

---

# Evolvable Random Number Generators: A Schemata-based Approach

---

Robert K. Watkins

Jason C. Isaacs

Simon Y. Foo

Department of Electrical Engineering  
FAMU-FSU College of Engineering,  
Tallahassee, FL 32310 USA

## Abstract

*We have recently succeeded at developing Genetic Algorithm (GA)-based Random Number Generators (RNG) by encoding generators in genomes that manifest themselves as combinations of basic mathematical operators and randomized seeds/parameters. The chief difficulty with this approach is that arbitrary combinations of operators and parameters often produce stillborns (genomes that cannot be decoded as RNG). We have designed an encoding that avoids this difficulty, and refer to it as a schemata-based RNG. With our schemata-based RNG, the combination of operators and parameters are "optimized" by selecting the "most fit" algorithm as measured by our GA's objective function (a metrized version of the Federal Information Protection Standard 140 (FIPS-140) statistical tests). Moreover, offline testing (with Marsaglia's Diehard test suite) shows that our schemata-based RNG perform well when compared to a variety of RNG in use today.*

## 1 INTRODUCTION

Somewhat paradoxically, our principle weapon against complexity is randomness. More and more, scientists are using computer simulations to approach problems that are insoluble by analytic techniques. An important, though often overlooked, component in every such simulation is the RNG. In recent years, the need for efficient, reliable random number generation has grown continuously, and this trend will likely continue into the foreseeable future. We are thus engaged in research aimed at meeting this demand by producing an evolvable RNG for implementation on a Field Programmable Gate Array (FPGA). The advantages of this approach are numerous, and include: 1) a near inexhaustible supply of 'new' RNG; 2) RNG that can be optimized for various reliability criteria; and, 3) greatly increased speed

(implementation of the RNG in hardware will result in a much more rapid execution-- and in parallel, if desired).

The first step towards our goal was the development of a GA-based RNG. Our search for an RNG that could be evolved by a GA has itself evolved through several generations. Our first attempts included unsuccessful experiments at serial employment of RNG, seed optimizations, and anti-bias filtering. Success came when we struck upon the idea of evolving the primitive characteristics of known RNG. The last of these we refer to as a schemata-based GA-RNG.

## 2 SCHEMATA-BASED GA-RNG

Genetic algorithms, as developed by Holland (1975), are a means by which machines can emulate the mechanisms of biological genetics and natural selection. In GA, the parameters of a specific problem can be encoded into strings of finite length, just as the chromosomes (strands of DNA) encode for all of the characteristics of human beings through chains of amino acids. While a string can be a sequence of any symbols, our encoding for the schemata-based RNG utilizes integers in the range [0, 15] to represent mathematical operators and seed/parameters, as follows:

0001010509091203030605070010141313060004151007

Digraphs in the even positions refer to seed/parameters, while odd positions refer to operations. For example, a 00 in the zero position (even) is decoded as a seed 15881431; 01 in the one position (odd) is decoded as the addition operator; 01 in the two position specifies the second addend 3367900313; and so on to produce (Fig. 1) ...

```
[0|1|1|5|9|9|12|3|3|6|5|5|7|0|10|14|14|13|13|6|0|4|15|10|7|2|7|20|66|
15881431 + 3367900313 + 957690767 ©80309 + 65699
- 4231 + e48539 mod 4180125777 5©59669 - 102217 *
4186401 shift3·bits*37061 ©28597477 = 16·bit Number
```

Figure 1. Decoded Gene

The specific parameters returned during decoding are "randomly" selected from stored lists of appropriate values. Of course, every GA is a copious consumer of random numbers, but our gene decoder also requires random digits-- all of these are provided by bootstrapping the GA-RNG to seed an XORassignment RNG (Isaacs & Watkins, 2000). The final two positions in the genome ( 20 | 66 in the example, above) are tags to indicate the string's "fitness" and are used for "optimizing" the population. Optimization is performed on the set of strings by using a fitness function. Our fitness function is a metrized version of the FIPS-140 (described in detail, in section 3.2.1, below).

In outline, the GA (Figure 2) requires: initialization of a population of strings, decoding and testing of the RNG (strings) for selection, and prescribed iterations (generations) of reproduction, crossover, mutation, decoding, testing, and further selection of strings.

