Cmsc 191 Exercise 2: Solving a 6-Tuple Function using the Genetic Algorithm

Gimel David F. Velasco
Department of Mathematics and Computer Science
University of the Philippines Baguio
gfvelasco@up.edu.ph

ABSTRACT
This paper demonstrates a simple implementation of the Genetic Algorithm on solving a 6-tuple function.

1. INTRODUCTION
The Genetic Algorithm is an algorithm inspired by the concept of Genetics and Natural Selection. It approaches a particular problem by looking for the fittest individual and considers it as the solution to the problem. The kind of approach made in the tests are based on how the algorithm was implemented in the article made by Hermawanto [1]. Furthermore, to solve the 6-tuple function to be presented later, the algorithm implemented by Hermawanto is used based on how the algorithm performed on a 4-tuple function.

2. METHODOLOGY
The program will seek the solution of the following 6-tuple linear equation

$$f(a,b,c,d,e,f) = a + 2b + 3c + 4d + 5e + 6f - 240 \quad (1)$$

such that a,b,c,d,e,f is in the interval [0,45]. Equation 1 is also the set fitness function or objective function of the Genetic Algorithm.

The program followed the flowchart of the Genetic Algorithm demonstrated by Hermawanto. With some, changes and modifications made. The different processes and how they are implemented in the code is further explained in the succeeding subsections. There were four different tests performed at a Maximum Generation of 50 and 100 with a Population Number of 10 and 50.

2.1 Step 1: Initialization
In the initialization step, the initial chromosome population or the first generation of chromosomes was initiated by randomizing the integer value of each allelle of each chromosome for the whole population. Each chromosome consists of integers representing a,b,c,d,e and f. This is done repeatedly until the maximum number of population was reached. Also, the crossover rate and mutation rate is initiated in this section. To avoid further complexities in the tests each run will stick to the values 0.5 for both the crossover rate and mutation rate.

2.2 Step 2: Evaluation
This step will compute for the fitness of each chromosome per given generation. It will plug each chromosome in the fitness function/objective function which is defined as

$$f(a,b,c,d,e,f) = |(a + 2b + 3c + 4d + 5e + 6f) - 240| \quad (2)$$

The values of each chromosomes' fitness is then saved in an array and then will be used in the selection process.

2.3 Step 3: Selection
The Roulette-wheel selection process is implemented in the algorithm. The program will compute for each chromosomes' probability to be selected based on their fitness. Which also means that the higher the fitness of a particular chromosome, the higher are its chances to be selected to be part of the next generation which will undergo crossover. The roulette-wheel will be simulated using the cumulative probabilities of each chromosome. After this is done the program will now 'spin the wheel' and then the parents will be selected in the crossover process.

2.4 Step 4: Crossover
In the crossover process, a number of chromosomes is selected at random based on the crossover rate. These chromosomes will be the parents of the next generation children chromosomes. Once the selection of parents is finished, the crossover process will then be done in the parents. The algorithm implements a single point crossover at a random position in each of the chromosome. And the chromosomes will be crossed over with their succeeding chromosome. If ever the program wasn't able to select any parent for the next generation at all, the program will terminate. This is because the crossover rate might be very small that there was no chromosome qualified to be selected.

2.5 Step 5: Mutation
In this process, a random allelle will be selected and then the integer value of the selected allelle will then be replaced

with a new random number which of course is in the interval [0,45]. The number of mutations is based on the number of the allelles in the whole population multiplied by the mutation rate. The number of mutations is expressed in the equation below:

$$\mu = mutationrate * 6 * populationsize \quad (3)$$

where mu is the number of mutations that will be performed with a 6-allelle chromosome.

2.6 Step 6: Repeat Steps 2 to 5

The steps 2 through 5 will repeatedly be performed until the maximum number of generations is reached. After the maximum number of generations is reached, the program will then proceed to Step 7.

2.7 Step 7: Solution

The program will then compute for each chromosomes' fitness in the latest generation. After the fittest chromosome was found wherein when the objective function of the chromosome is very close to zero. The values of a,b,c,d,e and f will then be displayed.

2.8 Step 8: Multiple Runs

This is a bonus step implemented in the algorithm. This step has enclosed the whole Steps 1 through 7 in a while-loop. Steps 1 through 7 will then be repeatedly performed until a certain value of the objective functoin is reached. The terminating condition depends on the test being performed. That is, if the maximum generations and the population number gets bigger and bigger, the terminating condition becomes stricter and stricter.

3. RESULTS AND DISCUSSION

With 4 different tests performed. With varying Maximum Generations and Population size but the crossover rate and mutation rate is constant.

3.1 50 Max Generations and 10 Population Size

For this kind of parameters, the program usually wasn't able to select a new set of parents for the next generation since the mutation rate of 0.5 is not very much viable. To fix this, the terminating condition is much more loosened so that the algorithm would be able to reach the maximum generations of 50. Loosening the terminating condition means that the program can stop if it was able to find a chromosome that has a fitness of less than 5. Though the side effect of loosening the terminating condition is that the algorithm won't be able to get the exact solutions for the 6-tuple function (The best result yielded from this test is found in another file). The run time for this test averages at 0.3 second but sometimes would reach more than 1 second.

3.2 50 Max Generations and 50 Population Size

For this parameters, the program will stop when it finds the exact solution for the 6-tuple function. This is effective for this kind of test. The program will still be able to select the parents for the next generation with this kind of parameters. The fastest time performed by the program with this parameters is 0.3 second. In the other hand, the slowest time is 11 seconds. Run time averages at 4 seconds. (The best result yielded from this test is found in another file).

3.3 100 Max Generations and 10 Population Size

The program initially wasn't able to reach 100 generations since it wasn't able to select any parents for the next generation. Which implies that if the population size is small, the crossover rate should be higher. And so, the terminating condition is once again loosened just like how it was loosened in the first test. The run time averages at 0.3 second. Though the crossover rate is loosened, there are still instances when the program ends because it can't select any parent for the next generation at all. (The best result yielded from this test is found in another file).

3.4 100 Max Generations and 50 Population Size

With this kind of aprameters, the program is able to rach the desired number of generations and yet is able to get the exact solution for the 6-tuple function. Fastest run time is 2 seconds. While the slowest run time took up to 25 seconds. A test run output is printed in another file (Different solutions found for the 6-tuple function is printed in a .txt file attached).

4. CONCLUSION

With several test runs, the most advisable number of max generations and population number is 100 and 50, respectively. With this, the program was able to complete the said generations with the exact solution. The Genetic Algorithm implemented in the program is capable of solving for the 6-tuple function.

5. REFERENCES

[1] Hermawanto, D.,"Genetic Algorithm for Solving Simple Mathematical Equality Problem", Indonesian Institute of Sciences, n.d.