IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

Genetic Algorithm by using MATLAB Program

Mashal Alenazi

University of Bridgeport, Biomedical Engineering, USA

Abstract: In this paper, an attractive approach for teaching genetic algorithm (GA) is presented. This approach is based primarily on using MATLAB in implementing the genetic operators: initialization, crossover, mutation, evaluation and selection. A detailed illustrative examples is presented to demonstrate that how to solve Traveling Salesman Problem (TSP) and Drawing the largest possible circle in a space of stars without enclosing any of them.

Keywords: Genetic, Matlab, Algorithm, Mutation.

I. INTRODUCTION

Genetic algorithms (GA's) are adaptive methods that may be use to solve search and optimization problems. They are based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival" of the fittest. By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, if they have been suitably encoded. The basic principles of Gas were first laid down rigorously [1].

GA's work with a population of "individuals", each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. The highly-fit individuals are given opportunities to "reproduce", by "cross breeding" with other individuals in the population. This produces new individuals as "offspring", which share some features taken from each "parent". The least fit members of the population are less likely to get selected for reproduction, and so "die out. A whole new population of possible solutions is thus produced by selecting the best individuals from the current "generation", and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the population. By favouring the mating of the more fit individuals, the most promising areas of the search space are explored. If the GA has been designed well, the population will converge to an optimal solution to the problem [2].

II. METHODOLOGY

The most common type of genetic algorithm works like this: a population is created with a group of individuals created randomly. The individuals in the population are then evaluated. The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task. Two individuals are then selected based on their fitness, the higher the fitness, the higher and the chance of being selected. These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer [4]. The general form of genetic algorithms is presented in Figure 1. The contents of the blocks in the flow chart are explained in greater details in the next paragraphs.

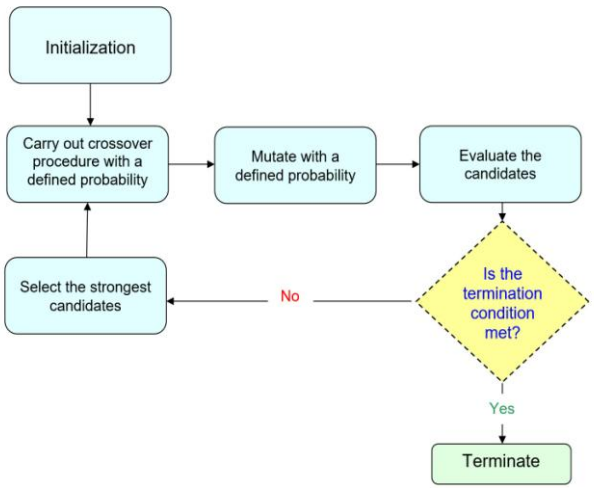


Figure 1: Flow chart of a general form of the genetic algorithm.

III.INITIALIZATION

In the initialization, one generates, often randomly, a population from which new generations are formed. At this point one also needs to de- fine the terminating condition so that the algorithm stops running once an acceptable solution is found [5].

IV. CROSSOVER

Crossover is one of the genetic operator used in producing new candidates using the features of the existing ones. The crossover procedure is illustrated in Figure 2 below.

The crossover procedure consists of three parts. First one selects two parents from the population. Then the crossover points are selected. The selection of crossover points is done at random, usually so that the distribution from which the points are drawn from is uniform. In Figure 2 two crossover points are marked with dotted lines. Once the points are defined two off springs are generated by interchanging the values between the two parents as illustrated in the figure. In the genetic algorithm

DOI 10.17148/IJARCCE.2015.41170

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

crossover is the operator that spreads the advantageous characteristics of the members around the population.

