Optimization of Different Fitness Functions Using Reproduction, Crossover and Mutation of Genetic Algorithm With Binary Number Scheduling

J PRINCE JEROME CHRISTOPHER (jeromepsj1991@gmail.com)
Anna University, Chennai

K LINGADURAI
Anna University, Chennai

G SHANKAR
Periyar University

Research Article

Keywords: Crossover, Fitness Function, Genetic Algorithms, Initial Population, Mutation, Optimization and Reproduction

Posted Date: December 3rd, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1060904/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Abstract

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. In this paper, we investigate a novel approach to the binary coded testing process based on a genetic algorithm. This paper consists of two parts. The first part addresses the problem in the traditional way of using the decimal number system to define the fitness function to study the variations of counts and the variations of probability against the fitness functions. Second, the initial populations are defined using binary coded digits (genes). For the evaluation of the high fitness function values, three genetic operators, namely, reproduction, crossover and mutation, are randomly used. The results show the importance of the genetic operator, mutation, which yields the peak values for the fitness function based on binary coded numbers performed in a new way.

1. Introduction

Many problems in engineering, mathematical modeling, and numerical optimization can be solved efficiently using global optimization algorithms [1-7]. To address global optimization problems, researchers have proposed many optimization techniques, such as the Davidon-Fletcher-Powell algorithm, conjugate gradient method, and gradient-based algorithm [8-11]. These algorithms have achieved good results in solving engineering problems. However, many new optimization algorithms inspired by nature have been proposed in recent years, and one such algorithm is known as the “genetic algorithm”. It is a class of evolutionary algorithms that generate solutions to optimization problems using techniques inspired by natural evolution, such as reproduction, crossover and mutation.

The idea of a genetic algorithm comes from the natural rule [12]. Each organism lives with specific features coded in genes that are stored in chromosomes. They are allowed to transfer their genes to create new generations, and the process is called crossover. During reproduction, mutation possibly occurs, which involves the changing of one gene in a chromosome of offspring. The offspring that can fit to the environment better will be left. The genetic algorithm imitates this hereditary procedure in a mathematical way, which means that after random selection, crossover and mutation, the best result whose fitness is highest will be obtained. The aim of the title paper is to use the above operators only to find that operator that yields the highest fitness function.

2. Materials And Methods

2.1 Genetic Algorithm

2.1.1 Description of genetic algorithm

Genetic Algorithm for Test Data Generation: We will not describe the basic idea of a genetic algorithm here, since it is well known. We indicate only its adaptation as applied to the problem being solved. In this case, the chromosomes are the data variants received at the program input. In this paper, we assume that the data are numerical. The purpose of the algorithm is to select such a multitude of test data variants
that would provide the maximum coverage of the code, that is, to pass through as many program
operations as possible. This goal is realized through the formulation of the fitness function of a certain
kind, which expresses how well a particular data variant is adapted for mutual coexistence with other
individuals in a given generation. Let us introduce the important definitions as given below:

**Population:** a set of individuals of a specified size.

**Individuals** of a population are sets of task parameters coded in the form of chromosomes.

**Chromosomes:** ordered sequences of genes (chains or coded sequences)

**Gene:** is a single element in the genotype

**Genotype:** is a set of chromosomes of a given individual (can be a single chromosome)

**Initialization:** The initial population is formed randomly taking into account constraints on the variables’
values.

**Population assessment:** Each of the chromosomes is evaluated by a fitness function.

**Random Selection:** After the value of the fitness function for each of the chromosomes is calculated, the
best chromosomes are selected for crossover.

**Fitness Function:** This measures fitness of adaptation of a given individual to the environment. The
function allows the evaluation of the degree of fitness of particular individuals in the population, and
based on this degree of fitness, we select the individuals who fit best.

