TECHNOLOGY, Volume 7, (2004), Issue 1, 105-114

TECHNOLOGY

GENETIC ALGORITHMS: AN OPTIMIZATION TECHNIQUE

Hamit SARUHAN
Abant Izzet Baysal University, Technical Education Faculty, 81620, Düzce, Turkey

ABSTRACT

The main purpose of this paper is to present genetic algorithms as a viable alternative optimization technique and introduce their elements. Also the efficiency and ease of application of this technique are demonstrated by employing an illustrative example.

Key Words: Genetic algorithms, Optimization

GENETİK ALGORİTMALAR: BİR OPTİMİZASYON TEKNİĞİ

ÖZET

Bu makalenin asıl amacı genetik algoritmaların diğer optimizasyon tekniklerine alternatif bir metod olduğunu ortaya koymaktır. Bu çalışmayla; genetik algoritma elemanlarının tanıtımı, ayrıca kullanımlarının kolaylığı ve uygunluğu bir örnek üzerinde gösterilmiştir.

Anahtar kelimeler : Genetik algoritmalar, Optimizasyon

1. INTRODUCTION

The development of faster computer has given chance for more robust and efficient optimization methods. One of these robust methods is genetic algorithm, which has gained recognition as a general problem solving technique in many applications such as mathematics, engineering, medicine, and political science [1]. Genetic algorithms have received a rapidly growing interest in the community of combinatorial optimization dealing with problems characterized by a finite number of feasible solutions. Genetic algorithms are guided random search techniques. They are parameter search procedures based on the idea of natural selection and genetics [1]. They use objective function information instead of derivatives as in gradient-based methods. Numerical search techniques are good at exploitation but not exploration of the parameter space. They focus on area around the current design point, using local gradient calculations to move to a better design. Since there is no exploration for all regions of parameter space, they can easily be trapped in local optima [2]. Genetic algorithms are a class of general purpose algorithm that can make a remarkable balance between exploration and exploitation of the search space [3].

2. THE STRUCTURE OF GENETIC ALGORITHMS

Genetic algorithms were first proposed by Holland [4] and extended further by De Jong [5] and Goldberg [1]. Goldberg and De Jong have made significant advances in this field. Holland's genetic algorithm is a method for moving from one population of chromosomes to a new population by using a kind of natural selection together with genetic-inspired operators of crossover, mutation, and inversion [6].

Genetic algorithms are efficient search techniques involving a structured yet randomized information exchange resulting in a survival of the fittest among a population of string structure [7]. Genetic algorithms maintain a population of encoded solutions, and guide the population towards the optimum solution [1].

Thus, they search the space of possible individuals (strings) and seek to find high-fitness strings. Viewing the genetic algorithms as optimization techniques, they belong to the class of zero-order optimization methods [8] and [9]. The description of a simple genetic algorithm is outlined in Figure 1.

An initial population is chosen randomly in the beginning and fitness of initial population members is evaluated. Then an iterative process starts until the termination criteria have been satisfied. After the evaluation of each individual fitness in the population, the genetic operators, selection, crossover and mutation, are applied to produce a new generation. Other genetic operators are applied as needed. The newly created individuals replace existing generation and re-evaluation is started for fitness of new individuals. The loop is repeated until an acceptable solution is found. Genetic algorithms differ from traditional searchtechniques in the following ways [1]:

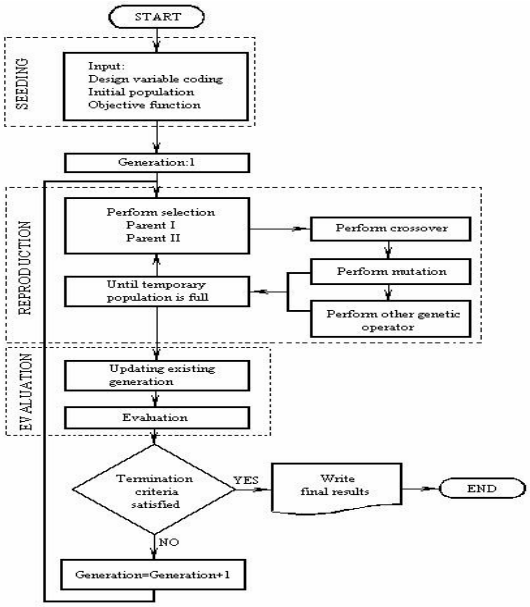

Figure 1 Flow chart for a simple genetic algorithms.

