Experiment No. 7

**Aim:** Case study of Maximising a function using Genetic Algorithm.

**Theory:**

**Genetic Algorithm:**

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of mixed integer programming, where some components are restricted to be integer-valued.

The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

* Selection rules select the individuals, called parents, that contribute to the population at the next generation.
* Crossover rules combine two parents to form children for the next generation.
* Mutation rules apply random changes to individual parents to form children.

**Example:Maximizing a Function:**

Consider the problem of maximizing the function, f(x)= x2, where x is permitted to vary between 0 to 31.The steps involved in solving this problem are as follows:

**Step 1:** For using genetic algorithms approach, one must first code the decision variable ‘x’ into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained. The objective function here is f(x)= x2 which is to be maximized. A single generation of a genetic

algorithm is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random. Here initial population of size 4 is chosen, but an y number of populations can be elected based on the requirement And application. Table 1. shows an initial population randomly selected.

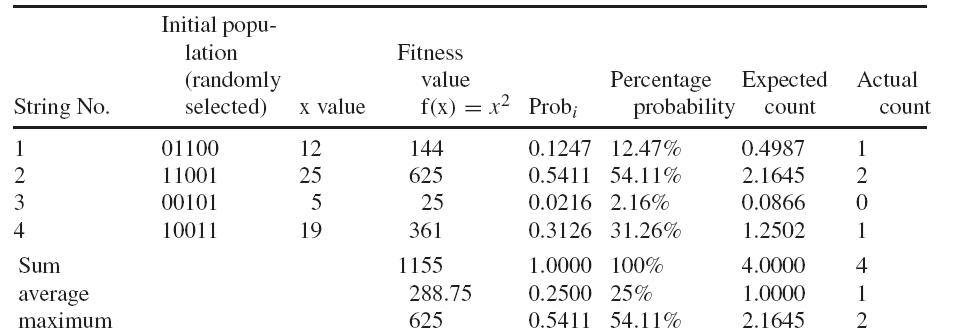


Table 1: ***Selection***

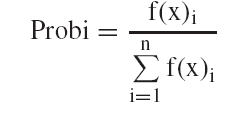
**Step 2:** Obtain the decoded x values for the initial population generated. Consider string 1,Thus for all the four strings the decoded values are obtained.

**Step 3:** Calculate the fitness or objective function. This is obtained by simply squaring the ‘x’ value, since the given function is f(x)= x2.

When, x = 12, the fitness value is f(x) = 144

for x = 25, f(x) = 625 and so on, until the entire population is computed.

**Step 4:** Compute the probability of selection,



n = no of population

f(x) = fitness value corresponding to a particular individual in the population

Σf(x)- Summation of all the fitness value of the entire population.

Considering string 1,

Fitness f(x) = 144

Σf(x) = 1155

The probability that string 1 occurs is given by,

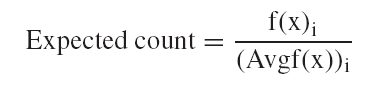
P1 = 144/1155 = 0.1247

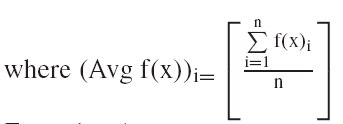
The percentage probability is obtained as,

0.1247∗100 = 12.47%

The same operation is done for all the strings. It should be noted that, summation of probability select is 1.

**Step 5:** The next step is to calculate the expected count, which is calculated as,



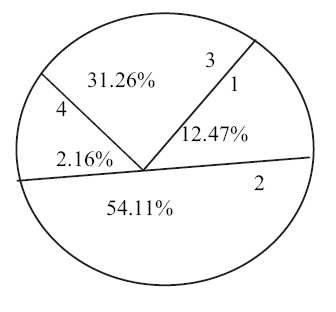


For string 1,

Expected count = Fitness/Average = 144/288.75 = 0.4987

Computing the expected count for the entire population. The expected count gives an idea of which population can be selected for further processing in the mating pool.

**Step 6:** Now the actual count is to be obtained to select the individuals, which would participate in the crossover cycle using Roulette wheel selection. The Roulette wheel is formed as shown in Fig. below. Roulette wheel is of 100% and the probability of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the no of occurrences of population is noted to get actual count. String 1 occupies 12.47%, so there is a chance for it to occur at least once. Hence its actual count may be 1. With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its actual count can be considered as 2.