**New generation:** a newly created population of individuals (offspring generations)

**Reproduction:** To form a new solution, a couple of solutions from the previous generation are selected for
breeding from the pool selected previously. It creates a child solution by means of crossover and
mutation techniques. This new solution shares many of the characteristics of its parents. The process
continues until a new population of solutions of appropriate size is generated.

**Crossover:** In GA, crossover is a genetic operator used to vary the programming of a chromosome or
chromosomes from one generation to the next. It is analogous to reproduction and biological crossover,
upon which genetic algorithms are based. In the present research, a single crossover point was used for
each chromosome. In other words, each string of a proposed solution is split into two to be recombined
and create a member of the next generation.

**Mutation:** a genetic operator used to maintain genetic diversity from one generation of a population of
algorithm chromosomes to the next. It is analogous to biological mutation, in which new offspring are
born with certain changes that are not inherited from their parents. In other words, mutation randomly
alters the values of genes in a parent chromosome before the crossover operation is performed. Uniform random mutation was employed in the present algorithm.

2.2 Flow chart of solving steps

2.2.1 Flow chart diagram of genetic algorithm

The process steps of the genetic algorithm are sketched in Figure 1.

2.2.1.1 Initialization

An initialization population consists of random selection of a demanded number of individuals. Every individual is represented by chromosomes coded in a determined length of sequence of digits (binary number). In general, binary sequences to create the codes of chromosomes such as the string 10010 and 10101 are selected for crossover and the genetic algorithm limited to only four digits. The newly created strings in the title paper are extended to five digits such as 10011 and 10100. The crossover is performed until the new population is created. Then, the cycle starts again with selection. This iterative process continues until any user-specified criteria are met.

Parent 1 = 10010  Parent 2 = 10101
Child 1 = 10011    Child 2 = 10100

2.2.1.2 Calculation of fitness function

The fitness function is used to measure the adaptability of chromosomes and usually needs to be maximized. The form of the fitness function depends on the real problem, and a suitable transform seems to be necessary when we face a minimum optimization problem.

To maximize the fitness functions $f(x) = x^2, x^2 + x, x^3 + x, x \leq (0, 31)$

Here, $x$ represents a five digit number using integers. Therefore, we select a randomly generated preset of solution as [13].

Table 1. Selection of Initial Population

| String No. | Gene   | Value of $x$ |
|------------|--------|--------------|
| 1          | 1 0 1 0 0 | 20           |
| 2          | 0 1 0 0 0 | 8            |
| 3          | 1 0 1 0 1 | 21           |
| 4          | 0 1 1 0 0 | 12           |
We show one generational cycle done by hand. Therefore, the maximum function \( f(x) \) can be treated as the fitness function of this problem. After determining the fitness function of the solved problem, the evaluation of each chromosome is carried out according to the fitness functions. The higher the value of the fitness function is, the better the result of the solved maximization problem.

### 2.2.1.3 Reproduction

The reproduction operator allows individual strings to be copied for possible inclusion in the next generation. The chance that a string will be copied is based on the string fitness value calculated from a fitness function. For each generation, the reproduction operator chooses strings that are placed into a mating pool, which is used as the basis for creating the next generation.

### 2.2.1.4 Crossover

In the natural world, the genes of children are formed from both of their parents. To imitate this situation, we crossed the chromosomes of selected parents to generate offspring. Crossover just happens between two chromosomes. Therefore, the first stage of crossover is arranging the chromosomes of the selected parent’s population in pairs that are made randomly. After that, we should decide the crossover point. If the length of each chromosome is (i.e., for \( ch = 11000 \)) and the length of this chromosome is 5, we draw a random number, which belongs to interval \((1, 4)\). Then, the genes of the parents are exchanged from the gene on position. As a result, a pair of offspring is created. Example:

| A pair of parent       | A pair of offspring |
|------------------------|---------------------|
| Parent 10100           | crossover ->        |
|                        | ch1 10000           |
| Parent 01000           |                     |
|                        | ch2 11100           |

### 2.2.1.5 Mutation

Different from crossover, mutation occurs quite rarely in genetic algorithms. Therefore, we give a very small mutation rate, which is often in the interval \((0, 0.03)\). Every gene should be given a random mutation probability. If the drawn probability is less than or equal to random mutation probability, then this gene value should be changed to the opposite value (binary from 0 to 1 or from 1 to 0), which means that mutation occurs.

