**EXPERIMENT 5**

**AIM:** Program on Genetic Algorithm to solve an optimization problem in AI.​

**THEORY:**

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that are based on the ideas of natural selection and genetics. Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

They are commonly used to generate high-quality solutions for optimization problems and search problems.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

Application of Genetic Algorithms

* Recurrent Neural Network
* Mutation testing
* Code breaking
* Filtering and signal processing
* Learning fuzzy rule base

**PROBLEM STATEMENT:**

Consider the population of string with 10 bits each. The objective function can assume number of 1’s in a given string. The fitness function then perform “divide by 10”operation to normalizer the objective function. Show computation of minimum of two generations. Assume crossover rate as 0.5 and mutation probability rate as 0.05.

​

**CODE:**

import java.util.\*;

import java.lang.Integer;

import java.util.Arrays;

import java.text.DecimalFormat;

public class Genetic{

public static int MAX =1024 , BITS=10, string=4;

public static int[][] GeneString= new int[string][BITS+1] ;

private static final DecimalFormat df = new DecimalFormat("0.00");

public static String displayGene(int[] arr){

String str ="" ;

for(int i=BITS; i>0 ; i--){

str += ""+arr[i]+"" ;

}

return str;

}

public static int toDecimal(int[] arr){

int num = 0;

for(int i =arr.length -1; i>0; i--){

num = num\*2 + arr[i];

}

return num ;

}

public static int[] getGenes(){

int m = (int)(Math.floor((Math.random())\*MAX));

int[] arr = new int[BITS+1] ;

arr[0] = m;

int count=1;

while(m>0){

arr[count] = m%2 ;

count++ ;

m /=2;

}

return arr;

}

public static double[][] calcObjective(){

double avg\_f=0 , total\_f=0 , Max\_f=Integer.MIN\_VALUE ;

double[] arr\_fX = new double[string];

double[][] result = new double[string+3][4]; // Strings,Sum,Avg,Max

for(int i=0; i<string ; i++){

int numOnes =0 ;

for(int j=1; j<BITS+1 ; j++){

if(GeneString[i][j] == 1){

numOnes++ ;

}

}

double fX = (double)numOnes/10 ;

arr\_fX[i] =fX ;

total\_f += fX;

if(Max\_f < fX){

Max\_f = fX ;

}

}

avg\_f = total\_f/string;

double min\_expected = Integer.MAX\_VALUE ;

// RESULTS

for(int i=0; i<string ; i++){

// F(x)

result[i][0] = arr\_fX[i];

// Fi / tot\_f

result[i][1] = (double)arr\_fX[i]/total\_f ;

// Fi / avg\_f

result[i][2] = (double)arr\_fX[i]/avg\_f ;

// ACTUAL COUNT

result[i][3] = Math.round(result[i][2]) ;

if(min\_expected > result[i][2]){

min\_expected= result[i][2] ;

}

}

// SUM

result[string][0] =total\_f;

result[string][1] = (double)(total\_f/total\_f) ;

result[string][2] = (double)(total\_f/avg\_f) ;

// AVG

result[string+1][0] =avg\_f;

result[string+1][1] = avg\_f/total\_f ;

result[string+1][2] = avg\_f/avg\_f ;

// MAX

result[string+2][0] =Max\_f;

result[string+2][1] = Max\_f/total\_f ;

result[string+2][2] = Max\_f/avg\_f ;

return result;

}

public static int[] displayInitialise(double[][] result ,int iter){

double min\_Actual=Integer.MAX\_VALUE ;

double max\_Actual=Integer.MIN\_VALUE ;

int min\_index= -1, max\_index= -1;

int[] index = new int[2] ;

System.out.print("\n\n String No\tPopulation(P"+iter+")\tX\tf(X)\tFi/SUM(f)\tfi/f\tActual Count" ) ;

for(int i=0; i<string; i++){

System.out.print("\n "+(i+1)+"\t\t"+displayGene(GeneString[i])+"\t"+ Math.round(GeneString[i][0])+"\t"+

df.format( result[i][0])+"\t"+df.format(result[i][1]) +"\t\t"+df.format(result[i][2])+"\t"+

Math.round(result[i][3]) );

if(min\_Actual> result[i][3]){

min\_index = i;

min\_Actual= result[i][3] ;

}

if(max\_Actual<= result[i][3]){

max\_index = i;

max\_Actual= result[i][3] ;

}

}

index[0] = min\_index;

index[1] = max\_index ;