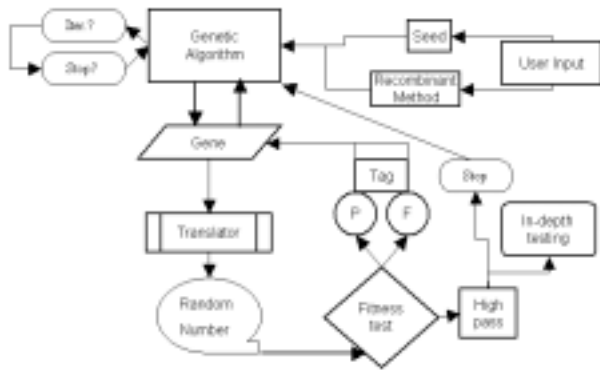


Figure 2. Schematic of Schemata-based GA-RNG.

We have experimented with varying a number of GA parameters to optimize RNG behavior, including: selection, recombination, and mutation techniques. Results from these experiments have been incorporated into the algorithm to help weed-out weak populations.

## 2.1 SELECTION

The selection operator in a GA selects chromosomes from the population for reproduction. In most cases, the fitter the chromosome (according to the fitness function), the more times a chromosome will be selected to reproduce. Example selection methods are elitist, roulette, tournament, and hybridized combinations of these. In an elitist selection, only the top performing individuals are selected for reproduction. This has the advantage of culling out weak performers, but also promotes inbreeding and can cause population performances to settle at local maxima. We have experimented using an elitist selection mechanism preserving the top 20% of each generation. Typical results for 100 generations of evolution are shown in Figure 3, next column. In a tournament selection, top performs are preferentially selected for reproduction, but weak performers are not

necessarily eliminated. We applied this selection strategy to a population of 20 chromosomes, by preserving the top 5 performers in each generation and providing the winner with 5 randomly selected mates; the first runner-up had 4 mates; the second runner-up, 3 mates, and so on. Typical results from our experiments using a tournament selection mechanism are presented in Figure 4, below. And finally, a roulette selection method (completely random selection) decouples objective function performance and reproduction selection criteria. It clearly avoids inbreeding, but does not actively promote the transmission of "strong" genes. Figure 5 shows experimental results incorporating a roulette selection in our schemata-based GA-RNG.

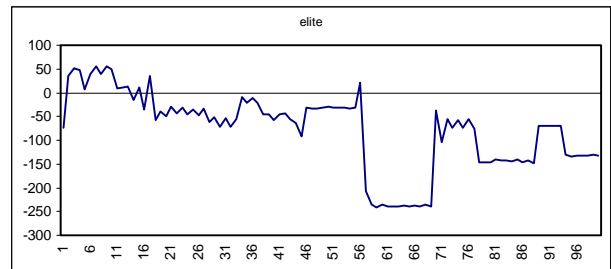


Figure 3. Average Performance over 100 Generations using an Elitist Selection Mechanism

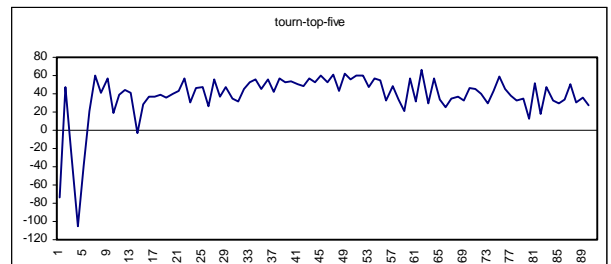


Figure 4. Average Performance over 100 Generations using a Tournament Selection Mechanism

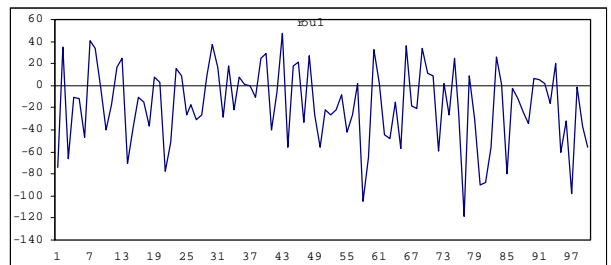


Figure 5. Average Performance over 100 Generations using a Roulette Selection Mechanism

## 2.2 RECOMBINATION

This operator chooses a position on the genome and exchange sub-sequences (before and after) the locus

between two chromosomes to create two offspring. We employed both a midpoint and a random point crossover initially. Figures 6 & 7, show experimental results obtained from applying these recombination techniques to our schemata-based GA-RNG. From these results we have decided the best method for our problem would be a random point single-site crossover.

