Research Inventy: International Journal Of Engineering And Science
Vol.4, Issue 3(March 2014), PP 46-54
Issn (e): 2278-4721, Issn (p):2319-6483, www.researchinventy.com

The Genetic Algorithm for finding the maxima of single-variable functions

[1]Dezdemona Gjylapi, [2]Vladimir Kasëmi
[1](Informatics Engineering Department, Pavaresia University, Vlore, Albania)
[2](Naval Engineering Department, University of Vlore, Albania)

ABSTRACT - *The aim of this paper is to present an application of genetic algorithms for finding the maxima of single variable functions, which is being tested on complex, multi-modal, non-differentiable and not continues functions. Genetic algorithm using binary presentation of solutions, ranking selection and elitist selection, as well as one point crossover and mutation for each bit, in fact concludes that if the function has only a maximum all strings will converge to that value, but if the function is sufficient complex the algorithm finds a maxima very close to the real maxima of the function, in a reasonable number of generations.*

Keywords - *Genetic algorithms, maxima, multimodal function, solution, stochastic sampling*

I. INTRODUCTION

The algorithms for finding the functions' maxima being typically a part of the optimization are generally limited for regular convex functions. However, many functions are multimodal, not continuous, and not-differentiable. Methods with stochastic selection are used to optimize these functions. While traditional searching techniques use the characteristics of the problem to define the next point of the chosen sampling (e.g. the gradient, hessian, linearity and continuity), the stochastic technique does not require such information, except the value of the function at the points being analyzed. Instead, the other sampling points are defined based more on the stochastic rules of selection or decision making than on several deterministic rules. Genetic algorithms are used to solve difficult problems that have as objective functions that do not possess "good" features such as continuity, derivability, fulfillment of Lipchitz conditions, etc. [1],[3],[4],[5].

The aim of this paper is to show how Genetic Algorithms (GA) is used for finding the maximum of various functions, especially those that are complex, multimodal, non-differentiable and not continuous.
Section 2 will present the basics of GAs, in section 3 GA is tested over different multimodal continuous single-variable functions and not continuous single-variable functions to show that GA is efficient in the optimization process.

II. GENETIC ALGORITHMS OVERVIEW

A genetic algorithm applies the biological principles of natural evolution in the artificial systems. A genetic algorithm is an iterative procedure that includes a population of individuals, each of them represented by a finite string of symbols known as genes, which encode a possible solution in the space of a given problem. This space, we refer to as the searching space, includes all the possible solutions of the respective problem. GA is applied mainly in the spaces that are too big to be searched in a finite way. In Fig. 1 it is given a block scheme which presents a simple GA.

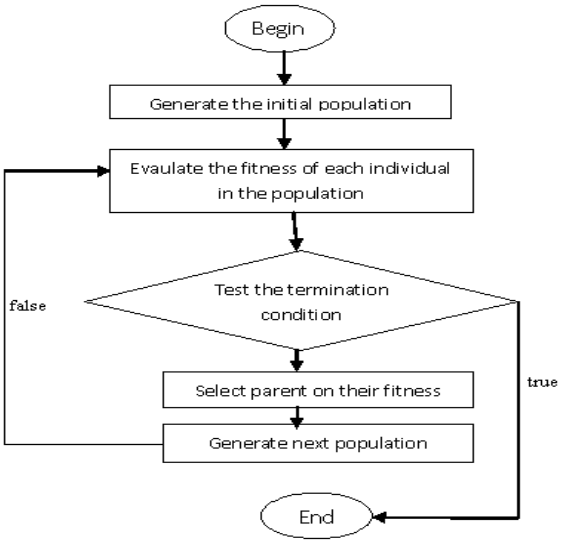

Figure 1. Block schema of a simple genetic algorithm

GAs starts with a population of individuals generated randomly, because GAs is a stochastic algorithm. Randomness plays a fundamental role in GAs, either during selection phase as well as during reproduction.
Each individual of the population represents a possible solution of the problem that is being analyzed. The individuals are evolved in the next iterations which are called generations. In each generation, the individuals are assessed according to the fitness they have. The individuals that are selected to create new individuals (offspring) are selected based on their fitness – the more adequate (fitting) they are, the more the chances they have for reproduction.

