International Journal of Soft Computing 5 (5): 200-205, 2010
ISSN: 1816-9503

Chaotic Random Number Generator and It's Evaluation by Genetic Algorithm

[1]Hamed Rahimov, [1]Mohsen Farhadi and [2]Majid Babai
[1]Faculty of Computer Engineering,
Shahrood University of Technology, Shahrood, Iran
[2]Faculty of Computer Engineering,
Iran University of Science and Technology, Shahrood, Iran

Abstract: Pseudorandom number generators are used in many areas of contemporary technology such as modern communication systems and engineering applications. In recent years a new approach to secure transmission of information based on application of the theory of chaotic dynamic systems has been developed. The idea of Constructing Random gystems based on chaotic (CRNG) intrinsically exploits the property of extreme sensitivity of trajectories to small changes of initial conditions/system parameters and other properties can be considered analogous to the confusion, diffusion with small changes in plaintext/secret key, deterministic pseudorandom numbers and algorithmic complexity properties of traditional cryptosystems. As a result of this close relationship, several chaos based cryptosystems have been put forward since 1990. We discuss suitability of the chaos based random number in random number generator by using the genetic algorithm that can calculate approximate answer in optimization problems. The simulation's result showed that chaos based random number generators can generate more uniform than low-discrepancy random number generator.

Key words: Genetic algorithm, optimization, chaotic random number generator, deterministic pseudorandom, trajectories, Iran

INTRODUCTION

Pseudo-random number generators with good properties are frequently used in modern communication systems as well as in a variety of engineering applications. The quality of this means: How well a given chaotic system produces pseudorandom numbers that has independent uniformly distributed numbers? Many cryptographic schemes and protocols require a secure of random or pseudorandom numbers. The quality of this source is crucial for the security of the scheme or protocol in question.

For the largest value of the control parameter, the logistic map is able to generate an infinite chaotic sequence of numbers (Andrecut, 1998). The reverse Logistic map is a very simple mathematical model often used to describe the growth of biological populations. It's showed this simple model has complex behavior. The simple modified mathematical form of the reverse logistic map is given as Eq. 1.

$$f(x_n) = x_{n+1} = (1 - rx_n)^2 \quad 0 < x_n < 1 \tag{1}$$

Where:

x_n = The state variable which lies in the interval (0, 1)
r = Called system parameter which can have any value between 0 and 2

If the value of growth rate is equal to 0, then x_n tends to 1 i.e., the chaotic system has an absorbing point and this point is 1.

For $0<r<0.75$, x_n tend to the stable state i.e., likewise, the system has an absorbing point, the difference with that the value of absorbing point decreased with increasing value of r. In r = 0.75 is the first step of bifurcation. This type of vibration that x_n iterated both times called cycle with two periods. For $r \approx 1.25$ population leads to a cycle that has four stable states, previous cycle's periods duplicated and changed to cycle with four periods. With increasing the value of r iterate duplications of period continuously. This system's behavior called bifurcation event. In fact bifurcation event causes duplications of period. For larger value of r, not only the value of x_n tends to a fixed point or iterative orbit but it also has chaos behavior. In Fig. 1 has depicted one such example of sensitivity on initial conditions for

Corresponding Author: Hamed Rahimov, Faculty of Computer Engineering, Shahrood University of Technology, Shahrood, Iran

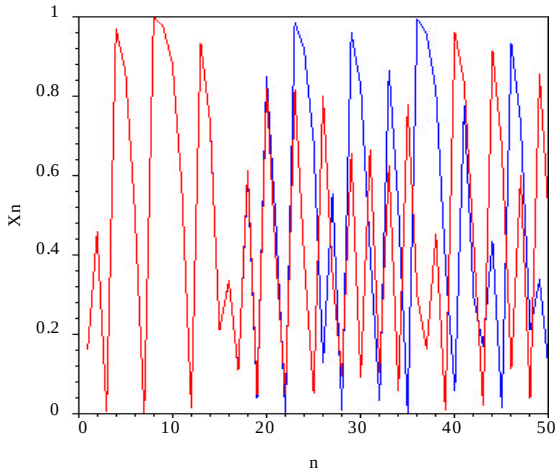

Fig. 1: The sequences of reverse logistic map, behavior of inverse logistic map and sensitivity of initial conditions for r = 1.99