-Genetic algorithms work with a coding of design variables and not the design variables themselves.
-Genetic algorithms use objective function or fitness function information. No derivatives are necessary as in more traditional optimization methods.
-Genetic algorithms search from a set of points not a single point.
-Genetic algorithms gather information from current search points and direct them to subsequent search.
-Genetic algorithms can be used with discrete, integer, continuous, or a mix of these three design variables. The essential aspects of genetic algorithms are discussed in the following.

2.1 Coding

A genetic algorithm's data structure consists of one or more "chromosomes". A chromosome is typically a string of bits. Binary coding is generally used, although other coding schemes have been used such as floating point coding. For more information about other coding possibilities, interested reader may refer to [14]. The success of a genetic algorithm depends on the design of suitable representation (coding) of the problem. This includes the design of representation of chromosome and fitness function, which determines how well a program is able to solve the problem. Encoding of the design variables as chromosome, binary digits, is developed for this study. The number of digits in the binary string, l, is estimated from the following relationship [10]:

$$2^l \geq [(X^{upper} - X^{lower})/\varepsilon] + 1 \qquad (1)$$

where l is the string length and X^{lower} and X^{upper} are the lower and upper bounds of variable, X. The physical value of design variable, X, can be computed from the following relationship [11]:

$$X = X^{lower} + \left[(X^{upper} - X^{lower}) / 2^{l} - 1 \right] d \qquad (2)$$

where d represents the decimal value of string for the design variable which is obtained using base-2 form.

2.2 Fitness Function

One of the most crucial aspects of genetic algorithm is the fitness function. The fitness function provides a measure of performance of an individual, which is used to bias the selection process in favor of the most fit members of the current population. Therefore, the fittest members of the population should have the highest fitness, while the weakest members should exhibit relatively low fitness. If the objective is to maximize a function, $F(X)$, the fitness value may be calculated directly from the objective function, Eq. (3). However, when faced with minimization of a function a conversion to fitness function form, Eq. (4), is required. In case of constrained optimization problems, fitness function should be transformed into an unconstrained optimization problem by penalizing the objective function value to ensure that the solutions meet any imposed constrained.

$$Fitness_{Objective} = F(X) \quad \text{for maximization} \qquad (3)$$

$$Fitness_{Objective} = F - F(X) \quad \text{for minimization} \qquad (4)$$

Where F is a positive number which has to be large enough to exclude negative fitness values [1].

2.3 Population

Rather than starting from a single point solution within the search space as for traditional methods, genetic algorithms are initialized with a population of solutions. For the successful application of a genetic algorithm, there must always be sufficient diversity in the population. Once the chromosomes in the population become similar to each other, no further evolution is possible. This phenomenon is called *premature convergence*. As the population size increases the genetic algorithm has a better chance of finding the best solution, but the computation cost also increases as a function of the population size [12]. A guideline for an appropriate population size is suggested by Goldberg [13]. The guideline for optimal population size depends on the individual chromosome length, which is valid up to 60 expressed as follows:

$$population\ size = 1.65 * 2^{0.21 * l} \qquad (5)$$

As an example, for a string length of 18 bits, an optimal population size of 22 may be used. For this study, a randomly selected set of 26 strings is used as the starting population shown in Table 1.

Table 1 Randomly chosen initial population.