Afterwards, the population of the next generation is created through the genetic operators. This procedure continues until the termination criterion of the genetic algorithm is met (e.g. the number of the populations or the improvement of the best solution).

The usage of one GA requires the definition of six fundamental issues: Solution representation, creation of the initial population, evaluation function, termination criteria, selection function and genetic operators. Genetic operators constitute the reproduction function to generate the new population. The rest of this section we describes each of these topics.

A. Solution Representation

To better explain the steps of a genetic algorithm, we will examine a population of *N* individuals which represent possible solutions to the problem. In GA, the individuals are represented as a string of length *n* like below:

$$s = s_1 s_2 \ldots s_j \ldots s_n$$

The chromosome *s* is also called individual's genotype, whilst the corresponding possible solution *x* of the string *s* is called phenotype [6].

Usually, it is supposed that a one-to-one correspondence is established between genotypes and phenotypes. However, depending on the situation, many-to-one and one-to-many relationships are also useful. In both cases, the mapping from phenotype to genotype is called *coding*, and the mapping from genotype to phenotype is called *decoding* [6].

Coding is the process of the representation of individuals (a.k.a chromosomes). The representation can be using bits, numbers, trees, strings, lists or other objects. The coding depends mainly on the problem that needs to be solved. The typical way of coding is the binary coding where each chromosome is coded as a binary string.

Another possibility is that the whole string may represent a number and this is also the approach that we have used for finding the functions' maxima. The method of the representation by a binary string changes from problem to problem. Binary string coded with 0 and 1, are the most used.

B. Initialization and Termination

GA should generate a population to start with. In principle this can be a set of solutions generated in a totally random manner. Certainly, the initial general fitness of these individuals is very bad. The advantage of a population generated in such a manner is that it covers a big part of the solutions' space. With the size increase of the population, the biggest part of the solutions' space can be covered, but the solution will be slower. It is also possible to be covered only a given subset of the solution's space, i.e.: if we know where to search for an optimal solution.

The genetic algorithm passes from one generation to the other until the termination criteria of the algorithm is met. The most used criteria for the termination of the algorithm is a fixed maximal number of generations. There is another strategy uses the criteria of the population's convergence. In general, GA pushes the major part of the population to converge in one single solution. So, when the sum of deviations between individuals becomes smaller than a predefined value, the algorithm can terminate. Also the algorithm may terminate as result of the improvement of the best solution after a certain number of generations. Also, different strategies can be combined together for terminating a genetic algorithm.

C. Evaluation function (fitness)

The nature obeys to Darwin's principle "the survival of the strongest", i.e., individuals with high fitness value will be reproduces more often compared to those with low fitness value. In GA, the fitness is defined in such a manner that the strongest strings have higher fitness values, and this is used to evaluate each individual of a population. We should highlight here that fitness is the only relationship between GAs and the problem to be solved, and this is the measure to select an individual to be reproduced for the next generation.

Likewise it is discussed in Goldberg [3], in the minimization's problems, like the minimization of some costs' functions z(x), introducing C_{max} that satisfies the condition $C_{max} – z(x)>0$, is preferable to be taken as a string's fitness function: $f(x) = C_{max} - z(x)$. C_{max} can be taken as the greatest value z(x) in the current population, or as the greatest value z(x) of t-generations.

Similarly, in maximization problems, such as maximizing the profit or the service, which are represented as function u(x), if $u(x) <0$ for some values of x, introduces C_{min} that satisfies $u(x) + C_{min} > 0$, thus the fitness function can be defined as $f(x) = u (x) + C_{min}$. C_{min} can be defined as the absolute value of the smallest value of u(x) in the current population or after t-generations.