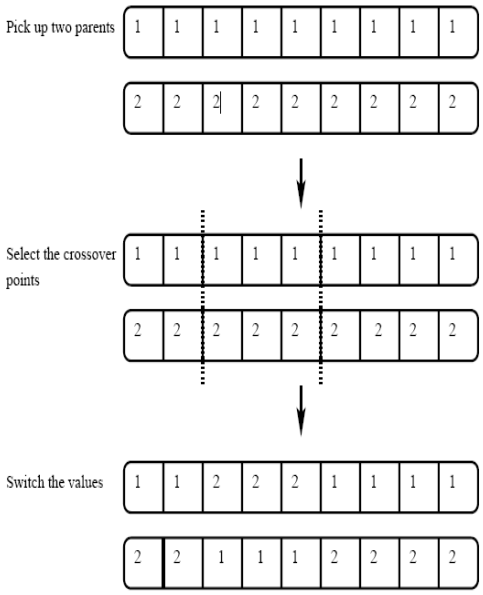


Figure 2: Crossover procedure.

V. MUTATION

In the genetic algorithm mutation is the operator that causes totally new characteristics to appear in the members of the population. In many cases the mutations, of course, result in off springs that are worse than the other members, but sometimes the result has such characteristics that make it better.

Figure 3 below demonstrates the mutation operation. First, one selects a member from the population to be mutated and a point of mutation. Then the values at the point of mutation is replaced by another value that is picked randomly from the set of all possible values [8].

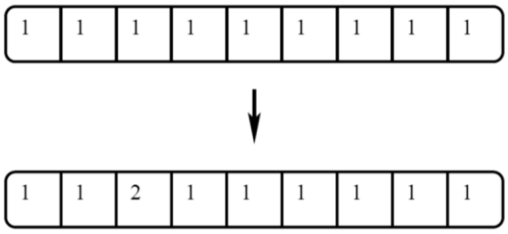


Figure 3: Mutation

VI. EVALUATION AND SELECTION

After the population is manipulated using the genetic operators, the fitness of each of the new off springs is evaluated. For this one needs to have a numerical function and fitness function. In the selection the weakest individuals in the population are eliminated. The fit off springs survive to the next generation.

VII. MATLAB IMPLEMENTATION

Initialization, For GA, a binary representation is needed to describe each individual in the population of interest. Each individual is made up of a sequence of binary bits (0 and 1). Let string length and pop size denote the length of the binary sequence and the number of individuals involved in the population. Each individual uses a string codification of the form shown in Fig. 4. Using MATLAB, the whole data structure of the population is implemented by a matrix of size pop size× (stringlength+2):

$$\text{pop} = \begin{pmatrix} \text{binary string 1} & x_1 & f(x_1) \\ \text{binary string 2} & x_2 & f(x_2) \\ \vdots & \vdots & \vdots \\ \text{binary string } popsize & x_{popsize} & f(x_{popsize}) \end{pmatrix}$$

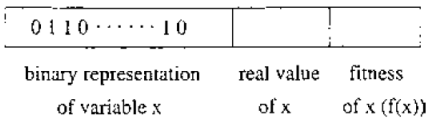


Figure 4: Binary string representation for the optimization of a one-variable function.

The first string length column contains the bits which characterize the binary codification of the real variable x. The strings are randomly generated, but a test must be made to ensure that the corresponding values belong to the function domain. The crossover and mutation operators will be applied on this string length-bit sub-string. The (stringlength+1)-th and (stringength+2)-th columns contain the real x value, used as auxiliary information in order to check the algorithm's evolution, and the corresponding f (x), which is assumed to be the fitness value of the string in this case. Then the initialization process can be completed using the code in the. In the above routine, we first generate the binary bits randomly, and then replace the (stringlength+1)-th and (stringlength+2)-th columns with real x values and objective function values, where fun is the objective function, usually denoted by a .m file.

Crossover takes two individuals parent1, parent2, and produces two new individuals child1, child2. Let pc be the probability of crossover, then the crossover operator can be implemented as the code in the. At the top of the crossover routine, we determine whether we are going to perform crossover on the current pair of parent chromosomes. Specifically, we generate a random number and compare it with the probability parameter pc. If the random number is less than pc, a crossover operation is performed; otherwise, no crossover is performed and the parent individuals are returned. If a crossover operation is called for, a crossing point (C point) is selected between 1 and string length [7]. The crossing point (C point) is selected in the function round, which returns a pseudorandom integer between specified lower and upper limits (between 1 and string length −1). Finally, the partial exchange of crossover is carried out and the real values and fitness of the new individuals' child1, child2, are computed.