### 2.2.1.6 Obtaining new generation (Optimal Solution)

The parent population undergoes selection, crossover and mutation, creating offspring. It is then possible that the offspring are probably not feasible solutions at all. Therefore, we need to modify them or compel them to satisfy the conditions. After that, we obtain new feasible solutions, the so-called current population, with the same size as their parent’s population.

### 2.2.1.7 Stopping process
The process for stopping the genetic algorithm depends on the specific situation of the application of the genetic algorithm. If the stopping process is met, then the best result is taken. Otherwise, we have to go back to the initial step.

### 2.2.1.8 Best result of fitness function

The stop process should be checked when we obtain a new generation. If it is satisfied, then we stop the whole program and output the final result. Otherwise, we will return to the initial step until we achieve the stop process. The best result should be a chromosome that has the highest fitness function value.

### 3. Results And Discussion

The origin of evolutionary algorithms is an attempt to mimic some of the process taking place in natural evolution. Although the details of biological evolution are not completely understood, there are some points supported by strong experimental evidence [14-18]. Genetic algorithms are one of the most popular evolutionary algorithm techniques based on the concept of imitating the evolution of a species. In the case of GA, a population of chromosomes is generated using an intelligent method or a random selection method. Each of these individuals is encoded as a binary string that represents a possible candidate solution to the problem at hand. In each iteration, the survival strength of each candidate solution is measured by a fitness function. Afterwards, the evolutionary process is constrained by three genetic operators: selection, crossover, and mutation. Through the selection procedure, individuals are selected that enter into the crossover process.

The crossover operator alters two or more parents to create offspring, where a probabilistic crossover rate is usually used to generate offspring. The mutation operator produces one child from one parent by flipping slightly(s) of the parent. A probabilistic mutation rate is usually used to determine whether a particular change occurred within an individual [19-23]. There are some important characteristics of crossover and mutation operators that are not captured by the other. Błażej et al [24] have clearly pointed out that it has never been theoretically shown that mutation is in some sense less powerful than the crossover and vice versa. Mutation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behavior from components.

It is impossible for mutations to simultaneously achieve high levels of construction and survival. This would appear to be important since one without the other may not be extremely useful. High construction levels are accomplished at the expense of survival, while good survival is at the expense of construction [24-25]. In our study, we obtain high fitness function values for different mutation rates.

Table 2: Reproduction parameters when \( f(x) = x^2 \)
| String No | Initial Population | Value of x | Fitness Function | Probability | Actual count | Observed count |
|-----------|--------------------|------------|-----------------|-------------|--------------|----------------|
|          |                    |            | \( f(x) = x^2 \) |             |              |                |
| 1        | 1 0 1 0 0          | 20         | 400             | 0.38        | 1.52         | 1              |
| 2        | 0 1 0 0 0          | 8          | 64              | 0.06        | 0.24         | 0              |
| 3        | 1 0 1 0 1          | 21         | 441             | 0.42        | 1.68         | 2              |
| 4        | 0 1 1 0 0          | 12         | 144             | 0.14        | 0.56         | 1              |
| Sum      |                    |            | 1049            | 1           | 4            | 4              |
| Average  |                    |            | 262             | 25          | 1            | 1              |
| Maximum  |                    |            | 441             | 0.42        | 1.68         | 2              |