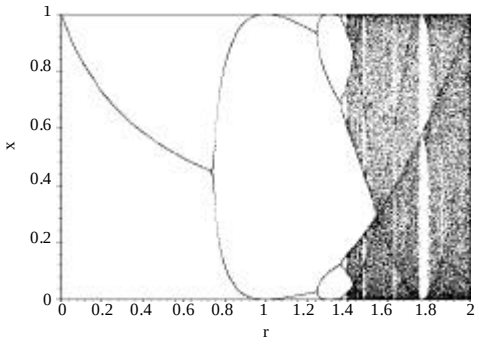

Fig. 2: Bifurcation reverse logistic function diagram, as 0<r<2

r = 1.99. It is clear that the two trajectories of the chaotic Inverse logistic maps starting nearby x_0 = 0.3 and x_0 = 0.3000001 soon diverge exponentially in the cross of time and have no coloration between them. If we calculate coloration coefficient for these two data sets confirms the completely unallocated behavior of two trajectories which are starting from almost same initial conditions.

The bifurcation reverse logistic function is shown in Fig. 2 to notice that the system hasn't chaos behavior for any r>1.4, so it's a mix of discipline and chaos in 1.4<r<2. Therefore for value of r that is 1.5<r<2, we can generate a chaos based sequence of $x_n \in (0, 1)$. Logistic map (function 2) and Parabolic map (function 3) are the other chaotic map samples that are shown in Fig. 3a and b.

$$f(x_n) = x_{n+1} = r \cdot x_n \cdot (1 - x_n) \quad 0 < x_n < 1 \tag{2}$$

$$f(x_n) = x_{n+1} = 1 - r \cdot x_n^2 \quad -1 \le x_n \le 1 \tag{3}$$

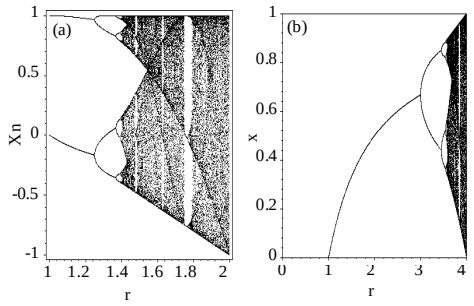

Fig. 3: Bifurcation (a) logistic map and (b) Parabolic map diagrams

In Fig. 3a and b, logistic map is chaotic system where r = 2 can generate $x_n \in [0,1]$ also a parabolic map is chaotic system where r = 2 can generate $x_n \in [1,1]$. Thus these systems have chaotic behavior.

Genetic algorithm is a search technique used to find approximate solutions to optimization and search problems. In this study we present a chaotic method to generate initial population's genetic algorithm (known as the parent population), so we use random numbers for parent population of some low-discrepancy and chaotic methods.

A genetic algorithm for optimization problem (Reese, 2009) is defined. That parent population was generated by Random Number Generator (RNGs), Pseudo Random Number Generator (PRNGs), Quasi Random Number Generator (QRNGs) and chaotic Random Number Generator (CRNGs) (Sen *et al*., 2006). Finally we ranked these generators with comparing precisions generated answers by GA.

GENETIC ALGORITHM

The first step of a genetic algorithm is to generate an initial population of candidate solutions by coding the parameter set as a finite-length binary string where each bit is assigned a meaning. The bit string is the concept definition language for the GA (Banzhaf *et al*., 1998). This binary string represents a real value of the variable. How should one choose the population size? It can be set anywhere from 10-1000,000. It has been suggested that for smaller problems, a population size of 1000 should be used; however, as the problem becomes more difficult, the population size should increase (Banzhaf *et al*., 1998). To generate the parent population, an array of random numbers is generated and then converted to binary strings. Each binary string will contain 22 bits so as to assure the required precision of six places after the decimal point. The binary strings are then converted to base 10 representation and then into a real number.

