like TestU01 and Diehard. It is also patent-free, so it and variants of its base algorithm can be used freely without worry of cost or expensive hardware.

The Mersenne Twister is, however, not a silver bullet for all PRNG needs. While a variant, CryptMT, exists, it is normally not cryptographically secure, disallowing its use where security is a major concern such as password encryption or gambling. It also requires a large state buffer of 2.5 KiB, and has mediocre throughput by modern standards, meaning it shouldn't be used on hardware with too little buffer or in situations where large streams of random numbers are needed.

Mersenne Twister takes a seed which is initialized into a 624 x 32bit integer state through multiplication, xor, and bit shifting. Then the state is manipulated with a "twist" function. After the state has been twisted at least once, the "temper" function can be applied to the state yielding up to 624 random numbers before it needs to be twisted again. Twist and Temper functions use Xor, AND, and bit shifting for faster processing.



Linear Congruential Generators



Linear congruential generators form a series of numbers using the recurrence relation formula: Xn = (a Xn-1 + c) mod m

EX: a = 2, c = 3, m = 10, X_0 = 5. Output: 5 3 9 1 5 3 9 1 5 3. . .

Notice that after four numbers have been generated in the previous example, the sequence begins to repeat. This is due to a modular arithmetic that forces wrapping of values into the desired range resulting in the period of 4. The value that primarily affects the period of the sequence is m, where larger m's result in a much longer period.

Unfortunately, a long period does not guarantee a random sequence. A sequence made from this formula can have sufficient randomness but a very short period, or it can have a long period with an obviously non random sequence. The most common way to work around this problem is to remove c from the equation by making it 0. With c, the formula can be referred to as being a mixed congruential generator. Removing c creates a formula used for a pure multiplicative generator:

Xn = aXn-1 mod m

This results in a simpler algorithm, so many random number algorithms use the second equation over the first.

An important note about the pure multiplicative generator formula is that aXn-1 cannot generate 0, or every subsequent value will become 0. This is not an issue with with mixed congruential generators (c is not equal to 0) because if aXn-1 results in zero, the constant c would be added to it.