Mutation alters one individual, parent, to produce a single new individual, child. Let pm be the probability of mutation, then as in the crossover routine, we first determine whether we are going to perform mutation on

DOI 10.17148/IJARCCE.2015.41170

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

the current pair of parent chromosomes [5]. If a mutation operation is called for, we select a mutating point m point, and then change a true to a false (1 to 0) or vice versa. The real value and fitness of the new individual child are then computed as code in the.

The selection operator determines which of the individuals will survive and continue in the next generation. The selection operator implemented here is roulette wheel selection and this is perhaps the simplest way to implement selection. We first calculate the probabilities of each individual being selected. Then the partial sum of the probabilities is accumulated in the vector prob. We also generate a vector rns containing normalized random numbers, by comparing the elements of the two vectors rns and prob, we decide the individuals that will take part in the new population as the code in the.

VIII. GA PARAMETER TUNING

There are three parameters related to the implementation of the genetic algorithm that must be tuned before the algorithm is used. These parameters are crossover probability, mutation probability ,and the number of generations to be produced Crossover and mutation probability effect on how the population evolves in time, and the number of generation must be chosen so that the optimal, or at least a good one, solution is found but not too many generations are produced in vain [6].

Crossover probability, In order to study the effect of crossover probability on the evolution, the algorithm is run with different crossover probabilities ranging from 0.2 to 1.0. For these runs the mutation probability is equal to 0.4, and the maximum fitness for each generation is traced. The results are shown in Figure 5 below.

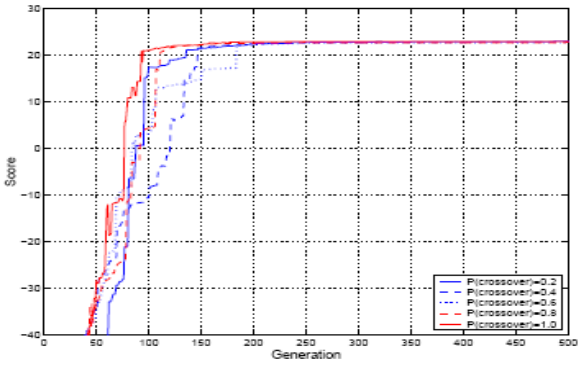


Figure 5: Evolution of fitness with different crossover probabilities ranging from 0.2 to 1.0.

From Figure 5 it can be seen that with all the values of cross over probability the evolution converges pretty nicely. Even though there are some differences between the runs, the time of convergence seems to be mostly a matter of luck. With the crossover probabilities closer to one the evolution converges slightly faster, and therefore, a value of0.8 is chosen to be the crossover probability with which the final results are generated. A probability of 1.0 is not selected because it seems to behave somewhat chaotically even though it converges fastest [11].

Mutation probability, the effect of mutation probability Is studied similarly to that for crossover probability in the previous subsection [10]. The algorithm is run with five different mutation probabilities from ranging 0.2 to 1.0. The maximum fitness as a function of generation is presented in Figure 6 for all of the five runs.

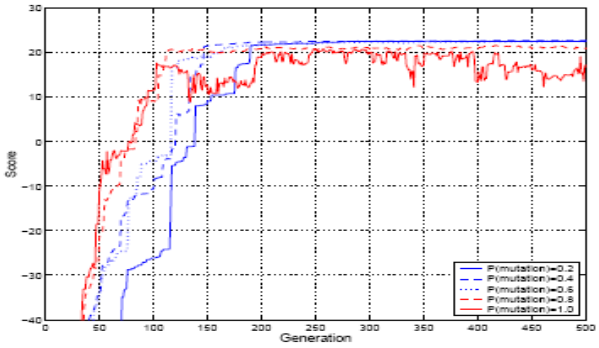


Figure 6: Evolution of fitness with different mutation probabilities ranging from 0.2 to 1.0.