Individual Number	Randomized Binary String																	
1	0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	0	1
2	1	0	0	0	0	1	1	0	1	1	1	0	0	0	0	1	1	0
3	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	1	0	1
. . . . .																		
25	0	0	0	1	1	1	0	1	1	0	0	1	1	0	0	0	0	0
26	1	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	0	1

2.4 Genetic Algorithm Operators

The basic types of operators that are used in simple genetic algorithms include selection, crossover, and mutation. More complicated, specialized, and advanced operators include niching (fitness sharing) and reordering (inversion) [19]. Genetic operators are employed to recombine highly fit individuals and search the solution space for a better and better solution. These basic operators will be described in detail. More specialized and advanced operators including niching and reordering will also be described briefly in this section.

2.4.1 Selection

Genetic algorithms use a selection operator to select individuals from the population to insert into a mating pool. Individuals from the mating pool are used by a selection operator to generate new offspring, with the resulting offspring forming the basis of the next generation of solution. There are many different types of selection operators such as the *Roulette-Wheel selection* and *tournament selection*. The *Roulette-Wheel selection*, [1], chooses individuals through n, population size, simulated spins of roulette wheel. These solutions chosen randomly each with a probability proportionate to its relative fitness in the population. For a particular member, this probability of selection is calculated as the ratio of that member's fitness to the sum of the fitness values for entire population. For the chromosome i with fitness f_i, its selection probability P_{select_i} is determined by Goldberg [1]:

$$P_{select_i} = f_i \Big/ \sum_{i=1}^{pop.size} f_i \qquad (6)$$

Each spinning, a single chromosome for the new population is selected. When selecting the 26 strings that will be placed in the mating pool, the wheel is spun 26 times, with the results indicating the string to be placed in the pool. Another selection method is *tournament selection.* In tournament selection, the most fit of each pair is selected to mate. Then the crossover operator takes place.

A specialized selection mechanism that can be added to a genetic algorithm is *elitism*, which causes the most fit individual in a given generation to proceed unchanged into the following generation. This has the effect of guaranteeing a monotonically increasing maximum fitness for the population. It also guarantees the genetic algorithm will ultimately converge to the appropriate solution [6].

2.4.2 Crossover

The crossover operator is the primary source of new candidate solutions in a genetic algorithm. It uses the mating pool as parents of the next generation. The main distinguishing feature of a genetic algorithm is the use of crossover, which consists of choosing randomly a pair of individuals among those selected previously and swap sub-strings between them with fixed probability. This operator provides the search mechanism that efficiently guides the evolution through the solution space towards the optimum. There are several popular crossover methods, including single-point crossover, multi-point crossover, and uniform crossover; for more information about these crossover operators reader is referred to Goldberg [1]. Single-crossover is the simplest form, which originally was used in genetic algorithms. Single-point crossover cuts two chromosomes in one point and splices the two halves to create new ones. In Figure 2, the strings, Parent I and Parent II, are selected for crossover and the genetic algorithm decides to mate them. The crossover point has been chosen at position 10 (10-18). The parent exchanges the sub-strings, which occurs around crossing point that is selected randomly.

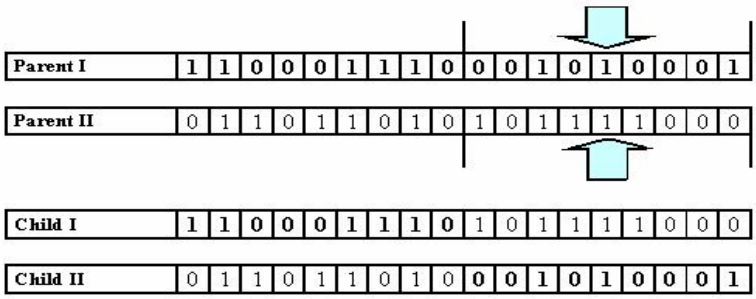

Figure 2 Single-point crossover operator.