But, moreover, in the case of functions' maximization we can use as fitness the function itself, and to solve the problem in the case of its negative values, we use different selection methods.

D. Selection function

The selection of individuals to create the succeeding generations plays an extremely important role in a genetic algorithm. Based on the individuals' fitness, a random selection is made in such a way that the strongest individuals have the biggest chances to be selected. An individual in one population can be selected more than once and moreover, as the selection is random; all the individuals have the possibility to be selected in the next generation. There exist some schemas for the selection process such as: the roulette and its versions, escalation technique, ranking methods, tournament and elitism selection.

The usage of the roulette limits the genetic algorithm in the maximization process because the evaluation function maps the solutions in a completely ordered set of values from R+.

In order to enable the minimization and the negativity in the fitness values, the roulette is completed with elements such as windowing and scaling.

Ranking methods make possible that evaluation function maps the solutions in a partially ordered set of values, thus enabling the process of minimization and negativity in the fitness values.

Ranking Selection means that only the range of individual fitness determines the probability of selection. In this method the population is ranked from the best to the worst according to a criterion, which depends on the problem [6].

In the case of maximization of a function, the population will be ranked from the individual with the biggest fitness to the one with the smaller fitness. The probability of the selection of each individual is determined by its rank. There are several methods to determine the probability of selecting each individual depending on whether the method is linear or not linear [6].

Likewise the ranking methods, the Tournament selection requires from the evaluation function only to map the solutions to a partial ordered set. However, this order will not assign probabilities to the individuals. Tournament selection works by randomly selecting j individuals from the current population, and introduces the best one in the new population. This procedure is repeated until N individuals are selected [2].

Another genetic operator is also the Elitism, introduced for the first time by De Jong [7]. If the fitness of an individual in the previous populations is greater than the fitness of each individual in current population, elitism preserves the individual to the next generation.

E. Genetic Operators

Genetic operators provide the basic mechanism of searching the GA. Operators are used to create new solutions based on existing solutions to the population. There are two basic types of operators: crossover and mutation. Crossover takes two individuals and generates two new individuals, while mutation changes an individual to generate a single new solution. The implementation of these two basic types of operators depends on the method used for chromosome representation [2].

1) Crossover

The main distinctive feature of GA is the use of crossover. Crossover, also called recombination, is an operator that creates new individuals from current population, combining pieces of information coming from different individuals of the population. In fact, it recombines genetic material of two parent individuals to create the offspring of the next generation [6].

Depending on how we are representing the individuals, various crossover techniques have been created. When individuals are represented by binary strings {0.1}, some of the techniques used are single point crossover, multipoint crossover and uniform crossover. Single point crossover, simple crossover otherwise, randomly chooses a point in the strings of both parents, and then in order to create two offspring, the substrings of the right side of the crossover point are exchanged.

In the multipoint crossover, several crossover points are randomly selected and in the uniform crossover a random *n*-bits mask is chosen. Parity of each bit in the mask determines for each bit corresponding to a seed, from which parent will get this bit. To be clear, for each bit position in the mask, i.e. its value "1" or "0" indicates that the first parent or second parent contributes to the value of his first successor in that position, and vice versa for the second successor.

The crossover is created with the hope that the new chromosomes will have the best part of the old chromosomes, and maybe the new chromosomes will be better. However, it is best to leave a portion of the population survive for the next generation.

Crossover Probability means how often the crossover will be performed. If the crossover probability is 100%, then the crossover creates all the offspring. More crossovers cause the calculation to be slower, but it will have greater effect and will also increase the randomness. If the probability is 0%, whole new generation is created from exact copies of chromosomes of the old population (but this does not mean that the new generation is the same as the old one). Stopping the crossover is a bad idea; in this case, randomness is the only thing we can work with [8].

