Cairo University
Faculty of Computers & Artificial Intelligence
Department of Computer Science
Soft Computing Course - 2020

Lab 2 – Floating-Point Genetic Algorithm

Why do we need floating-point GA?

The binary representation traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high precision numerical problems. For example, for 100 variables with domains in the range [-500, 500J where a precision of six digits after the decimal point is required, the length of the binary solution vector is 3000. This, in turn, generates a search space of about 10^{1000}. For such problems, the binary representation in genetic algorithms performs poorly.

The floating-point representation has the advantage of higher precision and convenience of searching in great space.

The structure of the chromosome in floating-point GA:

The chromosome will be an array of floating-point variables:

X_0	X_1	X_2	X_3	…	X_i	…	X_m

where each gene has a range; $\mathbf{X_i \in [a_i,b_i]}$ (or sometimes we say $\mathbf{X_i \in [LB_i,UB_i]}$ where LB stands for "Lower Bound" and UB stands for "Upper Bound").

Remember, the basic steps performed by the canonical GA are:

- Initialization (of the population)
- *Looping over generations and performing:*
  - Fitness evaluation
  - Selection (of parents for reproduction)
  - Crossover
  - Mutation
  - Replacement

Note: In floating-point representation, all operators **except** "Mutation" are performed similarly to how they were performed in the binary representation.

- ***Mutation:*** (**Two** types of mutation can be performed on FP chromosomes)

1. **Uniform mutation:** there is an equal chance to move left or right steadily in all generations.

   It is performed as follows:

   a) Calculate ΔL_{xi} -> $\Delta L_{xi} = X_i - LB_i$

   b) Calculate ΔU_{xi} -> $\Delta U_{xi} = UB_i - X_i$

   c) Generate **r1** ∈ [0,1]:

   if **r1 <= 0.5,** $\Delta = \Delta L_{xi}$

   if **r1 > 0.5,** $\Delta = \Delta U_{xi}$

   d) Generate **r2** ∈ [0, Δ]

   if $\Delta = \Delta L_{xi}$, $X_i new = X_i - r2$

   if $\Delta = \Delta U_{xi}$, $X_i new = X_i + r2$

Note: In **all** types of mutation, **don't forget** to generate a random number at the start to compare it to (P_m) to know whether mutation will be performed on this gene (X_i)or not.

2. **Non-uniform mutation:** the operator makes big changes to the population in early stages and smaller changes in later generations until it reaches zero at the final generation.

   It is performed as follows:

   a) Calculate ΔL_{xi} -> $\Delta L_{xi} = X_i - LB_i$

   b) Calculate ΔU_{xi} -> $\Delta U_{xi} = UB_i - X_i$

   c) Generate **r1** ∈ [0,1]:

   if **r1 <= 0.5,** $y = \Delta L_{xi}$

   if **r1 > 0.5,** $y = \Delta U_{xi}$

   d) Let **Δ(t,y)** be the value of mutation at generation **t**:

   Δ(t,y) = y * (1 – r ^ ((1 - t\T) ^ b)) where:

   r: a random number between 0 and 1
   t: the current generation
   T: the max. number of generations
   b: a parameter that controls the degree of non-uniformity (∈ [0.5,5])

   e) If $y = \Delta L_{xi}$, $X_i new = X_i - \Delta(t,y)$

   If $y = \Delta U_{xi}$, $X_i new = X_i + \Delta(t,y)$

Exercise:

How can we use the GA in **"Smooth Curve Fitting"**?

Curve fitting is the process of constructing a curve, or mathematical function (polynomial equation) that has the best fit to a series of data points, possibly subject to constraints. In smooth curve fitting, **the function is constructed to approximately fit the data**.

We are given set of points and would like to fit a curve to them using a polynomial equation. We will use GA to find the **best coefficients** that would make the distance between the polynomial and the points minimum.

Given:

- **N** data points
- The degree of the polynomial (**D**)

Objective:

- Minimize the error of the polynomial equation; i.e. the distance between the polynomial and the points.

Constraints:

- All the coefficients must be ∈ [-10, 10].

Sol.:

Since we want to find the best coefficients that minimize the error, our chromosome will be represented in floating point and its size will be **D+1** where each bit represents a coefficient in the equation. Then, we can use the mean square error as our fitness function.