To illustrate this procedure, suppose the random number 0.9501 was generated (Reese, 2009). This value is then multiplied by 2^{22} to ensure 22 bits in the binary conversion. The decimal to binary conversion results in the string 1111001100111001110000.

The conversion from binary to base 10 results in 3985008. Lastly, the conversion to a real number requires the endpoints of the interval for the variable. Suppose a value between -1 and 2 is to be generated. Then the real number conversion is found by multiplying the base 10 representation by the length of the interval, dividing by $(2^{22}-1)$ and then adding the left endpoint of the interval. In this case:

$$x = -1+3985008*3\ (2^{22}-1) = 1.8503$$

Starting with the parent population of strings, the GA then generates successive populations of strings. This simulated evolution allows the good strings to reproduce and the bad strings to die off (through the reproduction operator). The search for better structures is based solely on the fitness values associated with the individual strings and is guided by probabilistic transition rules. Reproduction (also known as selection) is a process in which individual strings from the parent population are copied according to their objective function values (i.e., payoff values) and is the second step of the GA. It is an asexual process in that only a single string is involved (Baker, 1998).

Holland's method uses fitness-proportionate selection with roulette wheel. In this method, the chance of an individual string being selected is proportional to the amount by which its fitness is greater or less than its competitor's fitness (Mitchell, 1997) causing stronger strings to be selected over weaker strings. This method will produce a new generation of the same size as the parent population. Holland's method is conducted as follows. First, the absolute value of the fitness value of each parent string is calculated.

From this the total fitness of the entire population can be found by summing the individual fitness values. Then for each parent string, the probability of selection is calculated by dividing the individual fitness value by the total fitness. Lastly, the cumulative probability distribution is calculated. To select parent strings to copy, a random number between 0 and 1 (inclusive) is first generated. Using the cumulative probability distribution, if the generated random number lies within a parent string's cumulative probability range then that parent string is chosen for reproduction. For example, if the generated random number is less than the first value of the cumulative probability distribution, then the first chromosome will be selected. This random number generation is repeated until a new generation of the same size as the parent generation has been created. The third step of the GA is crossover (also known as partial string exchange). Crossover combines the features of two parent strings to form similar (i.e., of the same size) offspring by swapping corresponding segments of the parents (Zbigniew, 1996). The expected number of parent strings to crossover can be found by multiplying the population size by the probability of crossover (which is chosen by the user). According to Mitchell (1997), the probability of crossover can be set at the fairly typical value of 0.7. However, Banzhaf *et al*. (1998) suggest that the crossover probability should be set high, around 90%. Obitko and Slavik (1999) also suggest that the crossover rate should be set high about 80, 95%.

To generate the set of strings to be involved in crossover, a random number is generated for each string. If the random number is less than the probability of crossover, the chromosome is selected for crossover. If an odd number of strings are chosen to crossover, another string can be added or an existing string can be deleted. Strings are then paired together to undergo crossover. For each pair, a random number between 1 and (the total number of bits-1) inclusive is generated (in this case between 1 and 21 inclusive). Crossover will occur at the bit after the generated random number. This is called single point crossover and is the simplest method. To illustrate, suppose we have the following two strings selected for crossover:

0011101001111100110111 and
1100000011010111000100

If the generated random number was 5, crossover would occur after the fifth bit:

00111|0100111100110111 and
11000|0001101011100100

Each part after the crossover position is exchanged to form two new offspring:

0011100011010111000100 and
1100001001111100110111

Other crossover methods include two point crossover (where two crossover points are selected and the binary string from the beginning of the chromosome to the first crossover point is copied from the first parent and the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent), uniform crossover (where bits are randomly copied from the first or from the second parent) and arithmetic crossover (where some arithmetic operation is performed to make a new offspring) (Obitko and Slavik, 1999).

The last step in the GA is mutation. Mutation plays a secondary role in the operation of GAs (Goldberg, 1989). It is the occasional random alteration of the value of a string position. Mutation, like reproduction, is an asexual process. The expected number of mutated bits can be found by multiplying the probability of mutation (this is also chosen by the user) by the total number of bits (in this case, 22) by the population size. According to Mitchell (1997), the probability of mutation can be set at a fairly typical value of 0.001. However, Banzhaf *et al.* (1998) suggest that the mutation probability should be set low, around 10%. Obitko and Slavik (1999) also suggest the mutation rate should be set low, about 0.5-1%.

