Proceedings of the Seventh Oklahoma Conference on Artificial Intelligence, pp. 200-206, November, 1993.

The Performance of a Genetic Algorithm on a Chaotic Objective Function *

Arthur L. Corcoran
corcoran@penguin.mcs.utulsa.edu

Roger L. Wainwright
rogerw@penguin.mcs.utulsa.edu

Department of Mathematical and Computer Sciences
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
(918) 631-2228
(918) 631-3077 Fax

1 INTRODUCTION

A *genetic algorithm* (GA) is a general, iterative search technique based on the concepts and mechanisms of natural evolution. GAs are especially well known for their ability to search a large problem domain in a relatively short period of time and find fairly good solutions. *Dynamical systems* are processes in motion or processes which change over time. Examples of dynamical systems can be found everywhere. They range from very simple to extremely complex systems. The motivation for studying dynamical systems is to be able to predict the future direction or state of such systems over time or space. However, under some circumstances even simple dynamical systems exhibit unpredictable, almost random behavior. Such behavior is called *chaos*. The *logistic equation* is the canonical example of a chaotic dynamical system. It is a simple model used to predict future populations of an organism over time.

This paper investigates the application of a genetic algorithm to a chaotic dynamical system: the logistic equation. The organization of the paper is as follows. Section 2 provides an introduction to genetic algorithms. Section 3 provides an introduction to dynamical systems including basic definitions. The logistic equation is introduced as an example of a chaotic dynamical system. Section 4 explores the application of a genetic algorithm to the logistic equation. Finally, Section 5 provides a summary and final conclusions.

*Research partially supported by OCAST Grant AR2-004 and Sun Microsystems, Inc.

2 GENETIC ALGORITHMS

A *genetic algorithm* is an iterative procedure which borrows the ideas of natural selection and 'survival of the fittest' from natural evolution. By simulating natural evolution, in this way, a GA can easily solve complex problems. Furthermore, by emulating biological selection and reproduction techniques, a GA can effectively search the problem domain in a general, representation-independent manner.

The genetic algorithm maintains a population or pool of candidate solutions for a given objective function. The candidate solutions represent an encoding of the problem into a form that is analogous to the chromosomes of biological systems. Each chromosome is made up of a string of genes (whose values are called alleles). The chromosome is typically represented in the GA as a string of bits. However, integers and floating point numbers can easily be used. Associated with each chromosome is a fitness value, which is found by evaluating the chromosome with the objective function. It is the fitness of a chromosome which determines its ability to survive and produce offspring.

Once an initial pool has been generated and all of its members have been evaluated, the genetic algorithm begins its emulation of the life cycle. The pool size remains constant throughout the GA. At each step in the iteration, chromosomes are probabilistically selected from the population for reproduction according to the principle of the 'survival of the fittest'. Offspring are generated through a process called crossover, which can be augmented by mutation. The offspring are then placed back in the pool, perhaps replacing other members of the pool. This process can be modeled using either a 'generational' [5, 6] or a 'steady-state' [7] genetic algorithm. The generational GA saves offspring in a temporary location until the end of a generation. At that time the offspring replace the entire current population. Conversely, the steady-state GA immediately places offspring back into the current population.

```
procedure GA
begin
  t = 0;
  initialize P(t)
  evaluate structures in P(t);
  while termination condition not satisfied do
  begin
    t = t + 1;
    P(t) = select from P(t-1)
    alter structures in P(t);
    evaluate structures in P(t);
  end
end.
```

Figure 1: Genetic algorithm

Figure 1 illustrates a genetic algorithm. At time zero ($t = 0$), the population ($P(t)$) is initialized and evaluated. While the termination condition is not satisfied, a portion of the population is selected, somehow altered, evaluated, and placed back into the population.

The genetic algorithm relys on genetic operators for selection, crossover, mutation, and replacement. The selection operators use the fitness values to select a portion of the population to be parents for the next generation. Parents are combined using the crossover and mutation operators to produce offspring. This process combines the fittest chromosomes and passes superior genes to the next generation, thus providing new points in the solution space. The replacement operators ensure that the 'least fit' or weakest chromosomes of the population are displaced by more fit chromosomes.

While the fundamental concepts of genetic algorithms are fairly simple and straightforward, there are numerous implementation variations and options to incorporate into a genetic algorithm. For example, there are numerous ways to parametrize a model and encode it into a finite length chromosome. There are numerous selection techniques for determining chromosomes for crossover. There are literally dozens of possible crossover operators that have been developed in recent years depending on the problem type and chromosome encoding scheme. There are also several techniques for introducing some random changes to a chromosome (i.e., mutation).