**Table. 3** Reproduction parameters when \( f(x) = x^2 + x \)

| String No | Initial Population | Value of x | Fitness Function | Probability | Actual count | Observed count |
|-----------|--------------------|------------|-----------------|-------------|--------------|----------------|
|          |                    |            | \( f(x) = x^2 + x \) |             |              |                |
| 1        | 1 0 1 0 0          | 20         | 420             | 0.38        | 1.51         | 1              |
| 2        | 0 1 0 0 0          | 8          | 72              | 0.06        | 0.26         | 0              |
| 3        | 1 0 1 0 1          | 21         | 462             | 0.42        | 1.67         | 2              |
| 4        | 0 1 1 0 0          | 12         | 156             | 0.14        | 0.56         | 1              |
| Sum      |                    |            | 1110            | 1           | 4            | 4              |
| Average  |                    |            | 278             | 25          | 1            | 1              |
| Maximum  |                    |            | 462             | 0.42        | 1.67         | 2              |

**Table. 4** Reproduction parameters when \( f(x) = x^3 \)
| String No | Initial Population | Value of x | Fitness Function | Probability | Actual count | Observed count |
|-----------|---------------------|------------|------------------|-------------|--------------|----------------|
| 1         | 1 0 1 0 0           | 20         | f(x) = x³       | 0.41        | 1.64         | 2              |
| 2         | 0 1 0 0 0           | 8          | 8000            | 0.03        | 0.11         | 0              |
| 3         | 1 0 1 0 1           | 21         | 9261            | 0.47        | 1.9          | 2              |
| 4         | 0 1 1 0 0           | 12         | 1728            | 0.09        | 0.35         | 0              |
| Sum       |                     |             | 19501           | 1           | 4            | 4              |
| Average   |                     |             | 4875            | 25          | 1            | 1              |
| Maximum   |                     |             | 9261            | 0.47        | 1.9          | 2              |

Table. 5 Reproduction parameters when f(x) = x³ + x

| String No | Initial Population | Value of x | Fitness Function | Probability | Actual count | Observed count |
|-----------|---------------------|------------|------------------|-------------|--------------|----------------|
| 1         | 1 0 1 0 0           | 20         | f(x) = x³ + x    | 0.41        | 1.64         | 2              |
| 2         | 0 1 0 0 0           | 8          | 8020             | 0.03        | 0.11         | 0              |
| 3         | 1 0 1 0 1           | 21         | 9282             | 0.47        | 1.9          | 2              |
| 4         | 0 1 1 0 0           | 12         | 1740             | 0.09        | 0.35         | 0              |
| Sum       |                     |             | 19562            | 1           | 4            | 4              |
| Average   |                     |             | 4891             | 25          | 1            | 1              |
| Maximum   |                     |             | 9282             | 0.47        | 1.9          | 2              |

Table. 6 Crossover parameters
| String No | Mating Pool | Crossover Point | Crossover After Crossover | Value of x | Fitness Function f(x) = x^2 | Fitness Function f(x) = x^2 + x | Fitness Function f(x) = x^3 | Fitness Function f(x) = x^3 + x |
|-----------|-------------|----------------|--------------------------|-----------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 1         | 1 0 1 0 0   | 2              | 1 0 0 0 0                | 16        | 256                         | 272                         | 4096                        | 4112                        |
| 2         | 0 1 0 0 0   | 2              | 1 1 1 0 0                | 28        | 784                         | 812                         | 21952                       | 21980                       |
| 3         | 1 0 1 0 1   | 4              | 1 0 1 0 0                | 20        | 400                         | 420                         | 8000                        | 8020                        |
| 4         | 0 1 1 0 0   | 4              | 0 1 1 0 1                | 13        | 169                         | 182                         | 2197                        | 2210                        |
| Sum       |              |                |                          |           | 1609                        | 1686                        | 36245                       | 36322                       |
| Average   |              |                |                          |           | 402                         | 421                         | 9061                        | 9080                        |
| Maximum   |              |                |                          |           | 784                         | 812                         | 21952                       | 21980                       |

