Journal of Basic and Applied Engineering Research
p-ISSN: 2350-0077; e-ISSN: 2350-0255; Volume 4, Issue 2; January-March, 2017, pp. 157-161

http://www.krishisanskriti.org/Publication.html

Genetic Algorithm Performance Assessment by Varying Population Size and Mutation Rate in Case of String Reconstruction

Ganpati Prasad[1], Darpan Singh[2], Abhinav Mishra[3] and Vishal H. Shah[4]

[1,2,3,4]Birla Institute of Technology MESRA, Ranchi, Jharkhand, India-835215
E-mail: [1]ganpati10714@gmail.com, [2]darpan466@gmail.com, [3]mishra.ab@hotmail.com, [4]vishalhshah@bitmesra.ac.in

Abstract—*This paper is aimed at studying about parameters of genetic algorithm. Population size and mutation rate are the two parameters which are being varied to check its effects on the solution. The quality as well as time taken to arrive at the solution is studied carefully. MATLAB is used for implementing genetic algorithm. Reason for choosing MATLAB is it executes array computations very fast and our problem statement have large computations of arrays. A simple problem statement of reconstructing a given string is taken and genetic algorithm is implemented on it. Tournament is conducted to decide the parent's pool. Individuals in parent's pool reproduce by uniform crossover method. The child of parents undergo mutation with a given probability. This forms a new generation. This whole process is repeated until stagnation in quality of solution takes place or the required solution is found.*

1. INTRODUCTION

In 1950s a professor at University of Michigan, John Holland studied the theory of evolution by Charles Darwin and tried to simulate the same theories on computer. These are known as genetic algorithms [1]. There are three principles of Darwinian evolution which must be present in a traditional genetic algorithm. These three principles are as follows:

Heredity: If the parent is able to survive long enough and reproduces with other individuals then, some characteristics of the parent must be present in the child. This property is known as heredity.

Variation: The population must have a variety in characteristics for evolution to occur. If each and every individual would be possessing same traits, then evolution would never take place. Every child would look like the parent and since every parent has same characteristics, combination of different traits would not occur.

Selection: A selection mechanism must be in place to decide which individual gets the chance to pass on its set of traits. This principle is known as "survival of the fittest". The term fittest is misleading. It doesn't mean bigger, stronger or faster. "Fittest" means the individual who has best adapted according to the requirement of the environment. Individuals must be tested to determine whose traits are better suited for the environment. The fitter individuals have a greater likelihood of surviving and reproducing.

These three principles form the essence of an implementation of genetic algorithm. But a genetic algorithm itself is divided into three parts [1]. These three parts are as follows:

1.1. Initialization:

This part of algorithm occurs only once i.e. at the beginning. It is the part where variation principle is applied. We generate a population with a variety of traits. Larger the population, higher is the diversity. If population size is less, there will be less combination of individuals possible, hence less chances of evolving to optimal solution. An individual is represented by a set of properties, characteristics or traits, this set is virtual DNA of the individual. Now this DNA has two kinds of representation. One is genotype and another is phenotype [1]. Genotype is the actual digital information which gets passed on generation to generation. On the other hand, phenotype is the visual expression of digital information. For example, 255 represents white color, whereas 0 represents black color. Here, 0 and 255 are genotypes, for which white and black are phenotypes respectively. In some cases, genotype and phenotype are same. The problem statement we have taken is of reconstructing a given string. In this problem, genotype and phenotype are same. The DNA data itself is a string of characters and the expression of that data is that very string.

1.2. Selection:

This part of algorithm is repeated for each generation. In this part, we apply selection principle of Charles Darwin. Selection procedure is divided into two parts, fitness evaluation, creating parent's pool [2].

1.2.1 Fitness evaluation: Fitness evaluation is done so that we can numerically determine which individual is better than others. The input of the function are the individuals of the

population and output are the numerical fitness scores of the respective individuals.

1.2.2 Creation of parent's pool: When every individual is assigned a fitness value, selection of individuals is done on the basis of their fitness value. There are various methods of selection, such as:

Tournament: A series of match is conducted between randomly picked individuals, winner of each match is selected as a parent [1][2].

Truncation: Individuals are sorted according to their fitness values and first fifty percent are selected as parents, rest are removed from the population. This forces elitism, but can cause premature convergence of the genetic algorithm [2].

Roulette wheel method: All the individuals are assigned a probability of selection on the basis of their fitness value. This gives an individual with very low fitness value some chance to pass on its DNA [1]. This method is good for sustaining variation in population. On the basis of this probability, selection of individuals takes place [2].