2) Mutation

Mutation is a very easy method to create a new individual. This method can be used after the crossover method. A mutation can be applied in several positions, once a new individual "is born". Mutation does a very simple thing: it replaces one or more genes randomly selected in the individual's representation.

Mutation probability means how often parts of chromosome will be subject to the mutation. If the probability of mutation is 100%, the whole chromosome is changed, if it is 0%, nothing is changed.

Mutation is done to prevent the GA from falling into local optima, but should not happen very often, because then the GA will actually turn into a random search and thus reduces efficiency [8].

III. TESTING AND RESULTS

Testing is done with a single execution of the program with the following parameters:

i. The length of the gene: 10 digits after the decimal point, full part depends on the user.
ii. Size of population: 6
iii. Crossover: single point crossover
iv. Mutation: for each bit with probability 0.1
v. Selection method: Elitism and Linear Ranking using (1) for the assignment of probabilities.

$$p(i) = \frac{1}{n}[\beta - 2(\beta - 1)\frac{i-1}{n-1}], 1 \le \beta \le 2 \quad (1)$$

The application has been tested on examples of functions that present difficulties for finding their maximum values. These examples are divided into two categories: continuous functions and not continuous functions in their entire domain.

All the examples chosen, present difficulties in solving the equation obtained from equalization with zero of the first order derivative of the function. In most cases are obtained equations of such orders, which have no determined methods of solution.

In continuous functions are considered:
- Polynomial functions
- Neither Odd nor Even functions
- Rational functions with no asymptote
- Continuous transcendental functions

In not continuous functions are considered:
- Rational functions with vertical asymptote
- Rational functions with vertical and horizontal asymptote
- Transcendental functions with breaking points.

For all of the above functions are found their maximum values, through our application and to verify their accuracy, corresponding graphs are built. In the example of Figure 12, which is a case that presents no difficulty for finding its maximum value and in many other examples that do not appear in the study, the application has found after a certain number of generations, the same maximum value (or with insignificant error) as standard methods do.