Table 7 Mutation parameters

| String No | Crossover After Crossover | Mutation After Crossover | Value of x | Fitness Function f(x) = x^2 | Fitness Function f(x) = x^2 + x | Fitness Function f(x) = x^3 | Fitness Function f(x) = x^3 + x |
|-----------|---------------------------|--------------------------|-----------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 1         | 1 0 0 0 0                 | 1 1 0 0 0                | 24        | 576                         | 600                         | 13824                       | 13848                       |
| 2         | 1 1 1 0 0                 | 1 1 1 0 0                | 28        | 784                         | 812                         | 21952                       | 21980                       |
| 3         | 1 0 1 0 0                 | 1 0 1 0 0                | 20        | 400                         | 420                         | 8000                        | 8020                        |
| 4         | 0 1 1 0 1                 | 1 1 1 0 1                | 29        | 841                         | 870                         | 24389                       | 24418                       |
| Sum       |                            |                          |           | 2601                        | 2702                        | 68165                       | 140266                      |
| Average   |                            |                          |           | 650                         | 675                         | 17041                       | 35066                       |
| Maximum   |                            |                          |           | 841                         | 870                         | 24381                       | 24418                       |

Four pairs of graphs that are drawn based on the parameter values given in Tables (1 to 7) are presented above. In all eight graphs, the defined fitness functions represented by numerical values are taken along the x-axes, while the probability is taken along the y-axes, as shown in Figure 2(a), Figure 3(a), Figure 4(a) and Figure 5(a). The y-axes in the rest of the graphs, i.e., Figure 2(b), Figure 3(b), Figure 4(b) and Figure 5(b), represent the values of the count (in seconds) against the very same fitness functions as in the former graphs.
The first pair of graphs provides information based on the fitness function defined as $f(x) = x^2$, the second being $f(x) = x^2 + x$, the third being $f(x) = x^3$ and $f(x) = x^3 + x$. Figure 2(a) illustrates that the probabilities for different fitness functions represent almost a straight line passing through the origin with a slope of $45^\circ$ with the horizontal. Although the graph has been plotted only with four randomly chosen initial population values, using decimal numbers, the probability for any other fitness function will be well within the straight line without any fluctuations.

Figure 2(b) illustrates a comparison of the counts as observed by the application of genetic algorithms used in the computational processes with the values of the actual counts. Although the shape of the actual count very closely resembles a straight line as that of the probability for different fitness functions [Figure 2(a)], we observe a fluctuation of the observed counts. This substantial fluctuation in the above observed counts is due to the splitting of their values, which are governed by the set of algorithms used in the title paper. The shape of the graphs resembles that of a slanting alphabet ‘Z’. However, the shapes of the graphs for the fitness functions $f(x) = x^3$ and $f(x) = x^3 + x$ resemble an inverted slanting ‘Z’, and the breadths of these figures depend on the scale of the fitness functions taken along the x-axes in all the graphs.

A significant observation is the fact that quite similar behavior is observed when we use genetic algorithms (GAs) as an optimization and search technique based on the principles of genetics and natural selection, as reported in the literature [Genetic algorithms and their applications in mechanical engineering by Mohamed and Bajpai, IJERT, Vol. 21 (5), 2013][26].

We note that the behaviors of Figure 3(a), Figure 4(a) and Figure 5(a) are similar to that of Figure 2(a). Although the behaviors of Figure 2(b) and Figure 4(b) are quite similar, Figure 3(b) and Figure 5(b) illustrate an inverted slanting alphabet ‘Z’. The widths of Z in all three graphs are different, which may be due to the difference in the definitions of the fitness function. A detailed study to explore the splitting of counts is worth attempting.