1.3. Reproduction:

After selection of individuals has taken place, process of reproduction occurs. Here heredity principle is applied. The child of parent individuals must have DNA of the parents involved up to some extent. There are two ways of reproduction. Single parent, where only one parent is needed for reproducing a child. Here child is clone of parent, except for a few traits where mutation occurs. Mutation is an operator of genetic algorithm through which some part of DNA is randomly changed [1][9]. In nature, it occurs because of error in copying parent's DNA. In genetic algorithms, mutation operator is introduced to keep variation in population. Apart from single parent reproduction, two parent reproduction also takes place. In this type of reproduction, crossover of DNA takes place. Crossover can be uniform, one point or two point.

Uniform crossover: Here, both parents contribute equally to child's DNA, but the part of the DNA which will be copied is completely random. Also, the parts of DNA being copied may or may not be continuous. From a no. of points, DNA will be copied from each parent.

One-point crossover: Here one parent may contribute less towards the child. This happens because, up to a single point first parent's DNA is copied. After the point, second parent's DNA is copied. The percentage of DNA contribution of each parents depend upon the point of crossover. If it is a midpoint then, contribution would be equal.

Two-point crossover: Here same principle from one point crossover is present with slight modification. Here, two points of crossover are present. Up to first point, first parent's DNA is copied, after first point second parent's DNA is copied. After reaching second point, again first parents DNA is copied.

Mutation: After crossover is completed, slight mutation is introduced in the DNA of the child being reproduced. Mutation is not a compulsory step. But mutation keeps variation in population alive. Mutation is described in terms of rate. If we have a mutation rate of 1%, this means that for bit of DNA generated from crossover, there is a 1% chance that it will mutate [9]. In our problem statement of reconstructing strings, it means that for each character of string, there is 1% chance that a certain character would change to new random character. The mutation rate is kept very low because a higher rate would change the fitter parts of the DNA received from parents, disrupting the process of evolution.

1.4. Overview of the described genetic algorithm

Step 1: Initialize: Create a population of N individuals, each with randomly generated DNA.

Step 2: Selection. Evaluate the fitness of each individual of the population and build a mating pool.

-Stop the algorithm if any of the following evaluates to true:

- If fitness of individuals has reached stagnation
- If required level of fitness is achieved
- If time of genetic algorithm is over
- If pre-decided number of generation have passed.

Step 3: Reproduction. Repeat N times:
a. Pick two parents with probability according to relative fitness.
b. Crossover—create a "child" by combining the DNA of these two parents.
c. Mutation—mutate the child's DNA based on a given probability.
d. Add the new child to a new population.

Step 4. Replace the old population with the new population and return to Step 2.

Step 5: Display the fittest individual's phenotype.

It can be easily interpreted that population size and mutation rate are one of the key parameters of a genetic algorithm [8]. In this paper, our aim is to vary mutation rate and population size, and evaluate performance of genetic algorithm on a simple problem of reconstructing a given string. The paper is organized in the following way; section II consists of problem statement and method of implementing genetic algorithm on it; in section III, population size and mutation rate are varied and graphs are plotted to evaluate the performance of genetic algorithm; section IV comprises of conclusive remarks.

2. PROBLEM FORMULATION

The problem statement is as follows, a string is given, which needs to be reconstructed using genetic algorithm. Main

purpose of choosing this problem statement is to evaluate the performance of genetic algorithms by varying parameters. The string we are considering for reconstruction is "To be, or not to be: that is the question".

First, we define a population size for the run of genetic algorithm. Then, we check the length of the given string. This length of the string is the length of the DNA. As our genetic algorithm instructs us, our first step should be to initialize the population [4][5][7]. This done by using random integer generation function of MATLAB. We generate an array of dimensions PopulationSize-by-DNAlength. Through random integer function values in the array is completely random. This array represents our population from current generation.

Second step in genetic algorithm is selection. To perform selection, we first need to formulate a fitness function for the problem statement. Our fitness function is as follows:

The fitness function starts by converting the characters into integers and the subtracting each element of each member of the population from each element of the target string. The function then takes the absolute value of the differences and sums each row and stores the function as a mx1 matrix. The output of the function is the absolute difference between the number of characters between the target string and the individual being evaluated. As the absolute difference decreases the fitness of that particular individual increases. When the absolute difference between the individual and the target reaches zero, individual becomes the exact copy of the target string.

After each and every individual is assigned a fitness value, parent's pool is made by the process of selection. In our problem formulation, we have implemented tournament selection process.