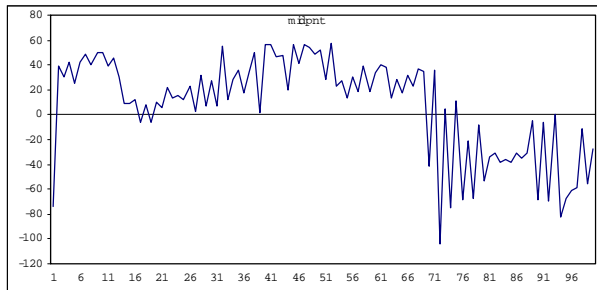


Figure 6. Average Performance over 100 Generations using a Fixed-point Crossover Recombination

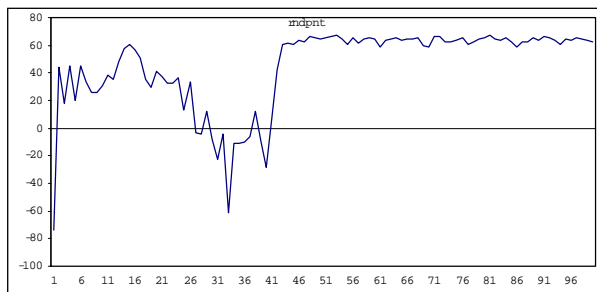


Figure 7. Average Performance over 100 Generations using a Random-point Crossover Recombination

### 2.3 MUTATION

This operator randomly inverts some of the bits in a chromosome. For example, a 1 can be changed to a 0. Our 0-15 integer encoding means that there is a probability equal to  $(1/28 * 14/15) * \text{mutation rate}$  of a change at any one locus. Typically, we use a mutation rate between 10 and 12 percent. This means a change occurs with a probability of .0039 at any position in gene.

## 3 RANDOMNESS

In 1927, the social scientist Leonard Tippett published the first table of random numbers. These were generated by the laborious process of drawing numbered balls from a sack (Bennett, 1998). If proper care is exercised, such a process will generate random digits. The essential feature of a true random number generator is that it avoids being deterministic by tapping some naturally occurring source of randomness. This is, however, much less useful than one might first believe, because natural sources of

randomness are often easily influenced by environmental factors and produce biased bit streams (*i.e.*, the probability of producing a 1 is not equal to the probability of producing a 0) or they show correlations between successive bits (*i.e.*, the probability of the next bit being a 1 is influenced by what bits have been produced recently). While numerous mechanical devices have been designed to produce true random number sequences, without anti-biasing filters, none are completely successful at producing digits both uniformly distributed and uncorrelated.

### 3.1 QUASI- and PSEUDO-RANDOM NUMBERS

Although they are not truly random, bit sequences produced by various deterministic means can appear just as random as, and usually exhibit greater uniformity than, true random sequences. While there is still debate over the definition of randomness (with some abjuring the concept altogether), among those whose job it is to produce random digits the concept of randomness is well-defined (in terms of probability distributions and statistical tests). For these people, the concept of randomness admits of varying gradations, and there are many performance specific RNG available. In some applications, uniformity is the overriding criteria; in others, it is equally important that the sequence be unpredictable. In general, uniformity is an easier measure to satisfy and generators that produce uniform, disordered streams (but which may have correlations among the bits) are referred to as quasi-random number generators (QRNG). Generators that produce uniform streams with no (or very few) correlations between the bits are referred to as pseudo-random number generators (PRNG).

### 3.2 TESTS FOR RANDOMNESS

The performance of an RNG is measured by computing various test statistics. There are few guarantees that RNG will perform as predicted in all situations. A variety of tests have been developed to determine the likelihood that a sequence of numbers is "random". The most stringent of these are batteries of tests that include dozens of independent, computationally intensive tests. It is important to realize that these tests do not indicate when a sequence is random; they are strictly elimination tests indicating when a sequence is probably not random. Moreover, the tests are probabilistic and, hence, not definitive. Even if a sequence passes the tests, it is not a guarantee that the generator will produce random numbers.