In the second part of the paper a novel idea is introduced, in the sense, the initial population values are represented not as a decimal number but as a binary coded digits and they are taken along the x-axes in all the twelve graphs and they are the four groups of graphs, each containing a set of three graphs providing information for the four randomly defined fitness functions, namely, $f(x) = x^2$, $f(x) = x^2 + x$, $f(x) = x^3$ and $f(x) = x^3 + x$. They have the same initial population values taken along their x-axes. (A collection of an ordered list of artificial chromosomes wherein every chromosome represents a sequence of nodes is known as the initial population. An important fact is that each gene of a chromosome takes a no delabel in such a way that no node appears twice in the same chromosome). As in the traditional binary coded genetic algorithm, the variables of interest are coded as binary digits (genes), which are plotted along the x-axes in all the above graphs. In the first set of graphs wherein probability versus initial population is plotted, the curves resemble an
inverted part of a nonnegative sine wave [Figure 6(a), Figure 7(a), Figure 8(a), and Figure 9(a)]. It clearly indicates that the probability varies between 0 and 0.5 for all four defined fitness functions. It is also observed that this variation is quite similar to the variation of probability (between 0 and 0.5) for any value of x, as shown in the former set of graphs presented above.

The second set of curves is plotted between the initial population taken along the x-axes and counts taken along the y-axes. The two graphs in Figure 6(b) and Figure 7(b) illustrate that the variation in the observed count is well in phase with the actual count. However, there are variations in the values of the counts. Both plots clearly illustrate that the observed counts are less than the actual counts observed, and both coincide at one particular value of the initial population's value, as shown in Figures 6(b) and 7(b). However, Figure 8(b) and Figure 9(b) indicate that the observed count is more than the actual counts but coincides at more than one place, as depicted by Figure 6(b) and Figure 7(b). The third set of curves is plotted for different fitness functions against the same initial population, and they are a 3-in-1 representation since three different types of curves are drawn, namely, the fitness function values, i) for offspring before reproduction ii) for offspring after crossover and iii) for offspring after mutation against the initial population. In all four plots, the offspring-after-mutation curves represent the peak values for the fitness functions, as shown in Figure 6(c), Figure 7(c), Figure 8(c) and Figure 9(c) [coloured green]. The fitness function is a defined function that measures the goodness of a solution. One should expect that it should be designed in such a way that better solutions will have a higher fitness function value than worse solutions. This seems to have been facilitated by mutation since the main goal of mutation is to obtain a solution that could not be obtained from existing genes.

The variation of the fitness function values against the initial population for offspring after crossover resembles a part of the nonnegative sine wave [coloured red]. It establishes the fact that it has never been theoretically shown that mutation is in some sense less powerful than crossover and vice versa, as pointed out in the literature [P. Bla´zej, M. Wnętrzak, and P. Mackiewicz, “The role of crossover operator in evolutionary-based approach to the problem of genetic code optimization,” Bio Systems, vol. 150, pp. 61–72, 2016.][27]. Mutation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behavior from components. The issue is the relative importance of diversity and construction, and one without the other may not be extremely useful.

The third type of curves in Figure 6(c), Figure 7(c), Figure 8(c) and Figure 9(c) are drawn for offspring before reproduction, which are generally similar to each other, representing part of the nonnegative sinewaves in all four cases. An analysis of these graphs illustrates that these variations are quite similar to the variations of i) probability and ii) count against the initial population. Generally, the objectives are conflicting for multiple-objective problems, hindering concurrent optimization of each objective. The probability of mutation, the initial population and the number of generations have been varied to study its effect on the fitness value involving genetic operations such as crossover and mutation and the present study provides evidence that the fact that the genetic operator mutation yields the peak values for the fitness function offering goodness of solution. On the whole the general observations
of all the plots offer sufficient scope for further works to deal with practical engineering problems in future.

4. Conclusion

This paper aims to introduce one of the very useful optimization methods, namely, the genetic algorithm. The flow chart vividly illustrates the various steps adopted in the present optimization method. The terminologies involved in the title paper have been explained. All the necessary parameters involved in the computational processes are tabulated. The first part of the paper clearly explains the two types of plots, and the significant observation is that the probabilities for different fitness functions represent almost a straight line passing through the origin with a slope of 45° with the horizontal. Although the graph has been plotted only with four randomly chosen initial population values, using decimal numbers, the probability for any other fitness function will be well within the straight line without any fluctuations. The shape of the graphs between the fitness function and counts resembles that of a slanting alphabet ‘Z’, and its breadth depends on the definition of different fitness functions. A significant observation is the fact that quite similar behavior is observed, as reported in the literature.