Figure 6 shows that the evolution of the population significantly depends on the mutation probability. With high mutation rate the better values of fitness are reached faster but, on the other hand, the population acts chaotically and the maximum is not reached in the later generations. This shows that if the mutation probability is very high, then some of the advantageous characteristics of the generations are eliminated by mutation and the fitness of the population does not converge. With lower values of mutation probability the fitness evolves more slowly but the population do not act chaotically. Therefore, the requirement of convergence is met. A good compromise between fast evolution and nice convergence seems to be at a mutation probability of 0.4. Number of generations, the number of generation to be produced is selected so that the algorithm almost always converges to a good value of fitness. From Figure 5 and Figure 6 it can be seen that with reasonable values of crossover and mutation probability the algorithm has easily converged before the 300th generation [4]. To be absolutely sure about the convergence the algorithm is run over 400 generations. Such high confidence margin can be selected because the computations for a system of this size do not take very long for a standard computer of today. For larger systems a less conservative approach may be selected.

IX. RESULT

Traveling Salesman Problem (TSP) has been an interesting problem for a long time in classical optimization techniques which are based on linear and nonlinear programming. TSP can be described as follows: Given a number of cities to visit and their distances from all other cities know, an optimal travel route has to be found so that each city is visited one and only once with the least possible distance travelled. This is a simple problem with handful of cities but becomes complicated as the number increases.

First example, Traveling Salesman Problem (TSP)

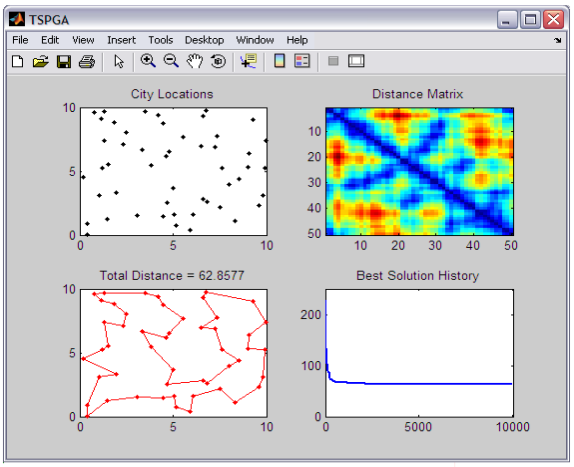
First optimization:

DOI 10.17148/IJARCCE.2015.41170

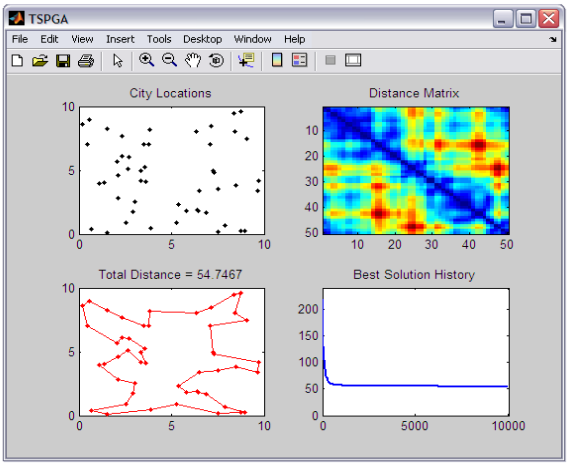
IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015



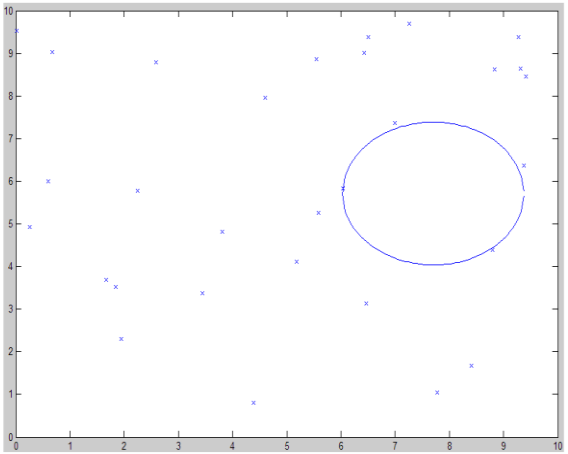
Second optimization:



The first and second optimization show that when the number of iteration increase, the total distance of selenium decrease.