3 CHAOTIC DYNAMICAL SYSTEMS

Dynamics is a branch of mathematics which attempts to understand *dynamical systems*, that is, processes in motion or processes which change over time. Such processes are found in all branches of science. For example, the motion of a simple pendulum and the reactions between various chemicals are dynamical systems. Other examples include the motion of celestial bodies, ups and downs of the stock market, and weather conditions throughout the world.

The motivation for studying a dynamical system is in predicting the future behavior of the system. Some systems, such as the motion of a simple pendulum and the reactions between various chemicals, are easily predicted. Others, such as the direction of the stock market and weather forecasting, seem impossible to predict. For many problems, the difficulty in predicting the behavior can be attributed to the complexity of the problem. However, recently researchers have noted that even simple systems can exhibit unpredictable behavior, or *chaos*. Such unpredictable systems are *chaotic dynamical systems*. Researchers study simple chaotic systems hoping it will lead to a better understanding of more complex and unpredictable problems such as predicting the direction of the stock market and weather forecasting.

Only basic concepts of chaotic dynamical systems are discussed below. Devaney [3] and Gleick [4] are excellent sources for a more thorough investigation into dynamical systems and chaos. Devaney [2] is an excellent source for researchers with an in-depth knowledge of calculus.

Definitions

The basic operation of dynamical systems is *iteration*. Iteration is the repetition of a process. Iteration begins with some initial seed or input and proceeds with the output of one application of the process as the input to the next application of the process. While iteration may be applied to just about any process, in dynamics it is usually applied to a mathematical function. For example, consider the positive square root function, $F(x) = \sqrt{x}$, and the initial input $x_0 = 256$. Iteration yields the following:

$$\begin{aligned}
\sqrt{256} &= 16 \\
\sqrt{16} &= 4 \\
\sqrt{4} &= 2 \\
\sqrt{2} &= 1.414214\ldots \\
\sqrt{1.414214\ldots} &= 1.189207\ldots \\
\sqrt{1.189207\ldots} &= 1.090508\ldots \\
\sqrt{1.090508\ldots} &= 1.044274\ldots \\
&\vdots \\
\sqrt{1} &= 1
\end{aligned}$$

For convenience, $F^i(x)$ is defined as the ith application of $F(x)$. Thus,

$$\begin{array}{rcl} F^1(256) & = & 16 \\ F^2(256) & = & 4 \\ F^3(256) & = & 2 \\ & \vdots & \end{array}$$

The list of successive iterates of a point is called the *orbit* of that point. The *orbit analysis* of $F(x) = \sqrt{x}$ reveals for any positive x_0,

$$\lim_{n\to\infty} F^n(x_0) = 1.$$

Since $F(x)$ eventually reaches 1 for any positive x_0 and remains fixed at 1 thereafter, 1 is a *fixed point* of $F(x)$.

Not all functions are as simple as the square root function. For example, consider the square function, $S(x) = x^2$. $S(x)$ has a fixed point at 1, however, it also has orbits which tend to 0 and ∞:

$$\lim_{n\to\infty} S^n(x_0) = \begin{cases} 0 & \text{if } 0 < x_0 < 1 \\ 1 & \text{if } x_0 = 1 \\ \infty & \text{if } x_0 > 1 \end{cases}$$

In the case of the square root function, 1 is an *attracting* fixed point of $F(x)$. For the square function, 1 is a *repelling* fixed point of $S(x)$. Clearly, it is easier to find an attracting fixed point than it is to find a repelling fixed point.

Another type of orbit is the *periodic orbit*, or *cycle*. A periodic orbit eventually returns to where it began. That is, the orbit of x_0 is periodic if there exists an integer n such that $F^n(x_0) = x_0$. In this case, x_0 is a *periodic point* of period n. The smallest such n is the *prime period* of the orbit. For example, consider the reciprocal function, $R(x) = 1/x$. Both 1 and -1 are fixed points of $R(x)$. Any other initial point x_0 generates a cycle of period 2, oscillating between x_0 and $1/x_0$.

The Logistic Equation

Biologists often use mathematical models to accurately predict the growth or decline of populations of organisms in future generations or years. These models are often dynamical systems. One of the simplest of these dynamical systems is the *logistic equation*.