In the second part of the paper, a novel idea is introduced. In the sense, the initial population values are represented not as a decimal number but as a binary coded digit, and they are plotted in four groups of graphs as shown above. Three steps of graphs are plotted for the four randomly chosen fitness functions. In the first set of graphs wherein probability versus initial population is plotted, the curves resemble an inverted part of a nonnegative sine wave line, clearly indicating that the probability varies between 0 and 0.5 for all four defined fitness functions. The second set of curves plotted between the initial population (coded as a binary digit) taken along the x-axes and counts taken along the y-axes illustrate that the variation of the observed count is well in phase with the actual count. However, there are variations in the values of the counts, sometimes both coinciding with each other at one or more points. The third set of curves is a 3-in-1 representation, namely, initial population against the fitness function values i) for offspring before reproduction (coloured black) type-I, ii) for offspring after crossover (coloured red) type-II and iii) for offspring after mutation (coloured green) type-III. An analysis of type-I graphs illustrates that these variations are quite similar to the variations of i) probability and ii) count against the initial population, and an analysis of type-I curves resembles a part of the nonnegative sine wave. In all four types of III plots, the offspring-after-mutation curves represent the peak values for the fitness functions and offer better solutions since the fitness function is a defined function that measures the goodness of a solution. This seems to have been facilitated by the type-III curves in the title paper. It implies that when we face various practical problems depending on the requirement of the question, we could adjust the mutation rate.

Abbreviations

GA: Genetic Algorithm
Ch: child

Ch1: child 1

Ch2: child 2

F(x): Fitness Function

References

1. H. Zhang, et al., Orthogonal Nelder-Mead moth flame method for parameters identification of photovoltaic modules. Energy Conversion and Management, 211 (2020) 112-764.

2. X. Zhao, et al., Feature selection based on improved ant colony optimization for online detection of foreign fiber in cotton. Applied Soft Computing, 24 (2014) 585-596.

3. M. Wang, and H. Chen, Chaotic multiswarm whale optimizer boosted support vector machine for medical diagnosis. Applied Soft Computing, 88 (2020) 105-146.

4. X. Zhao, et al., Chaos enhanced gray wolf optimization wrapped ELM for diagnosis of paraquat-poisoned patients. Computational biology and chemistry, 78 (2019) 481-490.

5. X. Xu, and H. L. Chen, Adaptive computational chemotaxis based on field in bacterial foraging optimization. Soft Computing, 18 (4) (2014) 797-807.

6. H. Chen, et al., An enhanced Bacterial Foraging Optimization and its application for training kernel extreme learning machine. Applied Soft Computing, 86 (2020) 105884.

7. W. Deng, et al., An Improved Quantum-Inspired Differential Evolution Algorithm for Deep Belief Network. IEEE Transactions on Instrumentation and Measurement, 10 (2020) 1109/TIM.2020.2983233: 1.

8. C, H, Tsai, J. Kolibal, and M. Li, The golden section search algorithm for finding a good shape parameter for meshless collocation methods. Engineering Analysis with Boundary Elements, 34 (8) (2010) 738-746.

9. F. Yin, Y.N. Wang, and S.N. Wei, Inverse kinematic solution for robot manipulator based on electromagnetism-like and modified DFP algorithms. Zidonghua Xuebao/Acta Automatica Sinica, 37(1) (2011) 74-82.

10. S. Babaie-Kafaki and R. Ghanbari, The Dai-Liao nonlinear conjugate gradient method with optimal parameter choices. European Journal of Operational Research, 234(3) (2014) 625-630.