In tournament selection process, number of matches being played is double the number of individuals in the population. We randomly choose any four individuals per match. The fittest individual out of the four, is declared winner of match and put in parent's pool. At the end of tournament, we have one winner from each match, which means total number of individuals in winner's pool (AKA parent's pool) is double the number of population size.

To reproduce, crossover is performed between the individuals present in the parent's pool. Individuals in odd index places, perform crossover with the individuals in even index places. We are performing uniform crossover, through which, child we have fifty percent string characters from first parent and rest fifty percent from second parent.

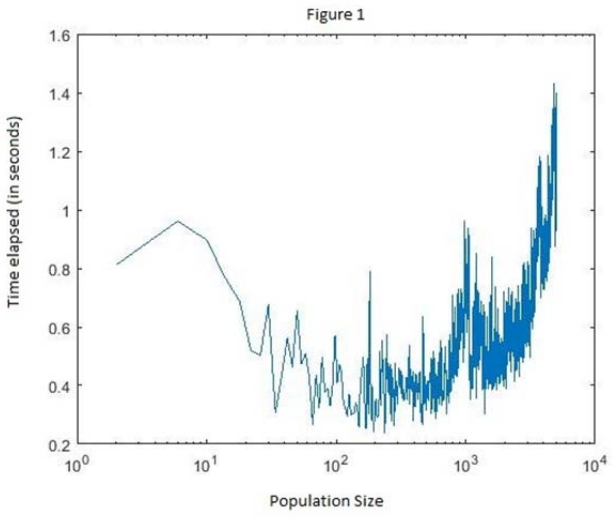
After crossover, mutation operator is applied. A mutation rate is pre-defined for a run of genetic algorithm. According to mutation rate, randomly any character in individual string is changed.

This forms the next generation of population. The previous generation is discarded and the newly formed generation replaces it. The process of creating new generations is continued until required string is formed.

3. EXPERIMENTAL RESULTS AND DISCUSSION

3.1 Varying population size:

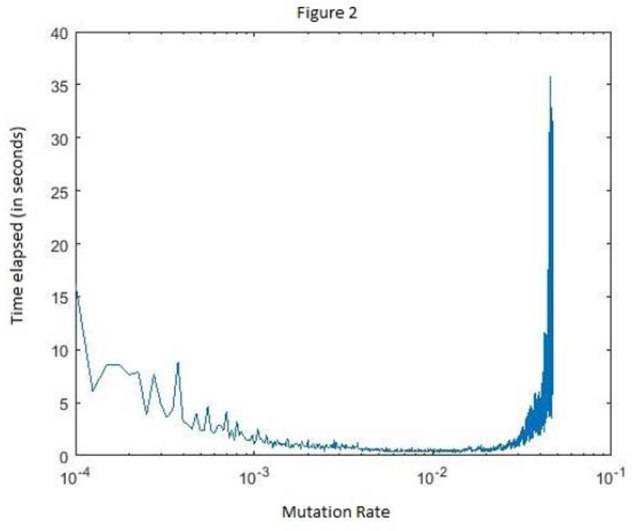
To evaluate the performance of genetic algorithm, the population size was varied from 2 to 5000. All the other parameters were kept constant. All the experiments were performed on Intel Core i5 2.30 Gigahertz quadcore processor. 8 Gigabytes of RAM was installed on the computer. Operating System was Windows 10. MATLAB 2016 was used to perform the simulations. Mutation rate was kept constant at 0.01. Graph below is plotted on a semi logarithmic graph. X-axis represents populations size, Y-axis represents time elapsed in seconds. In case, improvement of fitness levels stopped in a genetic algorithm, the program was terminated at generation 10,000. In figure 1, it can be observed that, a very small population size causes an increase in computation time. This happens because of a very simple reason. In a very small population, diversity is very low. This causes less crossover combinations possible for child reproduction. If there are more crossover combinations possible, the genetic algorithm would not converge to local optima, and search for global optima. When the populations size is increased, the computation time decreases drastically. But after a certain point, increasing the populations size causes the computation time to increase again. This happens when population size becomes more than 1000. After this point the number of individuals that need to be evaluated under fitness criteria is too much for computer processor to handle. The spikes which can be seen in the graph (figure 1) are due to randomness of initialization function. In certain runs of genetic algorithm, the first generation itself has the required solution in it. In such cases, computation time is very low.