The logistic equation models an idealized population of organisms in a controlled environment free from external influence. The only variable is the percentage of some limiting population that is alive during each generation. The limiting population, denoted by L, is the absolute maximum population size possible. The percentage of L alive at generation n is denoted by P_n. For example, if $P_n = .5$, then the population is exactly $L/2$ at generation n. Clearly, $0 \leq P_n \leq 1$. The logistic equation is

$$P_{n+1} = cP_n(1 - P_n)$$

where c is a constant determined by the amount of food available, temperature, etc. Given an initial population P_0, constant c, and the logistic equation, future populations can be determined. Note this is simply the iteration of the function $F(x) = cx(1-x)$. Also, for a given P_0, $P_n = F^n(P_0)$.

Orbit analysis of the logistic function for typical x_0 yields interesting results:

$c \leq 1$	all orbits tend to 0
$c = 1.5$	all orbits tend to $\frac{1}{3}$
$c = 2$	all orbits tend to $\frac{1}{2}$
$c = 3$	all orbits tend to $\frac{2}{3}$
$c = 3.2$	all orbits approach a period 2 cycle
$c = 3.5$	all orbits approach a period 4 cycle
$c = 3.55$	all orbits approach a period 8 cycle
$c = 3.83$	all orbits approach a period 3 cycle
$c \geq 4$	all orbits are **chaotic**

Note the above results are for typical x_0. There exist some x_0 which do not obey the above results.

4 EXPERIMENTAL RESULTS

The purpose of this research is to explore the behavior of a genetic algorithm on a chaotic dynamical system. The logistic equation is the canonical example used to illustrate chaotic dynamical systems. Consequently, it is a logical choice for our research.

We used the genetic algorithm package, LibGA [1], to implement a genetic algorithm for the logistic equation. The chromosome was composed of a single double precision floating point value. A pool of such chromosomes was generated at random with a pool size of 100. The allele values were limited to the range [0..1]. Thus, the pool represents a random set of x_0 to seed the logistic equation. Recall the functional form of the logistic equation is as follows:

$$F(x) = cx(1-x).$$

Our genetic algorithm used a generational model with crossover and mutation disabled. Only reproduction was used. No elitism was used. The goal of the GA was to find an x_0 to maximize the objective function. That is, to find an x_0 which results in the largest value in the final population, $F^n(x_0)$. Ideally, the GA should converge to 1.

Our implementation of the objective function had several variants. We used c values in the logistic equation which were both chaotic ($c = 4$) and non chaotic ($c = 2$). The fitness of the chromosome was based on a number, n, of iterations of the logistic function. The number of iterations was computed both **fixed** at $n = 50$ and **random** such that $10 \leq n \leq 100$. The logistic equation itself was applied using both **static** and **dynamic** methods. The static method computed the fitness by iterating the logistic equation n times. The dynamic method computed the fitness by iterating the logistic equation only once. The resulting value is then placed back into the chromosome. Thus, the dynamic method forces the GA to optimize a truly dynamic system. The static method was run with the fixed and random computation of n. The methods we ran are listed below:

- **Static/Fixed**: The iteration number is fixed, $n = 50$. The value of 50 was chosen arbitrarily. For a given x_0, the fitness returned by the objective function is

  $$\text{fitness}(x_0) \leftarrow F^{50}(x_0).$$

  Thus, the GA will maximize the 50th orbital of x_0.

- **Static/Random**: In this case the iteration number is computed at random: $10 \leq n \leq 100$. Each time a given x_0 is evaluated, the objective function will give a different result, based on

  $$\text{fitness}(x_0) \leftarrow F^n(x_0) = x_n, n \in [10..100].$$

  Thus, the GA will maximize the average of the first 100 orbital values of x_0.

- **Dynamic**: The objective function computes a single iteration

  $$\text{fitness}(x_0) \leftarrow F(x_0)$$

  and then sets the chromosome equal to the fitness, that is,

  $$x_0 \leftarrow F(x_0).$$

  Thus, the fitness is computed dynamically so that at generation n the fitness is $F^n(x_0)$.

Table 1 summarizes the results of the static/fixed method. 'Historical Best' refers to the best chromosome ever encountered during the running of the GA. 'Final Best' refers to the best chromosome found in the final pool. 'Seed' refers to the seed for the random number generator. The seed values were selected at random. The '# Gen' refers to the number of generations required to converge. In our case, convergence means that the entire population has the same fitness value. For $c = 2$ (non chaotic system) Table 1 shows that for all seeds the GA converged to 0.5 in the first generation. This indicates that by the fixed orbital ($n = 50$) all of the values in the initial pool are 0.5. This is reflected by the seemingly random set of values for the historical and final best chromosomes. For $c = 4$ (chaotic system) Table 1 shows quite different results. The different seeds required different numbers of generations to converge. In both the historical best and the final best, the values have converged nearly to the optimum, 1. The average of the historical best fitnesses is 0.999585. The average of the final best fitnesses is 0.993836. Note the average fitness values in the table refer only to the chaotic results.