On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, it actual count is 0. String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1. The above values of actual count are tabulated as shown in Table 1.

**Step 7:** Now, writing the mating pool based upon the actual count as shown in Table 2 The actual count of string no 1 is 1, hence it occurs once in the mating pool. The actual count of string no 2 is 2, hence it occurs twice in the mating pool. Since the actual count of string no 3 is 0, it does not occur in the mating pool. Similarly, the actual count of string no 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.

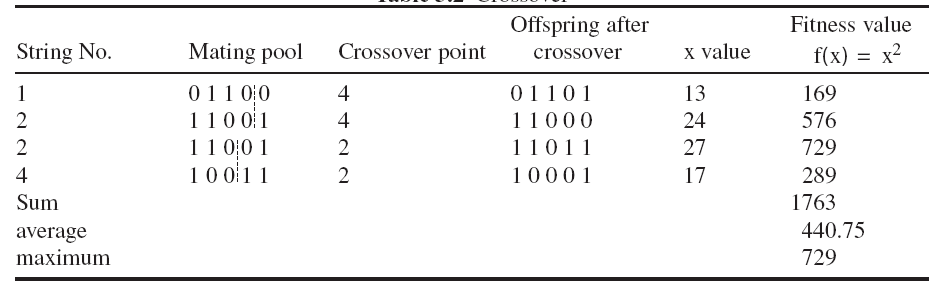


Table 2: ***Crossover***

**Step 8:** Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single point crossover is performed and new offspring is produced. The parents are:

Parent 1 0 1 1 0 0

Parent 2 1 1 0 0 1

The offspring is produced as,

Offspring 1 0 1 1 0 1

Offspring 2 1 1 0 0 0

**Step 9:** After crossover operations, new off springs are produced and ‘x’ values are decodes and fitness is calculate d.

**Step 10:** In this step, mutation operation is performed to produce new off springs after crossover operation. As discussed in mutation-flipping operation is performed and new off springs are produced.

Shows the new offspring after mutation. Once the off springs are obtained after mutation, they are decoded to x value and find fitness values are computed. This completes one generation. The mutation is performed on a bit-bit by basis.The crossover probability and mutation probability was assumed to 1.0 and 0.001 respectively. Once selection, crossover and mutation are performed, the new population is now ready to be tested. This is performed by decoding the new strings created by the simple genetic algorithm after mutation and calculates the fitness function values from the x values thus decoded.

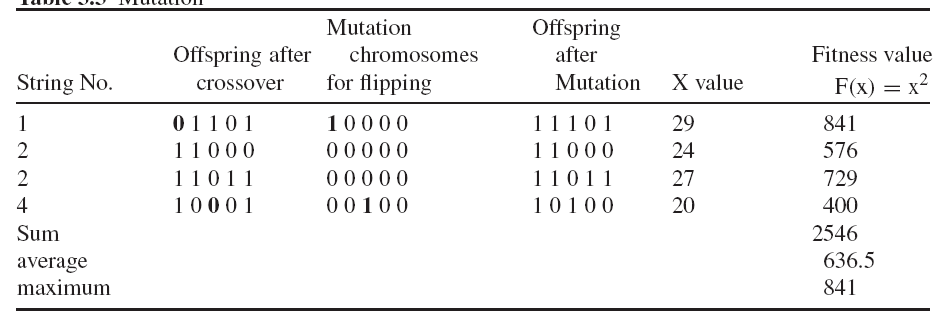


Table 2: ***Mutation***

The results for successive cycles of simulation are shown in Tables 1–3.From the tables, it can be observed how genetic algorithms combine high performance notions to achieve better performance. In the tables, it can be noted how maximal and average performance has improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 6841 during same period. Although random processes make this best solution, its improvement can also be seen successively. The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance. When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 3.2), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed. Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice.This example has shown one generation of a simple genetic algorithm.

**Conclusion:** Hence Maximising of function using genetic algorithm studied successfully. Genetic algorithm includes the selection, crossover, mutation operators along with fitness function.