A. Continuous functions

1. Polynomial neither Odd nor Even functions

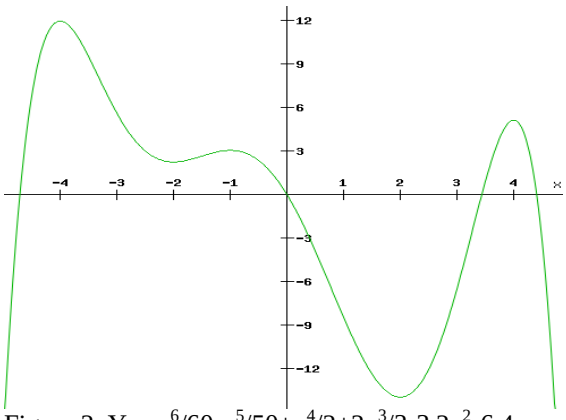

Figure 2. Y=- $x^6/60-x^5/50+x^4/2+2x^3/3-3.2x^2-6.4x$

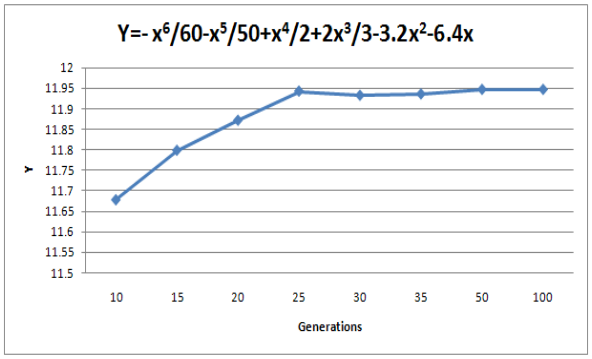

Figure 3. Fitness of Fig. 2 function depending on the generation

2. *Rational functions with no asymptote*

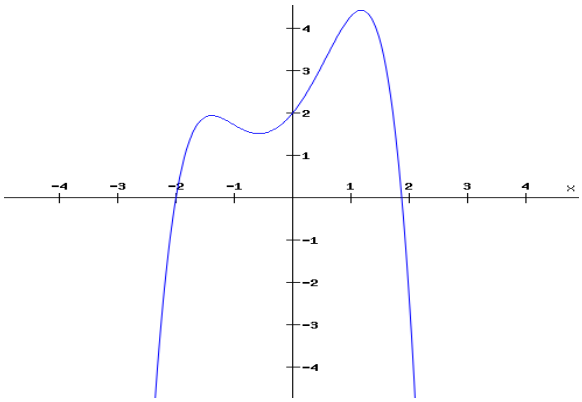

Figure 4. F(x) =-(x^6+x^5-10x^2-10x-12)/(x^2+6)

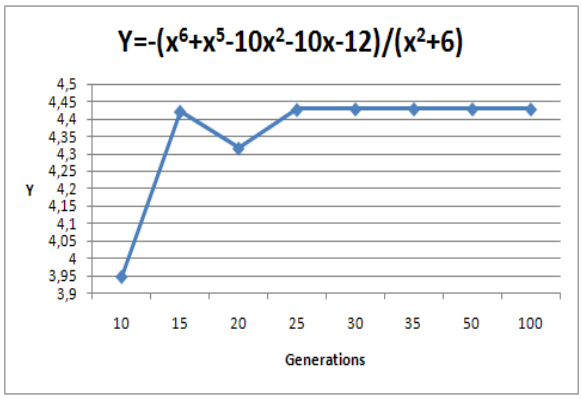

Figure 5. Fitness of Fig. 4 function depending on the generation

3. *Irrational functions*

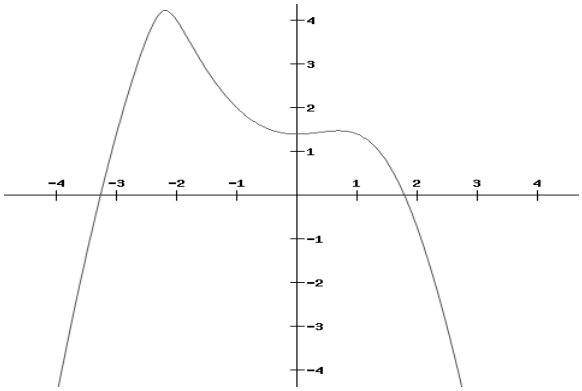

Figure 6. Y= 5-sqr(x^4+2x^3-3x^2+13)

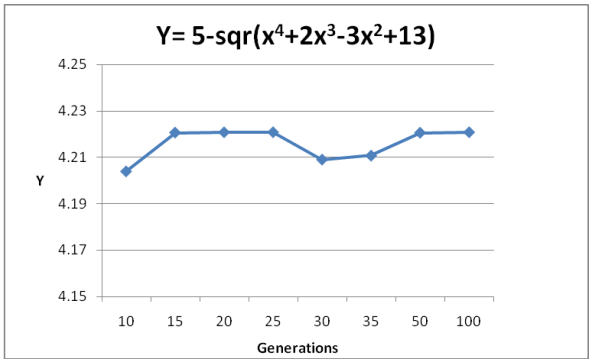

Figure 7. Fitness of Fig. 6 function depending on the generation

4. *Trigonometric functions*

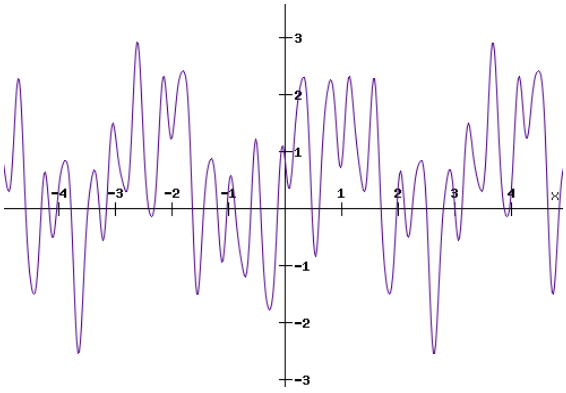

Figure 8. Y=$\sin^2$ (3x+45) +0.9$\sin^3$ (9x)-sin (15x+50) cos (2x-30)