Table 2 summarizes the results of the static/random method. As in the case of the static/fixed method, the non chaotic system converged to 0.5 in the first generation for all seeds. The chaotic system had more interesting results. Since the fitness for any x_0 depends on a randomly chosen n, the fitness may change from one generation to the next. Any given x_0 may be a top performer in one generation, but have a much lower fitness in another generation. Because of this unpredictability, the GA never converged. This was expected. The values in the table were obtained by limiting the GA to 500 generations. Despite the chaos, the GA was still able to produce excellent results. The average historical best was 0.999991 and the average final best was 0.997252. These results are even better than for the static/fixed method, which was a pleasant surprise.

Table 3 summarizes the results of the dynamic method. The dynamic nature of the fitness function allowed the non chaotic system to iterate for a few generations before converging. As expected, this model converged to 0.5 for all seeds. In the static methods, the distribution of the values for the historical and final best chromosomes was virtually random. In the dynamic method, the values for the final best are similarly distributed. However, for the historical best, the values seem to be clustered around 0.5. These values are members of the initial pool which converged to the optimum (0.5) in the least number of iterations. That is, their orbits are very short. This is an interesting result. The final pool values show that by the time the GA converged, short orbits became less of an advantage. The chaotic system also had interesting results. This time, the GA was able to converge quite rapidly. The average historical best was nearly optimal at 0.999974. Curiously, the average final best was a dismal 0.605444. This means that at one time the GA found a good performer in the historical best, but the GA was a failure at predicting future high performers.

Table 4 shows the orbit averages for each of the three methods for the chaotic model with seed of 1. 'Historical' refers to the average of the orbits using the historical best. 'Final' refers to the average of the orbits using the final best. For the static/fixed method, the historical best had a slightly better average than the final best. Most likely, the fitness of the historical best was not large enough in proportion to the other fitnesses to ensure its survival to the final population. Nonetheless, the GA was very successful in maintaining high fitnesses using the static/fixed method. For the static/random method, the final best had a much better average than the historical best. The historical best had an orbit that at one generation was the best. However, as time went on the chromosomes with higher orbit averages tended to dominate the pool. Thus, the GA was very successful in maintaining high fitnesses with this method as well. In addition, the GA was extremely successful in adapting to the randomness of this method. For the dynamic method, the final best also had a much better average than the historical best. Recall, however, that the average final best fitness was a dismal 0.605444. While preserving chromosomes with high average orbital fitness proved quite successful for the GA with the static/random method, it proved to be a failure with the dynamic method. The dynamic method was able to fool the GA into promoting those chromosomes with high initial orbit averages that failed to maintain those averages over time.

5 CONCLUSIONS

In this paper we have provided an introduction to genetic algorithms. We also introduced dynamical systems, processes in motion or processes which change over time. We introduced the logistic equation, a chaotic dynamical system. Our goal was to explore the behavior of a genetic algorithm on the logistic equation. We implemented three different methods: static/fixed, static/random, and dynamic. Our results showed the GA performed well on the static/fixed and static/random methods. The GA was especially successful in adapting to the randomness of the static/random method. Unfortunately, the GA had poor results on the dynamic method. Future work will include studying ways to improve the GAs performance on the dynamic method and exploring whether the GA itself may exhibit chaotic behavior under certain circumstances.

ACKNOWLEDGEMENTS

This research has been supported by OCAST Grant AR2-004. The authors also wish to acknowledge the support of Sun Microsystems, Inc.

REFERENCES

[1] A. L. Corcoran and R. L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In Ed Deaton, K. M. George, Hal Berghel, and George Hedrick, editors, *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–118, New York, 1993. ACM Press.

[2] Robert L. Devaney. *An Introduction to Chaotic Dynamical Systems.* Addison-Wesley, Menlo Park, California, 1989.

[3] Robert L. Devaney. *Chaos, Fractals, and Dynamics.* Addison-Wesley, Menlo Park, California, 1990.

[4] J. Gleick. *Chaos: Making a New Science.* Viking, New York, 1987.

