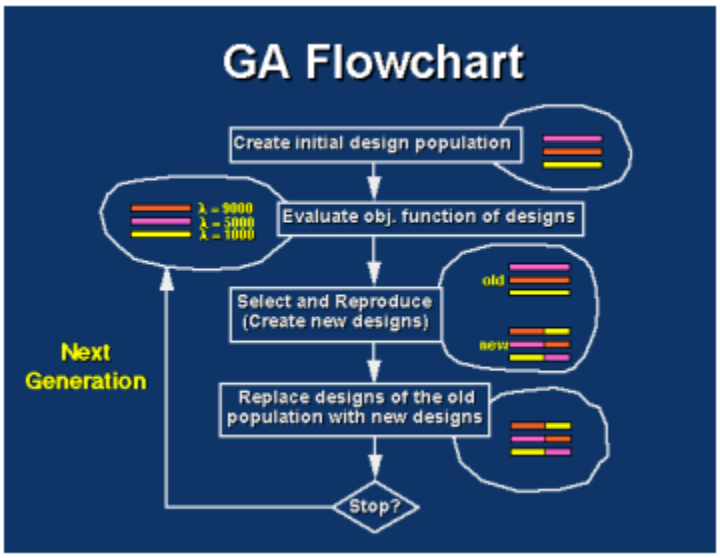
Assignment 11 – Genetic Algorithms

Submission

Submit your code files under Assignment 11 by the due date specified in Blackboard. Plan to **demonstrate** your working program to the instructor in class after the due date.

Details

Guessing a string using Genetic Algorithms. User inputs a string of arbitrary length, use genetic algorithms to guess the string.



Suggested outline

1. Three Classes are suggested (based on Java)
    a. Organism (Referred to as design in the above diagram)
    b. Population (a collection of organisms)
    c. GATest – a class to test the Genetic Algorithm
2. The Organism class
    a. Member variables
        i. A String or Array of char or an ArrayList of character. This string represents the Organism's guess for the unknown String. The number of characters in the Organism should be the same as the number of characters in the unknown String. In the discussion, we call this variable value.
        ii. We also need a goalString or unknown String we are trying to guess
    b. Methods
        i. A constructor public Organism(String goalString)
            1. Sets the goalString(goalsString is user input – for example Marshall University)

2. Initializes the variable value. If goalString is of length n, then value should have n characters each of which is randomly assigned one of {a,b,c, … w,x,y,z, ' ',A,B,C, … W,X,Y,Z}

ii. A constructor that sets goalString and value – signature is something like public Organism(String value, String goalString)

iii. A Fitness function for the class. You are free to select a fitness function. However the one demonstrated in class counted the number of positions in which value agreed with goalString. This is a simple for-loop.

If – for example - you choose to represent value and goalString as String variables, you simply code

```
Public int fitness
        Int count = 0;
        For(int I = 0; I < goalString.length(); i++)
                If(value.charAt(i) == goalString.charAt(i)
                        count ++;
```

count represents the fitness of the organism

iv. Crossover or mating method

Suggested signature be something like

Public Organism[] mate(organism other)

1. Generate a random integer – say crossover – in [0,n-1] where n is the length of value and goalString
2. Create two strings
   a. The first has the values of this from 0 to crossover + 1 to n-1
   b. The second has the values in other from 0 to crossover and the values of this from crossover + 1 to n-1
3. Create an array of two organisms
4. Use the constroctor in 2.b.ii above to fill the array. Array element 0 should have the string from iv.2.b above. Both have goalString as the other input parameter
5. Return the array of two Organisms

v. You will need a compareTo method, hence Organism will have to implement the comparable interface. Define organism A to be less than organism B if its fitness is greater. Two organisms are equal if they have the same fitness. Organism A is greater than organism B if its fitness is less. WHY!!!!! Looks reversed!

vi. You will need to write a mutate method. The signature should be

public void mutate(double mutateProb)

Recall that 0<mutateProb<1.

For each character in the string value (i=0 to value.length)

Generate a random number(4) in the interval [0,1]

If (r > mutateProb);

Else

Randomly select new character from {a,b,c, … w,x,y,z, ' ',A,B,C, … W,X,Y,Z}

Assign value.charAt(i) the character

vii. You may want to write a toString. The toString could concatenate the value of the Organism with its fitness. For example:
Value = Mbsshem1 Goal = Marshall Fitness = 4

viii. Any other methods as determined by you.

3. The Population class
   a. Member variables
      i. popSize (population size. It is better if this is an even number) The number of organisms in the population
      ii. mutateProb (mutation probability) a double between 0 and 1
      iii. numGenerations (number of generations). How many generations does the user want to compute before reporting the best answer and terminating the program.
      iv. Two arrays of Organisms. Each with popSize elements. E.g. thisGeneration and nextGeneration
      v. goalString - the same as goalString for Organism
   b. Methods: A constructor
      Public Population(String goalString, int popSize, int numGenerations, double mutateProb)
      i. Assign the class variables goalstring, popSize, numbGenerations and mutateProb
      ii. Create the array thisGeneration with popSize elements using the constructor in 2.b.i. This will initialize all the Organisms in the population to random String values. Each will have length equal to the length of the goal.
   c. You will also need a function to iterate through generations to create new populations. Here is an outline:
      Assume you have an instance of Population using the constructor above

```
Public void iterate
{
        For g = 0 to numGenerations
        {
                Sort thisGeneration using Arrays.sort
                For i = 0 to popSize/2
                {
                        Choose parent 1
                        Choose parent 2
                        Mate parent 1 and parent 2 to get 2 children
                        Mutate each child
                        nextGeneration[2*i] = child 0
                        nextGeneration[2*i+1] = child 1
                } // end I loop
                Sort nextGeneration
                Set thisGeneration = nextGeneration
                Print the best guess so far
```

```
                                        (thisGeneration[0].toString())
                    } // end g loop
                } // end iterate method
```

A sample test program

```
import java.util.Scanner;
public class GATest
{
    public static void main(String[] args)
    {

      Scanner myScanner = new Scanner(System.in);
      System.out.print("\nEnter string to guess-->");
      String goal = myScanner.nextLine();
      System.out.print("Enter number of organisms per generation-->");
      int popSize = Integer.parseInt(myScanner.next());
      System.out.print("Enter number of generations-->");
      int generations = Integer.parseInt(myScanner.next());
      System.out.print("Enter mutation probability-->");
      double mutateProb=Double.parseDouble(myScanner.next());
      System.out.println();
       Population aPopulation = new Population(goal,popSize,generations, mutateProb);
       aPopulation.iterate();
    }
}
```