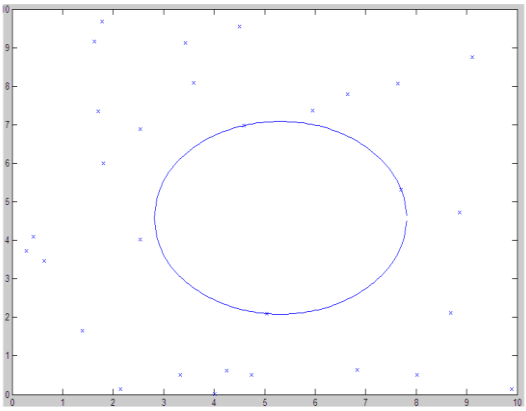
Second example, draw the largest possible circle in a space of stars without enclosing any of them.

First analysing



The first and second analysing reveal that the circle would be big enough as well as accurate when the number of iteration increase.

Second analysing



X. CONCLUSION

The major advantage of genetic algorithms is their flexibility and robustness as a global search method. They are "weak methods" which do not use gradient information and make relatively few assumptions about the problem being solved. They can deal with highly nonlinear problems and non-differentiable functions as well as functions with multiple local optima and readily amenable to implementation, which renders them usable in real-time. This approach is based primarily on using MATLAB in implementing the genetic operators. Genetic algorithms also be extremely useful if applied in conjunction with neural networks.

ACKNOWLEDGMENT

I would like to knowledge Prof. **Prabir Patra** Head of Department of Biomedical Engineering, and Prof. **Hassan Bajwa** Associate Professor, Electrical Engineering, University of Bridgeport for their support and efforts with me.

REFERENCES

[1] Awad, M.M., 2012. A new geometric model for clustering high resolution satellite images. Int. J. Remote Sens. 33, 5819–5838.

[2] Erbaş, M., & Bıyıkoğlu, A. (2015). Design and multi-objective optimization of organic Rankine turbine. International Journal of Hydrogen Energy, 40(44), 15343-15351. Doi 10.1016/j.ijhydene.2015.04.143.

[3] Farooji, N. R., Vatani, A., & Mokhtari, S. (2010). Kinetic simulation of oxidative coupling of methane over perovskite catalyst by genetic algorithm: Mechanistic aspects. Journal of Natural Gas Chemistry, 19(4), 385-392. doi: 10.1016/s1003-9953(09)60084-0.

[4] Grefenstette, J.J., 1986. Optimization of control parameters genetic algorithms. IEEE Trans. Syst. Man Cybern. 16, 122–128.

[5] Gu, J., Gu, X., & Gu, M. (2009). A novel parallel quantum genetic algorithm for stochastic job shop scheduling. Journal of Mathematical Analysis and Applications, 355(1), 63-81. doi: 10.1016/j.jmaa.2008.12.065.

[6] J. Gu, X.S. Gu, B. Jiao, A quantum genetic based scheduling algorithm for stochastic flow shop scheduling problem with random breakdown, in: Proceeding of the 17th IFAC World Congress, 2008, pp. 63–68.

[7] Kamali Shahri M, Alavi M. Investigation of Kinetic Model of Oxidative Coupling of Methane over Mn/Na2O3/SiO Catalyst [M.Sc. thesis], Iran University of Science & Technology, 2007.

[8] K.H. Han, K.H. Park, C.H. Lee, J.H. Kin, Parallel quantum-inspired genetic algorithm for combinatorial optimization problem, in: Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, May 2001, pp. 1422–1429.

[9] Kiatkittipong W, Tagawa T, Goto S, Assabumrungrat S, Silpasup K, Praserthdam P. Chem Eng J, 2005, 115(1-2): 63.

[10] S. Baluja, Structure and performance of fine-grain parallelism in genetic search, in: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, pp. 155–162.

[11] Wang, T.-S., Yu, T.-T., Lee, S.-T., Peng, W.-F., Lin, W.-L., & Li, P.-L. (2014). MATLAB code to estimate landslide volume from single remote sensed image using genetic algorithm and imagery similarity measurement. Computers & Geosciences, 70, 238-247. doi:10.1016/j.cageo.2014.06.004.