The newly created strings are Child I and Child II. The cycle starts again with selection. This iterative process continues until specified criteria are met. In single-point crossover, the head and the tail of one chromosome cannot be passed together to the offspring. If both the head and the tail of a chromosome contain good genetic information, none of the offspring obtained directly with single-point crossover will share the two good features. Using a multi-point crossover avoids this drawback. The following is an example of multi-point crossover. In Figure 3, crossover points have been selected at position 10 (10-12) and 16 (16-18). The parent exchanges the sub-string and form two new members, Child I and Child II, of the population. Multi-point crossover is generally considered better than a single-point crossover. However, researchers considered other crossover operators with more cut points such as uniform crossover.

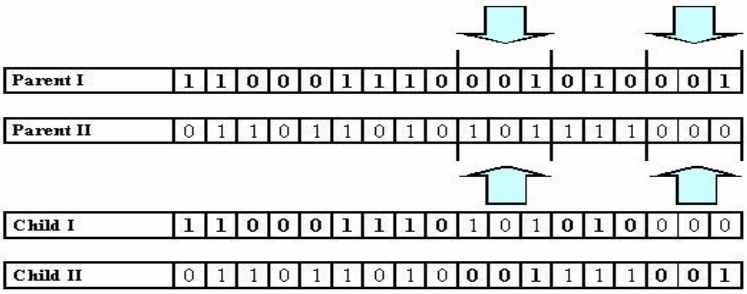

Figure 3 Multi-point crossover operator.

In uniform crossover, every bit of each parent string has a chance of being exchanged with corresponding bit of the other parent string. The procedure is to obtain any combination of two parent strings from the mating pool at random and generate new Child strings from these parent strings by performing bit-by-bit crossover chosen according to a randomly generated crossover mask [14]. Where there is a 1 in the crossover mask, the Child bit is copied from the first parent string, and where there is a 0 in the mask, the Child bit is copied from the second parent string. The second Child string uses the opposite rule to the previous one as shown in Figure 4. For each pair of parent strings a new crossover mask is randomly generated.

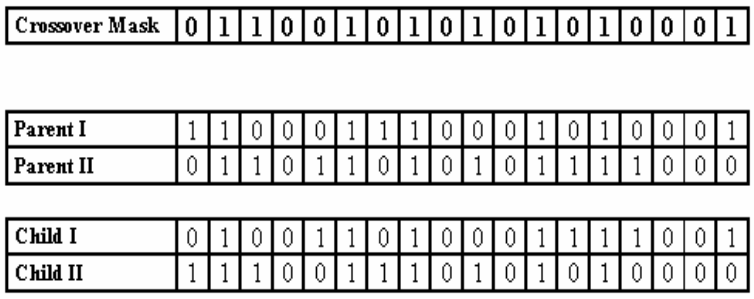

Figure 4 Uniform crossover.

2.4.3.Mutation

Mutation is the process of randomly altering a part of an individual to produce a new individual. If only selection and crossover are implemented, the population will become uniform in several generations. Mutation's primary role is to restore diversity that may be lost from the repeated application of selection and crossover and to prevent the genetic algorithm from *premature convergence* to a non-optimal solution. Genetic material can be lost, causing problems if, for instance, the entire population has 0's in a position which requires a 1 for optimality. In Figure 5, the genetic algorithm selected to mutate bit position 11 in the binary string. The digit, 0, is changed to 1 or vice versa. Typically the probability of mutation is very small. Too high a mutation rate can be detrimental to genetic searches. This will degenerate the genetic algorithm into a random walk through the string space. Studies have shown that mutation probabilities greater than 0.05 cause the search to become too random to be efficient [15, 16]. The mutation rate suggested by Bäck [17] is:

$$1/\,population\ size < P_{mutation} < 1/chromosome\ length \qquad (7)$$

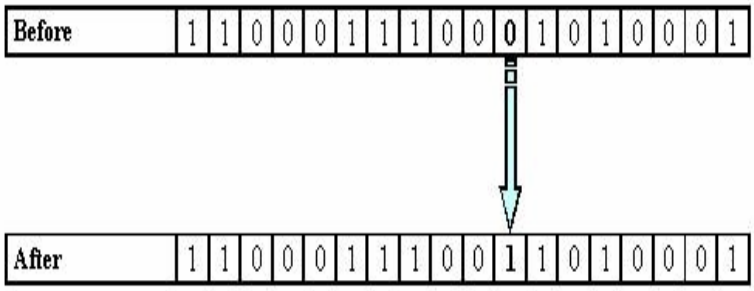

Figure 5 Mutation operator.

2.4.4 Other Operators

Though selection, crossover, and mutation are used in most genetic algorithm applications, many other operators are used by researchers in genetic algorithms. These include fitness sharing (niching) and reordering (inversion). For detailed discussion of these operators the reader can refer to [4], [1], [18], [19], and [20].

The main goal of *fitness sharing* is to distribute the population over a number of different peaks in the search space. Fitness sharing, [19], is an approach by which an individual's fitness is degraded by an amount related to the number of similar individuals in the population.

Inversion is a reordering operator inspired by a similar operator in a biological process, [21], which seeks a superior representation of the coded solutions through reorganization of coding. The inversion operator inverts the order of the bit values between two randomly selected points on the parent chromosome. The principal mechanism of inversion can be seen in Figure 6. This operator has not, in general, been found to be practical in genetic algorithms application as it adds to the computational complexity of the process.

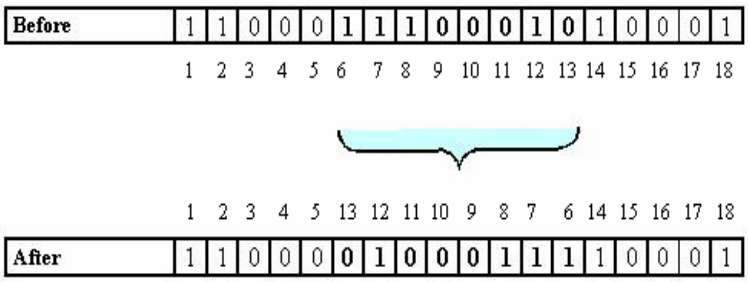

Figure 6 An inversion process.

2.5 Convergence

Convergence is the progression towards uniformity. A gene is said to have converged when 95% of the population share the same value [5]. Thus, most or all individuals in the population are identical or similar when population has converged. There are many different ways to determine when to stop running the genetic algorithm. One method is to stop after a preset number of generations or a time limit. Another is to stop after the genetic algorithm has converged.

2.6 An Illustrative Example

This section shows how the simple genetic algorithm optimization operates by employing an illustrative example. The problem presented is simply to find the maximum value of the following objective function over interval 0 - 25. Figure 7 shows the plots of the objective function versus design variable, X.

$$F(X) = e^{-|X-9|} \quad , \quad 0 \le X \le 25 \tag{8}$$

In order to make things easy, it will be assumed that the maximum value of the objective function is between the design variable value of 0 and 25 (the actual value is $X = 9$). To solve this problem, the design variable, X, is constructed with the binary digits (0, 1) representation. The number of digits in the binary string, l, is estimated from Eq. (1). The design variable is represented and discretized to a precision of ε (typically $\varepsilon = 0.001$). Based on the literature and trial cases, the parameters of the genetic algorithm for this study are chosen as follows: Chromosome length = 18, population size = 26, number of generation = 300, crossover probability = 0.5, mutation probability = 0.01. The illustrative example is solved using a computer code. The selection method used in the algorithm code is a tournament selection with a uniform crossover operator.