The procedure for mutation is as follows: for every bit in the population, a random number between 0 and 1 (inclusive) is generated. If the generated random number is less than the probability of mutation, the bit is mutated. If the original bit was 0, it now becomes a 1 and vice versa. Mutation and crossover can be performed in either order. However, it has been shown that mutation before crossover is less efficient than mutation after crossover (Vose, 1999). After crossover and mutation have been performed, the first generation (after the parent population) has been formed.

The fitness values of this newly formed generation are then evaluated. The GA should be repeated starting with reproduction using this first generation as the new parent population. The total number of generations to evaluate is often based on previous results and can range anywhere from 50 to >500 (Mitchell, 1997). One option for termination is to run the algorithm for a set number of generations (which is the approach used here). An optional approach is to end the algorithm after a certain number of generations pass with no improvement in the fitness of the best individual in the population.

NUMERICAL EXPERIMENT

To evaluate the test problems (many of which are violently fluctuating, i.e., containing many local maxima) the n-dimensional bisection method will be implemented.

This method is most commonly associated with root-finding, but it can also be used to show that a continuous function on a closed interval achieves its maximum (Wood, 1992). Bisection is the division into two equal halves of a given curve, figure or interval. The domain for one dimension (i.e., one variable) is a line segment. The one-dimensional bisection procedure for iteratively converging on a solution which is known to lie inside some interval (a, b) proceeds by evaluating the function at the midpoint of the original interval x = (a+b)/2 and testing to see in which of the subintervals [c, (a+b)/2] or [(a+b)/2,b] the solution lies. This procedure is then repeated with the new interval as often as needed to locate the solution to the desired accuracy.

For two dimensions (i.e., two variables), the domain is a regular hexagon. The bisection method will produce four search areas. For example, if the first variable lies between (a, b) and the second variable lies between (c, d) then the bisection method will produce two subintervals for each variable [(a+b)/2] and [(a+b)/2,b] for the first variable and [(c+d)/2] and [(c+d)/2,d] for the second variable. There are four possible combinations since there are two options for each variable.

The combination resulting in the largest function value will be chosen for the next iteration. For three dimensions, there would be eight subintervals in which to choose one for the next iteration. This procedure can be generalized for higher dimensions. The bisection method will terminate when the relative error is <0.005%. The relative error is calculated as the absolute value of the proposed solution minus the actual solution divided by the absolute value of the actual solution, i.e. $(|f_{proposed}-f_{actual}| \div |f_{actual}|)$.

The parameters of the GA are set at 512 for population size, 10 for the number of generations, 0.80 for the crossover rate and 0.001 for the mutation rate. The GA was run 30 times and the maximum function value is reported. In the next session we used GA that parent population generated by different random number generator (for example: Halton, Sobol, Faure, Niederreiter, R250 and also proposed our CRNGs) and then discuss about some optimization problems.

PERFORMANCE EVALUATION'S CRNGS

Unconstrained global optimization problems (Eq. 4) have the form:

$$\max f(x)\ x \in [a,b] \tag{4}$$

Where:

f (x) = The objective function
a and b = The endpoints of the search interval for each variable

Test function 1: A unimodal one variable problem. The goal is to find x in the closed interval from -3 to 3 that maximizes:

$$f(x) = -x^4+12x^3 15x^2-56x+60$$

Figure 4 shows the graph of this function over the domain of x. To obtain the precision requirement of six places after the decimal point, x requires 23 bits:

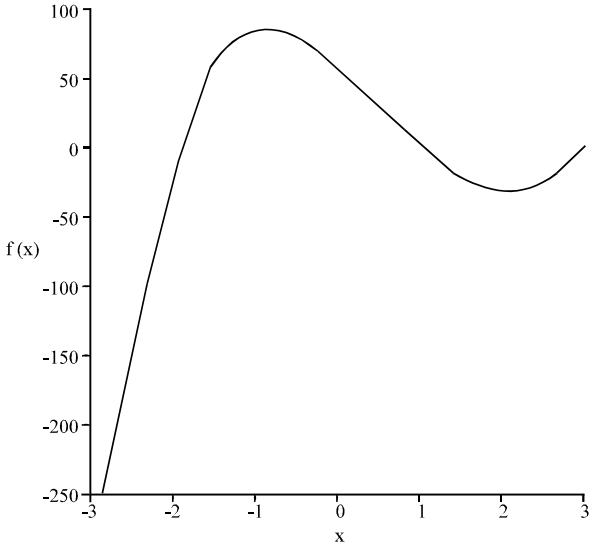

Fig. 4: Graph of f (x) over xϵ [-3, 3]

Table 1: GA results for f (x) over xϵ [-3, 3]

Ranking	Relative error (%)	F (x)	x	RNG
3	0.000006	88.891563	-0.869857	Halton (dim = 2)
5	0.000510	88.891115	-0.867187	Faure (dim = 2)
5	0.000510	88.891115	-0.867187	Sobol
2	0.000002	88.891566	-0.869992	Niederreiter
4	0.000009	88.891560	-0.870562	R250
1	0.0000001	88.891568	-0.870173	Reverse logistic
1	0.0000001	88.891568	-0.870173	Logistic
1	0.0000001	88.891568	-0.870173	Parabolic

Table 2: GA results for g (x) over xϵ [-1, 2]

Ranking	Relative error (%)	F (x)	x	RNG
2	0.0016	2.850274	1.850548	Halton (dim = 2)
1	0.0015	2.850272	1.850586	Faure (dim = 2)
1	0.0015	2.850272	1.850586	Sobol
2	0.0016	2.850274	1.850547	Niederreiter
2	0.0016	2.850274	1.850547	R250
1	0.0015	2.850271	1.850605	Reverse logistic
2	0.0016	2.850274	1.85054	Logistic
3	10.8186	2.541871	1.662185	Parabolic

(2^{22} = 4194,304<6×1000,000 = 6000,000<8388,608 = 2^{23})

The known solution is found at x = -0.8702 resulting in a function value of 88.891568. The initial results using the different RNGs as the parent population are shown in Table 1.

Test function 2: A multimodal one variable problem with one global maximum. Using a test problem provided by Goldberg (1989), the goal is to find x in the closed interval from -1 to 2 that maximizes g (x) = 1.0+x sin (10πx). Figure 5 shows the graph of this function over the domain of x. The known solution is found at x = 1.8508 resulting in a function value of 2.850227. The results for this test problem using the different RNGs as the parent population are shown in Table 2.

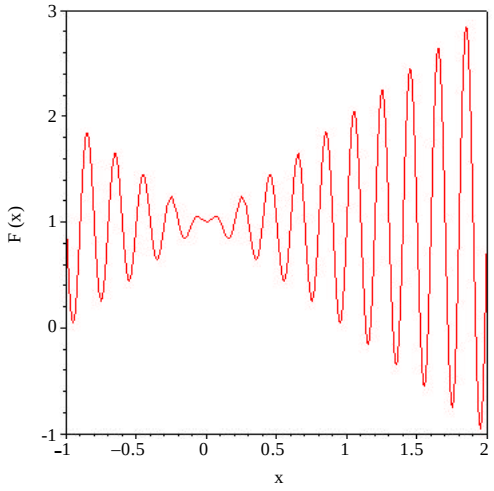

Fig. 5: Graph of g (x) over xϵ [-1, 2]

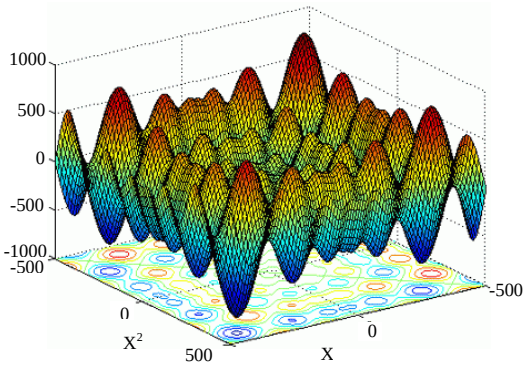

Fig. 6: Graph of h (x_1, x_2) over x_1, x_2 ϵ [-500, 500]