In some other runs, the randomly generated initial population doesn't have the favorable solutions or the required diversity. In such cases, the optimal solution is reached only after, several generations of mutations. This also causes a drastic increase in computational time. Controlling these conditions are beyond the scope of a genetic algorithm. In most of the cases, an ideal population size should be between, 100 to 1000.

3.2 Varying mutation rate:

To evaluate the performance of genetic algorithm the mutation rate was varied from 0.01% to 5%. All the other parameters were kept constant. All the experiments were performed on Intel Core i5 2.30 Gigahertz quadcore processor. 8 Gigabytes of RAM was installed on the computer. Operating System was Windows 10. MATLAB 2016 was used to perform the simulations. Population size was kept constant at 1000. Graph below is plotted on a semi logarithmic graph. X-axis represents mutation rate, Y-axis represents time elapsed in seconds. In case, improvement of fitness levels stopped in a genetic algorithm, the program was terminated at generation 10,000. In figure 2, it can be observed that, at 0% mutation rate, it took more than 15 seconds to find the optimal solution, whereas average computation time is around 2.5 seconds. The reason for such a high computation time, is absence of background diversity. There was only a certain level of diversity when the first generation was created. Without any mutation, diversity of population was reduced with each passing generation, hampering the process of evolution. In the latter part of graph, it can be observed that, very high mutation rate was not very beneficial either. It negated the evolutionary process itself. The many fit genes of the child were changed randomly. The middle part of the graph, has the lowest computation time. So, the ideal mutation rate, should be between, 0.001 to 0.01.



4. CONCLUSION

After, an exhaustive study of parameters of genetic algorithms, it can be conclusively said that, population size and mutation rate are one of the most influential parameters of genetic algorithms. If population size is very low, then, there won't be enough diversity to compute a global optimum solution of a given problem. On the other hand, if populations size is very large, then it would not contribute significantly towards the quality of the solution. An adequate population size gives a good quality solution in less computation time. For the problem statement considered in this paper, an optimum population size is from 100 to 1000 individuals per generation. The other, parameter studied in this paper was mutation rate. A very low mutation rate causes pre-mature convergence of the genetic algorithm to a local optimum. On the other hand, if mutation rate is very high, the algorithm searches for a global optimum in random directions. This defeats the purpose of evolutionary process itself. For the given problem statement, an optimum mutation rate should be between 0.1% to 1%.

The parameter values of a genetic algorithm largely depend on the problem statement itself. But through a little hit and trial, luck and study of genetic algorithms, adequate parameters can be found easily for any problem statement. It should be understood that, using good genetic algorithm parameters can immensely improve the performance of a genetic algorithm. In a similar problem statement [6] of reconstruction of a given image, fitness criteria can be defined using structural similarity index (SSIM) [3]. By using SSIM index, a quality reconstructed image is obtained in a less computation time.

REFERENCES

[1] Mitchell Melanie, "An introduction to genetic algorithms", fifth printing, 1999

[2] Tobias Blickle, Lothar Thiele, "A comparison of selection schemes used in genetic algorithms", Swiss federal institute of technology (ETH) Gloriastrasse 35,8092 Zurich Switzerland, version 2, 11 December 1995

[3] Zhou Wang, Hamid Rahim Sheikh, Eero P. Simoncelli, "image quality assessment: from error visibility to structural similarity", IEEE transactions on image processing, vol. 13, no. 4, April 2004

[4] Mohammed Mustafa Seddiq, "Blurred image restoration using genetic algorithm" college technical / Kirkuk, Iraq, 27/4/2006

[5] Jong Bae Kim, Hang Joon Kim, "GA-based image restoration by isophote constraint optimization", EURASIP journal on applied signal processing 2003:3, 238–243 @ 2003 hindawi publishing corporation

[6] Fengyun Qiu1, Yong Wang1, Mingyan Jiang1, Dongfeng Yuan1, "Adaptive image restoration based on the genetic algorithm and Kalman filtering", D.-S. Huang, L. Heutte, and M. Loog (eds.): ICIC 2007, CCIS 2, pp. 742–750, 2007 © springer-Verlag berlin Heidelberg 2007

[7] Dhirendra Pal Singh, Ashish Khare, "Restoration of degraded grey images using genetic algorithm", *I.J. Image, graphics and signal processing*, 2016, 3, 28-35

[8] Olympia Roeva, Stefka Fidanova, Marcin Paprzycki, "Influence of the population size on the genetic algorithm performance in case of cultivation process modelling", proceedings of the 2013 federated conference on computer science and information systems pp. 371–376

[9] Daniel Shiffman, Chapter 9- 'Genetic Algorithm', "The Nature of Code v1.0".