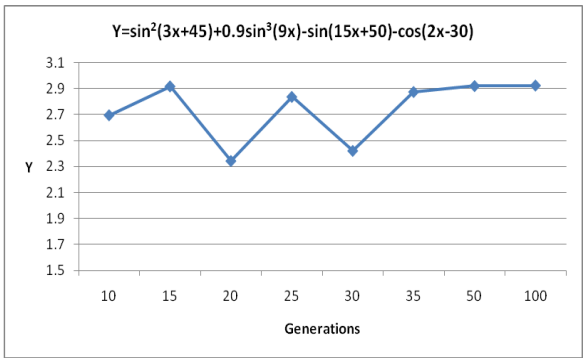

Figure 9. Fitness of Fig.8 function depending on the generation

B. Not continuous functions

1. Rational functions with vertical asymptote

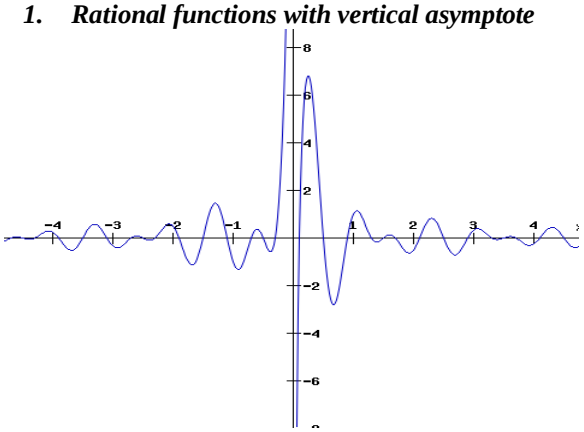

Figure 10. Y=sin (2πx)/x-cos (3πx)/x

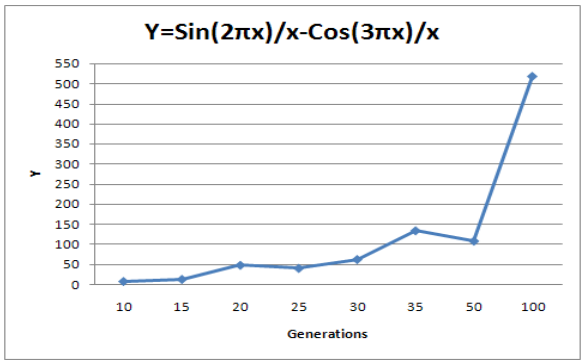

Figure 11. Fitness of Fig. 10 function depending on the generation

2. Rational functions with vertical and horizontal asymptote

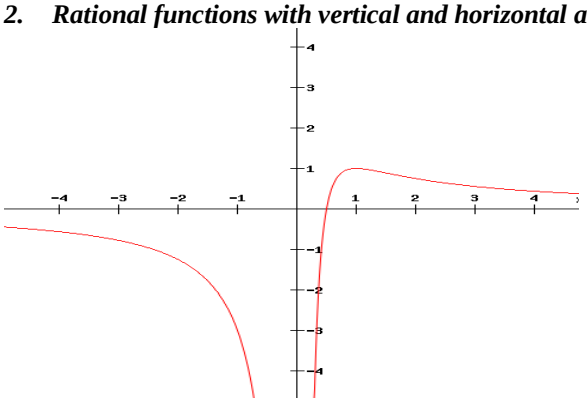

Figure 12. $F(x) = (2x-1)/x^2$

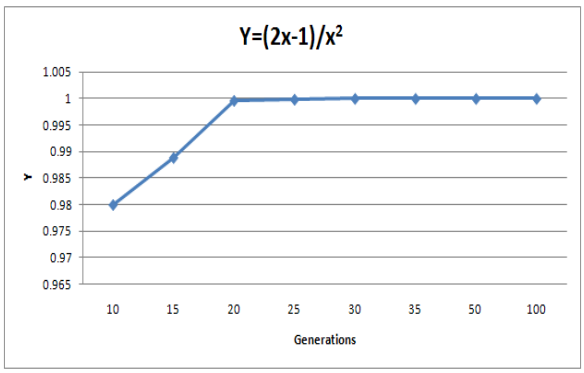

Figure 13. Fitness of Fig. 12 function depending on the generation

3. Logarithmic functions

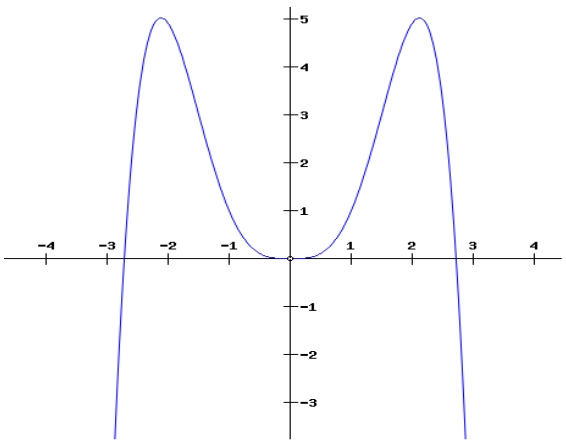

Figure 14. $Y=- x^4 (\log |x| - 1)$

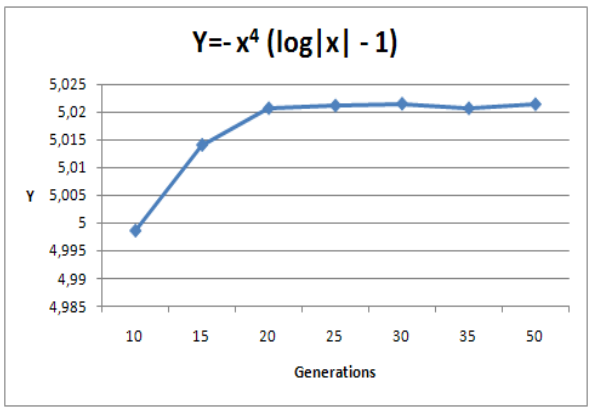

Figure 15. Fitness of the logarithmic function depending on the generation

4. Exponential functions

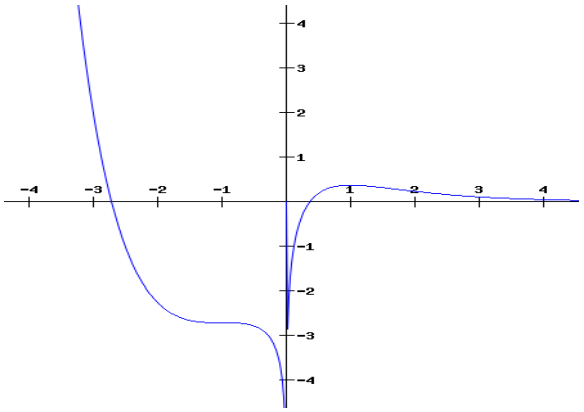

Figure 16. $Y = e^{-x} (\log |x|+x/|x|)$

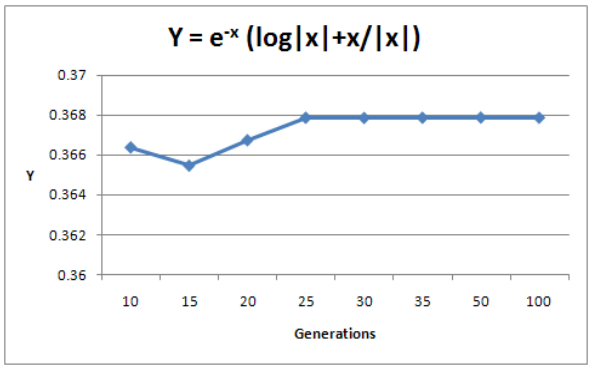

Figure 17. Fitness of the exponential function depending on the generation

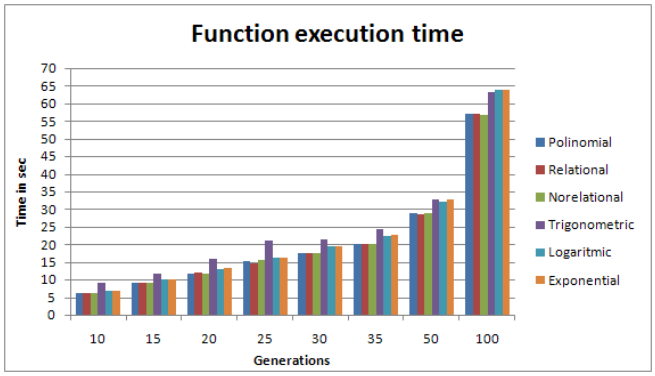

Figure 18. Execution time of each function depending on the number of generations

Figure 19. Best, worst, and average execution time depending on number of generations

IV. CONCLUSION