Test function 3: A multimodal two variable problem with one global maximum. Figure 6 shows the graph of this function. his two variable test problem is known as Schwefel's function and was also obtained at The GA Playground. The function is characterized by a global maximum that is geometrically distant from the next best local maxima. Therefore, search algorithms are prone to convergence in the wrong direction. The function is defined as:

$$h(x_1,x_2) = x_1 \sin(\sqrt{|x_1|}) + x_2 \sin(\sqrt{|x_2|})$$

over the closed interval of (-500, 500) for both x_1 and x_2. To obtain the precision requirement of six places after the decimal point, x_1 and x_2 each require 30 bits:

Table 3: Initial GA results forh (x_1, x_2) over $x_1,x_{2\epsilon}$ [-500, 500]

Ranking	Relative error (%)	F (x)	x_2	x_1	RNG
4	0.003750	837.934351	420.668662	420.569975	Halton (dim=2)
7	28.449170	599.571422	500.000000	420.898438	Faure (dim = 2)
7	28.449170	599.571422	500.000000	420.898438	Sobol
2	0.000287	837.963366	421.096803	420.916932	Niederreiter
3	0.000878	837.958415	420.895100	420.738733	R250
1	0.000276	837.963480	420.835399	420.944746	Reverse logistic
6	0.009890	837.8828526	421.779098	420.951363	Logistic
5	0.002109	837.948103	421.067159	421.329774	Parabolic

(2^{29} = 536,870,912<1000×1000,000 = 1000,000,000<1073,741,824 = 2^{30})

Therefore, the parent population will contain strings of 30+30 = 60 bits. The known solution for this test problem is found at $x_1 = x_2 = 420.9687$, resulting in a function value of f (x_1, x_2) = 837.965775. Table 3 shows the results of the initial run for the different RNGs.

CONCLUSION

Which existing random number generator is the best under all circumstances is still an open problem. The answer will depend on numerous parameters, such as parent population because the target of such algorithms is getting the best answer of optimization problem without survey of all of the state space. So the more the parent population is more uniform, we have higher chances to get the overall maximum and get away from local maximum. So the genetic algorithm is a criterion to realize what initial population is more uniformity. Then we evaluated that the proposed chaotic random number generator has the best performance to get the overall maximum with less percent relative error.

REFERENCES

Andrecut, M., 1998. Logistic map as a random number generator. Int. J. Modern Phys. B, 12: 921-930.

Baker, R., 1998. Genetic algorithms in search and optimization. Financial Engineering News.

Banzhaf, W., P. Nordin, R.E. Keller and F.D. Francome, 1998. Genetic Programming an Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers, San Francisco, California.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edn., Addison-Wesley Publishing Company, New York, USA., ISBN: 0201157675, pp: 36-90.

Mitchell, M., 1997. An Introduction to Genetic Algorithms. MIT Press, Cambridge, Massachusetts.

Obitko, M. and P. Slavik, 1999. Visualization of genetic algorithms in a learning environment. Proceedings of the Spring Conference on Computer Raphics, Apr. 25-28, Budmerice, Slovakia, pp: 101-106.

Reese, A., 2009. Random number generators in genetic algorithms for unconstrained and constrained optimization. Nonlinear Analysis, 71: 679-692.

Sen, S.K. and T. Samanta and A. Reese, 2006. Quasi- versus Pseudo-random generators: Discrepancy, complexity and integration-error based comparison. Int. J. Innovative Comput., 2: 621-665.

Vose, M., 1999. The Simple Genetic Algorithm: Foundations and Theory. MIT Press, Cambridge, Massachusetts.

Wood, G.R., 1992. The bisection method in higher dimensions. Mathematical Programm., 55: 319-337.

Zbigniew, M., 1996. Genetic Algorithms + Data Structures = Evolution Programs. 3rd Edn., Springer, New York, pp: 387.