Figure 8 shows the distribution of fitness function values in the first, 100th, 200th, and the last generation. Figure 9 shows the plots of crossover operator with different probability, (0.3, 0.5, and 0.9), for fitness function value in each generation as optimization proceeds. From the plots, it can be seen that the crossover probability, 0.5, performs better than the 0.3 and 0.9. Mutation probability of 0.001, 0.01, and 0.1 were tested for genetic algorithm performance. Figure 10 shows the result of fitness function obtained by three different mutation probabilities. It can be seen from Figure 10 that the mutation probability of 0.01 gives preferable results compared to 0.1 and 0.001. It should be noted that if a mutation rate of 0.001 is selected, many good strings are never evaluated. In the other hand if mutation rate of 0.1 is selected much random perturbation is happened. This causes the losing of parent resemblance and is disastrous for obtaining the optimum point.

A complete generation history for the genetic algorithm run is given in Figure 11. Figure 11 shows the plot of the average and best fitness function values in each generation (population size of 26) as optimization proceeds. From the plot, the genetic algorithm optimizer makes rapid progress towards the maximum value of the function after generation 10 (about 260 evaluations of fitness function) and the fitness function has converged to a uniform solution with similar values through generations.

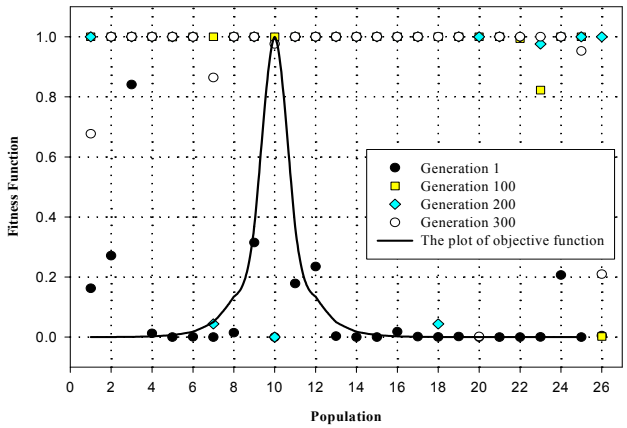

Figure 7 A plot of the objective function versus the design variable, $F(X) = e^{-|X-9|}$

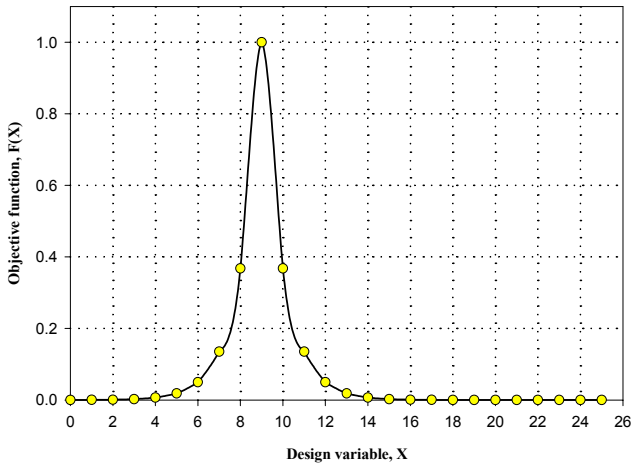

Figure 8 Convergence of the genetic algorithm through generations for objective function.

From the results, it can be seen that genetic algorithms are capable of finding the value of the design variable, $X = 9$, that maximize the objective function, $F(X) = 1$.