Each test begins by using the RNG to produce a sequence  $a_0, a_1, a_2, \dots, a_{n-1}$  of length  $n$ , where  $n$  is determined by the specific test(s) to be employed. We use two batteries of test: our fitness function is a modification of the Federal Information Protection Standard 140, (FIPS-140); for in-depth (offline) testing we currently use Diehard (Marsaglia, 1983).

### 3.2.1 FIPS-140

The FIPS-140 battery includes four tests (Beker & Piper, 1985) to determine whether a binary sequence  $\{a_n\}$  possesses some of the specific characteristics that any random sequence must exhibit (in the long-run).

**Monobit test:** This test indicates whether the number of 0's and 1's in  $\{a_n\}$  are approximately the same (as would be expected in a random sequence). With  $n_0, n_1$  denote the number of 0's and 1's, respectively. The statistic is:

$$X_1 = \frac{(n_0 - n_1)^2}{n} \quad (1)$$

For  $n > 30$ ,  $X_1$  should agree with a chi-square distribution with 1 degree of freedom

**Blocks and Gaps test:** this test indicates whether the number of blocks (runs of ones) and gaps (runs of zeros) of various lengths occur at the expected frequency for the sequence  $\{a_n\}$ . In a random sequence the number of blocks (or gaps) of length  $l$  of length  $n$ , is  $e_l = (n - l + 3)/2^{l+2}$ . With  $k$  equal to the largest integer  $i$  for which  $e_l \geq 5$ , and  $B_l, G_l$  the number of blocks and gaps, respectively, of length  $l$  in  $\{a_n\}$  for each  $l, 1 \leq l \leq k$ . The statistic used is

$$X_2 = \sum_{l=1}^k \frac{(B_l - e_l)^2}{e_l} + \sum_{l=1}^k \frac{(G_l - e_l)^2}{e_l} \quad (2)$$

$X_2$  approximately follows a chi-square distribution with  $2k - 2$  degrees of freedom.

**Max Run test:** this test determines if the longest run of 1s or 0s is of an acceptable length. For a sequence of 20,000 bits this should be no more than 34.

**Poker test:** With  $k$  the greatest integer less than or equal to  $n/m$  and  $k \geq 5 * 2^m$ , divide  $\{a_n\}$  into  $k$  non-overlapping segments of length  $m$ , and let  $n_i$  be the number of occurrences of the  $i^{th}$  type of sequence of length  $m$ . The poker test indicates if the segments of length  $m$  each appear at the expected rate. The statistic used is

$$X_3 = \frac{2^m}{k} \left( \sum_{i=1}^{2^m} n_i^2 \right) - k \quad (3)$$

for a random sequence,  $X_3$  should agree with a chi-square distribution with  $2^m - 1$  degrees of freedom.

In adopting the FIPS-140 as our fitness function, we have made two modifications: 1) the FIPS-140 uses a Poker Test on four-bit non-overlapping sequences; we have increased the stringency of the test by using an 8-bit test;

2) to allow rank ordering of the RNG, we have imposed a metric on these tests, assigning points in inverse proportion to a test statistic's absolute deviation from the expected value.

### 3.2.2 OFF-LINE TESTING (DIEHARD)

Since our ultimate objective is to write a GA-RNG to hardware, the choice of the FIPS-140 as a fitness function was strongly influenced by its compactness. However, a number of other tests have been proposed (Knuth, 1981; Beker & Piper, 1985; Maurer, 1990; Marsaglia & Zaman, 1993; NIST, 2001). For the last decade, Marsaglia's Diehard battery has been widely considered the most stringent.

Diehard includes 18 tests to determine whether a binary sequence  $\{a_n\}$  possesses specific characteristics that any random sequence must exhibit (in the long-run). The tests are: BIRTHDAY SPACING; OPERM5; 31x31 BINARY RANK; 32x32 BINARY RANK; 6x8 BINARY RANK; BITSTREAM; OPSO; OQSO; DNA; COUNT-THE-1's; (selective)COUNT-THE-1's; PARKING LOT; MINIMUM DISTANCE; 3D SPHERES; SQUEEZE; OVERLAPPING SUMS; RUNS; and CRAPS. (These tests are described in greater detail in (Isaacs, 2001)). They include both frequency counts on words 20-bits and less and simple Monte Carlo experiments intended to detect autocorrelations. The required input file is 20 million bits.