// SUM

System.out.print("\n\n \t\tSUM\t\tSUM(f)\t"+

df.format( result[string][0])+"\t"+df.format(result[string][1]) +"\t"+df.format(result[string][2]) );

// AVG

System.out.print("\n \t\tAVG\t\tf\t"+

df.format( result[string+1][0])+"\t"+df.format(result[string+1][1]) +"\t"+df.format(result[string+1][2]) );

// MAX

System.out.print("\n \t\tMAX\t \t"+

df.format( result[string+2][0])+"\t"+df.format(result[string+2][1]) +"\t"+df.format(result[string+2][2]) );

return index;

}

public static void replaceWeak(int[] index ){

System.arraycopy(GeneString[index[1]], 0, GeneString[index[0]], 0, GeneString[0].length);

}

public static int[] replaceMate(int cur , int mate , int crossSite){

int length = 5 ;

int[] arr = new int[BITS+1] ;

for(int i=crossSite+1; i<BITS+1; i++){

arr[i] = GeneString[cur][i] ;

}

for(int i=1; i<=crossSite; i++){

arr[i] = GeneString[mate][i] ;

}

return arr ;

}

public static void displayCrossOver(int iter){

System.out.print("\n\n\n CROSS OVER : ");

System.out.print("\n String No\tPopulation(P"+iter+")\tMate\tCrossover Site\tNew Pop(P"+(iter+1)+")\tNEW X" ) ;

int[][] newGeneString= new int[string][BITS+1] ;

for(int i=0; i<string; i++){

int mate = (i%2==0)? i+1: i-1 ;

int crossSite= 5;

// int[] arr = new int[BITS+1] ;

newGeneString[i] = replaceMate(i , mate, crossSite) ;

newGeneString[i][0]= toDecimal(newGeneString[i] ) ;

System.out.print("\n "+(i+1)+"\t\t"+displayGene(GeneString[i])+"\t"+(mate+1)+"\t5\t\t"+

displayGene(newGeneString[i])+"\t"+newGeneString[i][0] );

}

GeneString= newGeneString;

}

public static void mutation(){

System.out.print("\n\n String No\tPopulation(P2)\tP2 X\t Mutated Population\tMUTATED X" ) ;

int[][] gen2 = new int[string][BITS+1] ;

for(int i=0; i<string;i++){

int index;

System.arraycopy(GeneString[i], 0, gen2[i], 0, BITS+1);

do{

index =(int)(Math.floor((Math.random()\*BITS))) +1;

if(GeneString[i][index] == 0){

GeneString[i][index] =1; //mutate

break;

}else{

}

}while(GeneString[i][index] != 0) ;

GeneString[i][0] = toDecimal(GeneString[i]);

System.out.print("\n "+(i+1)+"\t\t"+displayGene(gen2[i])+"\t"+ gen2[i][0]+"\t "+displayGene(GeneString[i])+"\t\t"+GeneString[i][0] ) ;

}

}

public static void GeneticAlgo(){

System.out.print("\n\n\n \*\*\*\*\*GENERATION 1\*\*\*\*\*");

for(int i=0; i<string; i++){

GeneString[i] = getGenes() ;

// System.out.print("\n "+GeneString[i][0]+" : "+displayGene(GeneString[i] ));

}

double[][] result = calcObjective() ;

int[] index = displayInitialise(result, 0);

// REPLACE WEAK

replaceWeak(index);

displayCrossOver(0);

System.out.print("\n\n\n \*\*\*\*\*GENERATION 2\*\*\*\*\*");

double[][] result2 = calcObjective() ;

index = displayInitialise(result2, 1);

// REPLACE WEAK

replaceWeak(index);

displayCrossOver(1);

// MUTATION

System.out.print("\n\n\n \*\*\*\*\*FINAL GENERATION POST MUTATION\*\*\*\*\*");

mutation() ;

}

public static void main(String args[]){

Scanner sc = new Scanner(System.in) ;

System.out.print("\n Enter the number of Strings :”);

string= sc.nextInt() ;

GeneticAlgo();

sc.close() ;

}

}

**OUTPUT:**

**Calendar

Description automatically generatedText

Description automatically generated**

**CONCLUSION:** In this experiment, I implemented Genetic Algorithm for a population of string with 10 bits each, the objective function is the number of 1’s in a given string and the fitness function is “divide by 10” operation to normalize the objective function, assuming the crossover rate as 0.5 and mutation probability rate as 0.05. The algorithm iterated for 2 generations and in the final generation any random 0 of each population string was mutated to 1. The final mutated generation is displayed in the output.