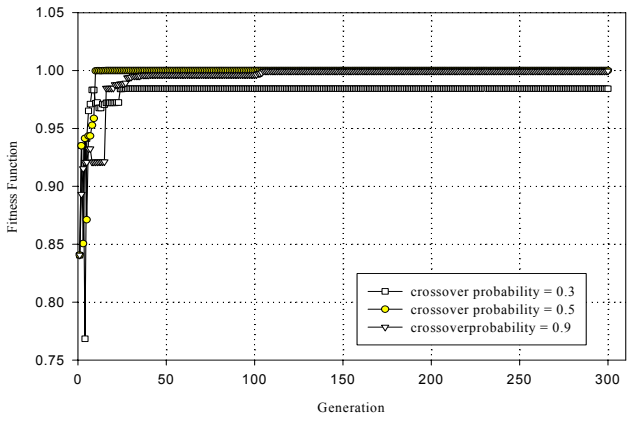

Figure 9 Effects of crossover probability on results for fitness function.

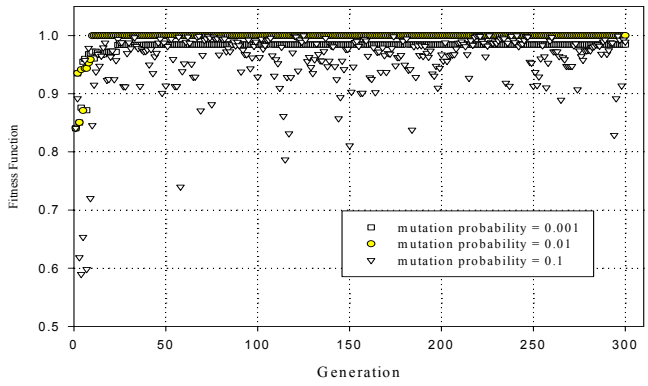

Figure 10 Effects of mutation probability on results for fitness function.

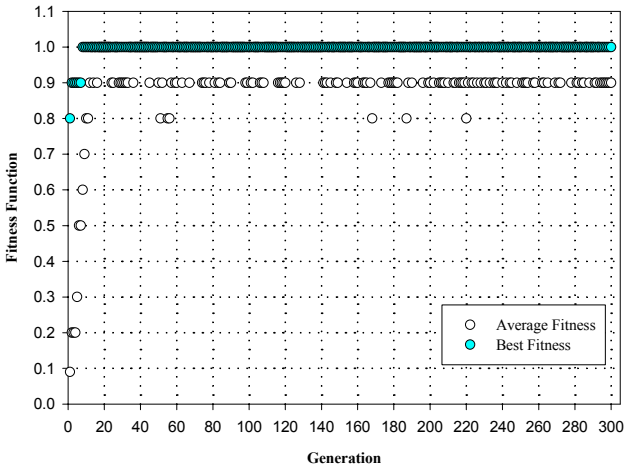

Figure 11 Convergence process of the genetic algorithms for average and best fitness function.

3. CONCLUSIONS

This paper gives an overview of the basics of genetic algorithms. All the different steps of the algorithm are outlined separately. It has been seen that genetic algorithms are very powerful tools. This algorithm is easy to produce and simple to understand. In the broadest sense, the genetic algorithm uses a couple of steps to solve problems. Although many genetic algorithms have been designed by different researchers and all of them are very different from each other, they all process in following pattern: initializing a starting population of random chromosome, evaluating the population to see which individuals will contribute to the next generation, generating new chromosomes by mating current chromosomes, using the new population of chromosome to replace the old population, evaluating the new chromosomes and inserting them into the population, and repeating this process (except the initializing a starting population) until some termination criterion is satisfied. The interested reader should consult introductory literature such as [1], [6]and [22], and for more detailed applications coverage of the topic such as [16], [23], and [24].