GA can be used in optimization problems when the solution space grows very quickly, or an exact solution is required in a limited time.

Implementation of the GA was tested successfully on the effectiveness and reliability for finding the maximum of multimodal functions, not continuous, and complex, as a part of the optimization problems.

In the case of continuous multi-modal functions, after a certain number of generations the best solution does not change.

In the case of rational function with vertical asymptotes, which goes to infinity for x approaching 0 from the negative side of the coordinate axis, the accuracy of the solution depends on the accuracy of the representation of numbers in the binary system. In this application accuracy used is 10 digits after the decimal point in binary representation. With the increasing of the number of digits after the decimal point, the genetic algorithm will find a better solution for this function.

In case of the functions that present no difficulty for finding their maximum value, the application has found the same maximum value or with insignificant error, as standard methods do.

All the functions tested are multi-modal functions. So, by using nonlinear programming methods we can take an optimal solution that varies depending on the initial values.

Whereas, despite the entire initial conditions, when problem is solved using GA, it will converge to the true optimal value. This is influenced by the fact of multiple initial values to begin with, and the crossover and mutation operations, to prevent trapping in local optimal solutions.

Even if a GA does not always give a perfect solution to a problem, it almost always can deliver at least one good solution. In order to achieve this, the four main components of the the GA work together, i.e.: parallelism, selection, mutation, and crossover.

REFERENCES

[1] Davis, Lawrence David. *Handbook Of Genetic Algorithms* (Van Nostrand Reinhold, 1991).

[2] A Genetic Algorithm for Function Optimization: A Matlab Implementation. Houck, Christopher R., Joines, Jeffery A. and Kay, Michael G. 1998.

[3] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine* (Addison-Wesley, 1989).

[4] J.Holland. *Adaptation in natural and artificial systems* (The University of Michigan Press, 1975).

[5] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs* (New York : Springer-Verlag, 1994).

[6] SAKAWA, MASATOSHI. *Genetic Algorithms and Fuzzy Multiobjective Optimization* (Kluwer Academic Publishers, 2002). pp. 12-26. ISBN 0-7923-7452-5.

[7] De Jong, K. A. *An analysis of the behavior of a class of genetic adaptive systems* (Ann Arbor,MI : University of Michigan, 1975).

[8] [Online]http://www.obitko.com/tutorials/genetic-algorithms/parameters.php