Assume N = 4, D = 2 and we are given this list of data points: [(1,5), (2,8), (3,13), (4,20)], so the polynomial equation will be $\mathbf{a_0 + a_1x + a_2x^2}$ and we want to find the values of a_0, a_1 and a_2. So, our chromosome will contain **3** bits.

Step 1: Let's say we started with following randomly initialized chromosomes:

C1	C2	C3	C4
1.95	4.26	3.36	0.23
8.16	-7.4	-0.3	0.12
-2	-2.5	-6.2	4.62

Step 2: Let's evaluate the fitness of each chromosome. We will need to use the mean square error ($\mathbf{1/N \sum_{i=1}^{N} (y_{calc.} - y_{actual})^2}$) between every given point and the polynomial.

For example, the fitness of C1 is:

Error at (1,5) = ((1.95 + 8.16 * 1 + -2 * 1^2) – 5)^2 = 9.67

Error at (2,8) = ((1.95 + 8.16 * 2 + -2 * 2^2) – 8)^2 = 5.15

Error at (3,13) = ((1.95 + 8.16 * 3 + -2 * 3^2) – 13)^2 = 20.88

Error at (4,20) = ((1.95 + 8.16 * 4 + -2 * 4^2) – 20)^2 = 303.1

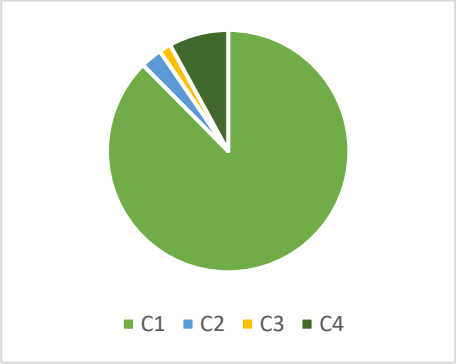
Total error = (9.67 + 5.15 + 20.88 + 303.1) / 4 = 84.7

Chromosome	*Fitness*
C1	1/84.7 = 0.011
C2	1/2766.6 = 0.00036
C3	1/4767 = 0.0002
C4	1/988.8 = 0.001

Note: We used **(1/error)** as our fitness because when we perform selection, we usually select the **"fitter"** chromosomes.

Step 3: Let's select the parents! First, we need to calculate the cumulative fitness function:

Chromosome	*Fitness*	*Cumulative Fitness*
C1	0.011	0.011
C2	0.00036	0.01136
C3	0.0002	0.01156
C4	0.001	0.01256



Continue the selection process as in the previous lab.

Step 4: Assume C1 and C4 were selected and $\mathbf{P_c}$ was greater than the random number we generated, let's perform the crossover at $\mathbf{X_c}$ = 2 for instance:

1.95	8.16	-2

0.23	0.12	4.62

1.95	8.16	4.62	O1

0.23	0.12	-2	O2

Step 5: Let's perform non-uniform mutation on the offspring:
You should iterate over **each bit** in **each offspring** chromosome, but we'll just show you the mutation on the first bit (i=0) of O1:

- Generate a random number ($\mathbf{r_m}$) between 0 and 1.
- If $\mathbf{r_m <= P_m}$:
  - $\mathbf{\Delta L_{xi} = 1.95 - (-10) = 11.95}$
  - $\mathbf{\Delta U_{xi} = 10 - 1.95 = 8.05}$
  - **Generate r1 ∈ [0,1]: if r1 <= 0.5, y = ΔL_{xi}, else r1 > 0.5, y = ΔU_{xi}**
    (Assume r1 = 0.19, therefore y = 11.95)
  - **Δ(t,y) = y * (1 – r ^ ((1 - t\T) ^ b))**
    =
    Δ(1,11.95) = 11.95 * (1 – 0.67 ^ ((1 - 1\100) ^ 1))
    =
    3.9
  - **Since y = ΔL_{xi}, therefore X_{inew} = 1.95 – 3.9 = -1.95**

Continue iterating over the bits in each offspring chromosome.

Step 6: Replace the current generation with the new offspring using any of the replacement strategies explained earlier, go to step 2 and repeat the process.