REFERENCES

1. Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, **Addison-Wesley,** Reading, 1989.
2. Davis, L., Handbook of Genetic Algorithms, **Van Nostrand Reinhold**, New York, NY, 1991.
3. Mitsuo, G., and Runwei, C., Genetic Algorithms and Engineering Design, **John Wiley Pub**., New York, 1997.
4. Holland, J.H., Adaptation in Natural and Artificial Systems, Ann Arbor, **University of Michigan Press,** MI, 1975.
5. DeJong K., The Analysis and Behavior of Class of Genetic Adaptive Systems, **Ph.D. Thesis, University of Michigan**, 1975.
6. Mitchell, M., An Introduction to Genetic Algorithms, **The MIT Press**, Massachusetts, 1997.
7. Mars, P., Chen, J.R., and Nambair, R., Learning Algorithms Theory and Applications in Signal Processing, **Control and Communications**, CRC Press, Inc., Florida, 1996.
8. Louis, S.J., Zhoo, F., and Zeng, X., Flaw Detection and Configuration with Genetic Algorithms, Evolutionary Algorithms in Engineering Applications, **Springer-Verlag**, 1997.
9. Dracopoulos, D.C., Evolutionary Learning Algorithms for Neural Adaptive Control, **Springer-Verlag,** London, 1997.
10. Lin, C.Y. and Hajela, P., Genetic Algorithms in Optimization Problems with Discrete and Integer Design Variables, **Engineering Optimization**, 19, 309-327, 1992.
11. Wu, S.J. and Chow, P.T., Genetic Algorithms for Nonlinear Mixed Discrete-Integer Optimization Problems via Meta-Genetic Parameter Optimization, **Engineering Optimization**, 24, 137-159, 1995.

12. Cantu-Paz, E., A Survey of Parallel Genetic Algorithms, IlliGAL Report No. 97003, **University of Illinois at Urbana-Champaign**, IL, 1997.
13. Goldberg, D.E., Optimal Initial Population Size for Binary Coded Genetic Algorithms, The Clearinghouse for Genetic Algorithms, **University of Alabama**, TCGA Rept. 85001, Tuscaloosa, 1985.
14. Beasly, D., Bull, D.R., and Martin, R.R., An Overview of Genetic Algorithms: Part2, Research Topics, **University Computing**, 15 (4), 170-181, 1993.
15. Grefenstette, J.J., Optimization of Control Parameters for Genetic Algorithms, **IEEE Trans**., On System, Man, and Cybernetics, Vol.16, No.1, pp.566-574, 1986.
16. Saruhan, H., Rouch, K.E., and Roso, C.A., Design Optimization of Tilting-Pad Journal Bearing Using a Genetic Algorithm Approach, **The 9th of International Symposium on Transport Phenomena and Dynamics of Rotating Machinery**, ISROMAC-9, Honolulu, Hawaii, 2002.
17. Bäck, T., Optimal Mutation Rates in Genetic Search, **Proceedings of the 5th International Conference on Genetic Algorithms**, Morgan Kaufmann, Los Angeles, 2-8, 1993.
18. Goldberg, D. E., and Bridges, C. L., An Analysis of A Reordering Operator on A GA-Hard Problem, **Biological Cybernetics**, 62, pp.379-405, 1990.
19. Goldberg, D.E., and Richardson, J., Genetic Algorithms with Sharing for Multimodal Function Optimization, In J. J. Grefenstette, ed., Genetic Algorithms and Their Applications, **Proceeding of the second International Conference on Genetic Algorithms**, Erlbaum, 1987.
20. Goldberg, D. E., and Wang, L., Adaptive Niching Via Co-evolutionary Sharing, IlliGAL Report No.97007, **Illinois Genetic Algorithms Laboratory**, Illinois, 1997.
21. Dawid, H., Adaptive Learning by Genetic Algorithms, **Springer-Verlag**, Berlin Heidelberg, 1996.
22. Michalewicz, Z., Genetic Algorithms+Data Structure=Evolution Programs, **Spriger-Verlag**, 1992.
23. Saruhan, H., and Uygur, İ., Design Optimization of Mechanical Systems Using Genetic Algorithms, **SAU Fen Bilimleri Enstitüsü** Dergisi, Vol.7-No.2, 2003.
24. Saruhan, H., Design Optimization of Rotor-Bearing System Using Genetic Algorithms, Dissertation, **University of Kentucky**, U.S.A, 2001