To accurately assess the quality of RNG produced by the GA, offline testing is a necessity. We have tested numerous RNG against both the FIPS-140 and Diehard. Diehard is considerably more stringent. Our GA fitness metric is such that scores of 50+ pass the FIPS-140, however, we have produced tens of thousands of generators with fitness scores of 60+ and many of these fail one or more of the Diehard tests.

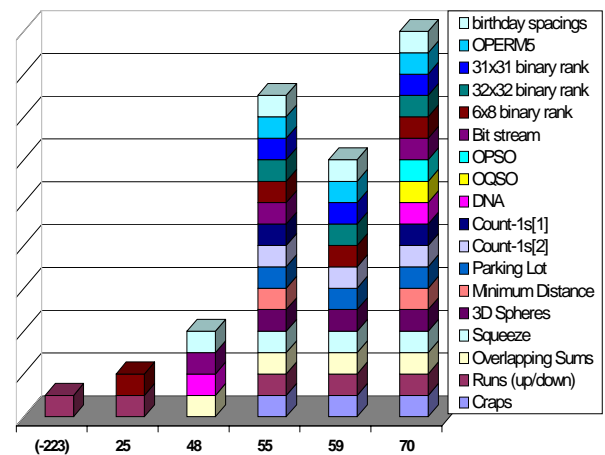


Figure 8: Diehard tests passed v. FIPS scores.

## 4 CONCLUSIONS / FUTURE WORK

We are pleased with the progress we have made thus far, but there are a number of difficulties impeding a hardware implementation inherent in our schemata-based GA-RNG approach. Our original intent was to implement a GA-RNG on a single FPGA. This approach relied heavily on the ability to store multiple configurations (or contexts) on-board and switch between them. The hardware platform chosen for this project was the context switching reconfigurable computing (CSRC) FPGA, under development by Sanders. Due to product development delays, we switched our hardware platform to the XESS XSV800 Virtex prototyping board, which features the Xilinx Virtex XCV800 FPGA. Unlike the CSRC, which is still in development, the XCV800 was not originally designed to perform context switching. However, our work thus far makes us hopeful that we can orchestrate context switching on an appropriately configured XSV800. In either case, we intend to write the GA-RNG in sub-modules to as many as five XSV800s and coordinate their operations via the XC9500 Complex Programmable Logic Device (CPLD). The switch in platforms means we will have to write our GA-RNG in even less space than originally planned.

Our schemata-based RNG do perform well when compared to various known RNG. But, we suspect that it will be difficult (perhaps impossible) to give these schemata-based behemoths of code a full implementation in hardware. As such, we will continue to generate, test, and catalogue schemata-based GA-RNG, but we must also continue our search for an RNG that can both be evolved by a GA and implemented on an FPGA.

Our attention has recently shifted to RNG that depended exclusively on hardware friendly computations. One such approach, which shows great promise, is the use of Ant Colony Simulations (Isaacs, 2001).

## 5 ACKNOWLEDGEMENT

The authors are grateful to Mr. Alan Hunsberger for his contributions and the Department of Defense for sponsorship of this work.

## 6 REFERENCES

Beker, H. & Piper, F., *Secure Speech Communications*. Academic Press, 1985.

Bennett, D., *Randomness*, Harvard University Press, 1998.

Holland, J. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

Isaacs, J., Watkins, R. & Foo, S., *Evolvable Ant Colony Systems for Pseudo-Random Number Generation*, Conference Proceedings: GECCO 2001 (late-breaking papers), submitted.

Isaacs, J., & Watkins, R., *An XORassignment QRNG: Pass the FIPS-140 using 148 gates and a 9 bit seed*, Technical Paper, 2000.

Knuth, D., *The Art of Computer Programming. Vol 2: Seminumerical Algorithms*. Addison-Wesley, 1981.

Marsaglia, G. & Zaman, A., *Monkey Tests for Random Number Generators*, Computers and Mathematics with Applications, 1993.

Mauer, U., *A Universal Statistical Test for Random Bit Generators*. *Advances in Cryptology-- CRYPTO '90*, Springer-Verlag, 1990.

NIST Special Publication 800-22, "A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications," May 15, 2001