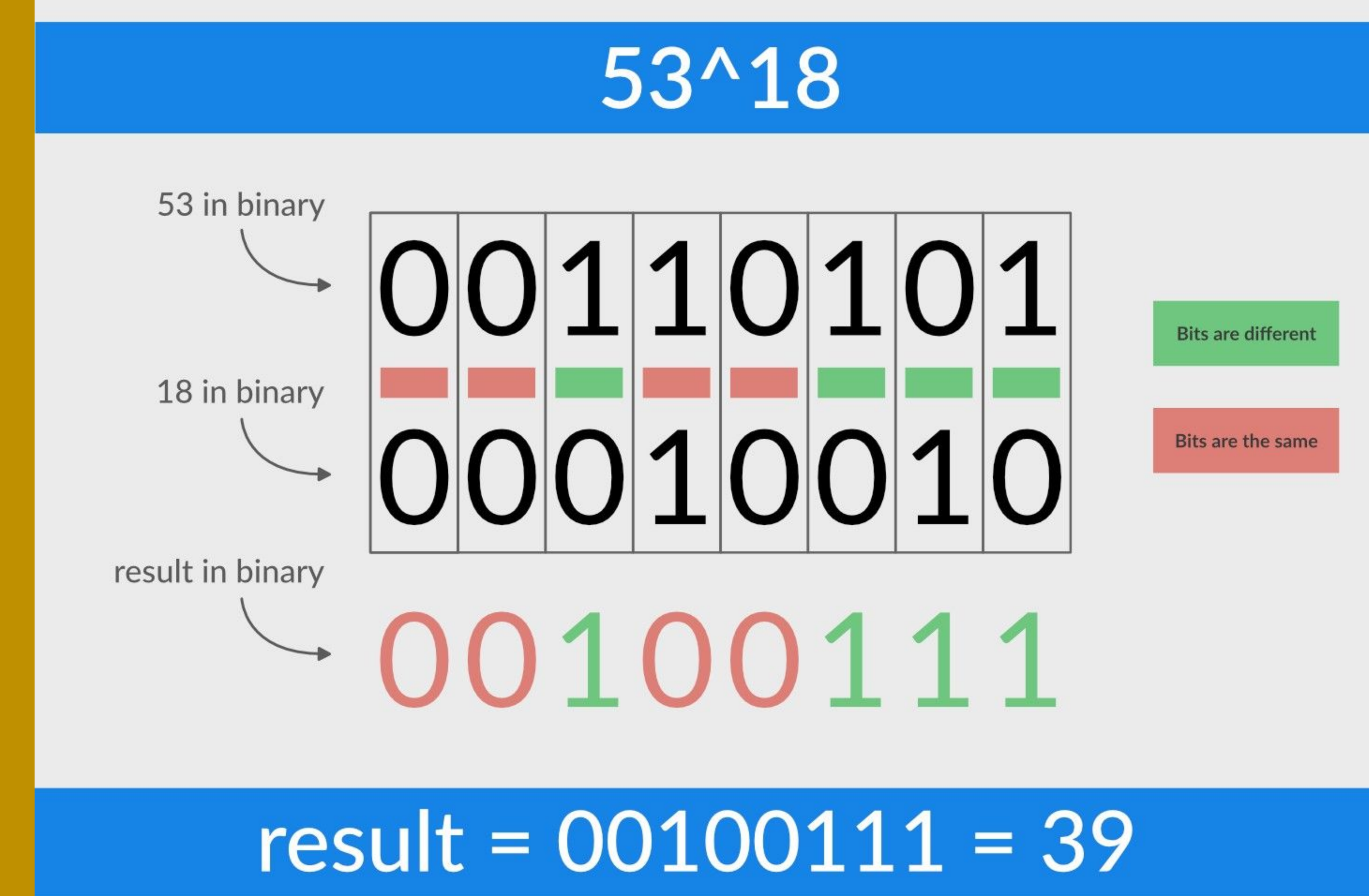
A major advantage of Linear Congruential Generators is that they require minimal memory and therefore are generally faster than other PRNGs. They are ideal for when an application's requirements for high quality randomness are not essential.

Being faster than other PRNGs, however, makes it less efficient than others. It is highly recommended to use a different more random generator for high level applications. LCGs are much better suited for smaller level applications, such as an environment like a video game console.

In conclusion, LCGs better fit smaller applications where speed is of critical value, and high levels of entropy are not.

Xorshift

Xorshift random number generators are an extremely fast and efficient type of PRNG discovered by George Marsaglia. An xorshift PRNG works by taking the exclusive or (xor) of a computer word with a shifted version of itself. For a single integer x, an xorshift operation can produce a sequence of 232 - 1 random integers. For a pair of integers x, y, it can produce a sequence of 264 - 1 random integers. For a triple x, y, z, it can produce a sequence of 296 - 1, and so on.



An xorshift algorithm is incredibly useful because it's such a simple algorithm, able to be written with only a few lines of code, and the sequence it creates do very well on tests of randomness. It is one of the fastest known PRNGs, being able to generate over 200 million random numbers a second in most cases.

Some of the drawbacks of xorshift include its lack of cryptological security. It also fails many tougher randomness tests, such as those of TestU01's BigCrush suite, however this is true for all PRNGs based on linear recurrence, including the Mersenne Twister. It has also been described as being unreliable. However, all of its flaws are well known and easily amenable.

Some variations of the xorshift include xorshift*, which applies a nonlinear transformation to a normal xorshift operation. This increases the period of the output, increasing its total randomness. Another variation is xorshift+ which uses addition instead of multiplication as a nonlinear transformation, which both increases the randomness and speed of the algorithm since addition is a faster operation for the arithmetic logic units of CPUs.

Resources: https://en.wikipedia.org/wiki/Mersenne_Twister, http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ewhat-is-mt.html, http://www.quadibloc.com/crypto/co4814.htm, http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_rand.aspx, https://en.wikipedia.org/wiki/Xorshift, http://www.jstatsoft.org/v08/i14/paper, https://hackernoon.com/how-does-javascripts-math-random-generate-random-numbers-ef0de6a20131,