11. X. Zhang, et al., Robust Low-Rank Tensor Recovery with Rectification and Alignment. IEEE Transactions on Pattern Analysis and Machine Intelligence, 10 (2019) 1109/TPAMI.2019.2929043.

12. L. Davis, “Handbook of Genetic Algorithms”. Van Nostrand Reinhold. (1991).

13. E. Goldberg and David, G. Algorithms in search, optimization & Machinelearning, ISBN81-7758-829-X16-20.
14. A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 2010.
15. D. Tagu, J. K. Colbourne, and N. Negre, “Genomic data integration for ecological and evolutionary traits in non model organisms,” BMC Genomics, 15 (1) (2014) 490.
16. M. Sokouti, B. Sokouti, S. Pashazadeh, M.-R. Feizi-Derakhshi, and S. Haghipour, “Genetic- based random key generator (GRKG): A new method for generating more-random keys for one-time pad cryptosystem,” Neural Computing and Applications, 22 (7-8) (2013) 1667–1675,
17. K. A. De Jong, Evolutionary Computation: A Unified Approach, MIT press, (2006).
18. S. Sivanandamand S. Deepa, Introduction to Genetic Algorithms, Springer Science & Business Media, (2007).
19. D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison- Wesly, (1989).
20. M. Srinivas and L. M. Patnaik, “Genetic algorithms: a survey,” Computer, 27 (6) (1994) 17–26.
21. Z. Michalewicz, G. Algorithms Data Structures Evolution Programs, Springer Science & Business Media, (2013).
22. T. F. Gonzalez, Handbook of Approximation Algorithms and Metaheuristics, CRC Press, (2007).
23. M. D. Vose, The Simple Genetic Algorithm: Foundations and Theory, vol. 12, MIT Press,(1999).
24. P. Błażej, M. Wnęk, and P. Mackiewicz, “The role of crossover operator in evolutionary-based approach to the problem of genetic code optimization,” BioSystems, 150 (2016) 61–72.
25. W. M. Spears et al., Crossover or Mutation, Foundations of Genetic Algorithms 2 (1992).
26. Genetic algorithms and its applications in mechanical engineering by Mohamed and Bajpai, IJERT, 21(5) (2013).
27. P. Błażiej, M. Wnęk, and P. Mackiewicz, “The role of crossover operator in evolutionary- based approach to the problem of genetic code optimization,” Bio Systems, 150 (2016) 61–72.

Figures
**Figure 1**

Flow chart for genetic algorithm

**Figure 2**

(a). Probability versus Fitness Function \( f(x) = x^2 \)  
(b). Count versus Fitness Function \( f(x) = x^2 \)
Figure 3
(a). Probability versus Fitness Function \( f(x) = x^2 + x \)  (b). Count versus Fitness Function \( f(x) = x^2 + x \)

Figure 4
(a). Probability versus Fitness Function \( f(x) = x^2 + x \)  (b). Count versus Fitness Function \( f(x) = x^3 \)

Figure 5
(a). Probability versus Fitness Function \( f(x) = x^3 + x \)  (b). Count versus Fitness Function \( f(x) = x^3 + x \)
Figure 6
(a). Probability versus Initial Population \((F(x) = x^2)\) (b). Count versus Initial Population \((F(x) = x^2)\) (c). Fitness Function versus Initial Population \((F(x) = x^2)\)

Figure 7
(a). Probability versus Initial Population \((F(x) = x^2 + x)\) (b). Count versus Initial Population \((F(x) = x^2 + x)\) (c). Fitness Function versus Initial Population \((F(x) = x^2 + x)\)

Figure 8
(a). Probability versus Initial Population (F(x) = x^3) (b). Count Function versus Initial Population (F(x) = x^3) (c). Fitness Function versus Initial Population (F(x) = x^3)

**Figure 9**

(a). Probability versus Initial Population (F(x) = x^3+x) (b). Count versus Initial Population (F(x) = x^3+x) (c). Fitness Function versus Initial Population (F(x) = x^3+x)