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[6] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[7] Darrell Whitley and J. Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, Denver, Colorado, 1988.

c	Seed	# Gen	Historical Best		Final Best	
			Value	Fitness	Value	Fitness
2	1	1	0.968071	0.500000	0.378829	0.500000
	7494	1	0.837649	0.500000	0.592731	0.500000
	19469	1	0.460122	0.500000	0.664277	0.500000
	2089	1	0.946140	0.500000	0.631332	0.500000
	17	1	0.627609	0.500000	0.919280	0.500000
	8001	1	0.736738	0.500000	0.577852	0.500000
	53941	1	0.433308	0.500000	0.158668	0.500000
	350	1	0.354229	0.500000	0.059068	0.500000
	5129	1	0.349787	0.500000	0.470580	0.500000
	11111	1	0.934307	0.500000	0.717895	0.500000
4	1	79	0.587424	0.999833	0.511382	0.984692
	7494	79	0.746603	1.000000	0.710099	0.981542
	19469	196	0.370569	0.999928	0.611467	0.992142
	2089	197	0.128294	0.999177	0.690101	0.996262
	17	319	0.330434	0.998981	0.111458	0.994722
	8001	77	0.851055	0.999996	0.462653	0.997615
	53941	81	0.680238	0.998399	0.648961	0.997131
	350	221	0.159987	0.999991	0.449100	0.997957
	5129	78	0.338930	0.999576	0.963082	0.996734
	11111	136	0.549437	0.999967	0.683301	0.999567
	Average Fitnesses			0.999585		0.993836

Table 1: Static/Fixed Results

c	Seed	# Gen	Historical Best		Final Best	
			Value	Fitness	Value	Fitness
2	1	1	0.968071	0.500000	0.120798	0.500000
	7494	1	0.837649	0.500000	0.140289	0.500000
	19469	1	0.460122	0.500000	0.446268	0.500000
	2089	1	0.946140	0.500000	0.835305	0.500000
	17	1	0.627609	0.500000	0.786189	0.500000
	8001	1	0.736738	0.500000	0.232552	0.500000
	53941	1	0.433308	0.500000	0.336634	0.500000
	350	1	0.354229	0.500000	0.852365	0.500000
	5129	1	0.349787	0.500000	0.743547	0.500000
	11111	1	0.934307	0.500000	0.762643	0.500000
4	1	500	0.747450	1.000000	0.871333	0.997986
	7494	500	0.795218	1.000000	0.967209	0.999311
	19469	500	0.815944	0.999995	0.972782	0.998878
	2089	500	0.600598	0.999999	0.968311	0.999823
	17	500	0.011908	1.000000	0.247763	0.999735
	8001	500	0.134800	0.999993	0.504729	0.999272
	53941	500	0.694956	0.999995	0.005622	0.997860
	350	500	0.585227	0.999990	0.941532	0.997154
	5129	500	0.976955	0.999942	0.728120	0.983698
	11111	500	0.506702	0.999996	0.755679	0.998800
	Average Fitnesses			0.999991		0.997252

Table 2: Static/Random Results

c	Seed	# Gen	Historical Best		Final Best	
			Value	Fitness	Value	Fitness
2	1	4	0.456381	0.500000	0.408019	0.500000
	7494	5	0.526577	0.500000	0.293868	0.500000
	19469	4	0.488345	0.500000	0.635880	0.500000
	2089	7	0.520674	0.500000	0.548777	0.500000
	17	5	0.474033	0.500000	0.594659	0.500000
	8001	7	0.497689	0.500000	0.266313	0.500000
	53941	8	0.544323	0.500000	0.347056	0.500000
	350	6	0.500256	0.500000	0.813362	0.500000
	5129	8	0.502224	0.500000	0.464418	0.500000
	11111	4	0.538590	0.500000	0.431158	0.500000
4	1	14	0.747450	1.000000	0.635435	0.999807
	7494	28	0.597586	1.000000	0.341447	0.229156
	19469	22	0.251185	0.999990	0.635880	0.996731
	2089	22	0.222274	0.999999	0.727774	0.556116
	17	39	0.652859	0.999780	0.073699	0.323634
	8001	26	0.501443	0.999992	0.278507	0.999136
	53941	22	0.777716	0.999998	0.234059	0.992341
	350	17	0.500256	1.000000	0.801048	0.559598
	5129	30	0.502224	0.999980	0.346517	0.273588
	11111	32	0.886497	1.000000	0.800092	0.124335
	Average Fitnesses			0.999974		0.605444

Table 3: Dynamic Results

Method	Historical	Final
Static/Fixed	0.5980832	0.5435709
Static/Random	0.4044938	0.5637647
Dynamic	0.4044938	0.4939187

Table 4: Orbit Averages for Seed = 1