Mutation and Fitness Scalling in GAs

By
Dr. Diya Vadhwani

Important GA Operations

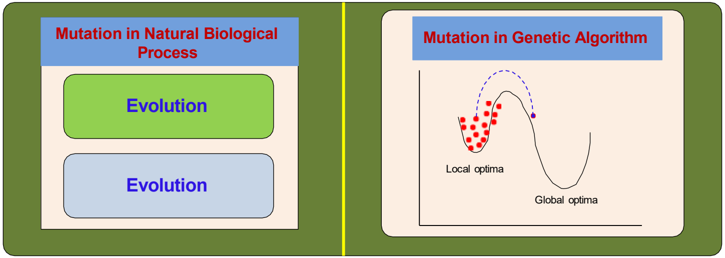
1. Encoding
2. Fitness Evaluation and Selection
3. Mating pool
4. **Reproduction**
    - Crossover
    - Mutation
    - Inversion
5. Convergence test

This lecture includes ...

1. Encoding
2. Fitness evaluation and Selection
3. Mating pool
4. Crossover
5. **Mutation**
6. **Inversion**
7. **Convergence test**
8. **Fitness scaling**

Mutation Operation

- In genetic algorithm, the mutation is a genetic operator used to maintain genetic diversity from one generation of a population (of chromosomes) to the next.
- It is analogues to biological mutation.
- In GA, the concept of biological mutation is modeled artificially to bring a local change over the current solutions.


Mutation in Natural Biological Process
Evolution
Evolution
Mutation in Genetic Algorithm
Local optima
Global optima

Mutation Operation in GAs

Like different crossover techniques in different GAs there are many variations in mutation operations.

- **Binary Coded GA :**
  - Flipping
  - Interchanging
  - Reversing
- **Real Coded GA :**
  - Random mutation
  - Polynomial mutation
- **Order GA :**
- **Tree-encoded GA :**

Mutation Operation in Binary coded GA

- In binary-coded GA, the mutation operator is simple and straight forward.
- In this case, one (or a few) 1(s) is(are) to be converted to 0(s) and vice-versa.
- A common method of implementing the mutation operator involves generating a random variable called mutation probability (μ_p) for each bit in a sequence.
- This mutation probability tells us whether or not a particular bit will be mutated (i.e. modified).

Note :

- To avoid large deflection, μ_p is generally kept to a low value.
- It is varied generally in the range of $\frac{0.1}{L}$ to $\frac{1.0}{L}$, where L is the string length.

Mutation in Binary-coded GA : Flipping

- Here, a mutation chromosome of the same length as the individual's chromosome is created with a probability p_μ of 1's in the bit.
- For a 1 in mutation chromosome, the corresponding bit in the parent chromosome is flipped (0 to 1 or 1 to 0) and mutated chromosome is produced.

1	0	1	1	0	0	1	0

offspring

1	0	0	0	1	0	0	1

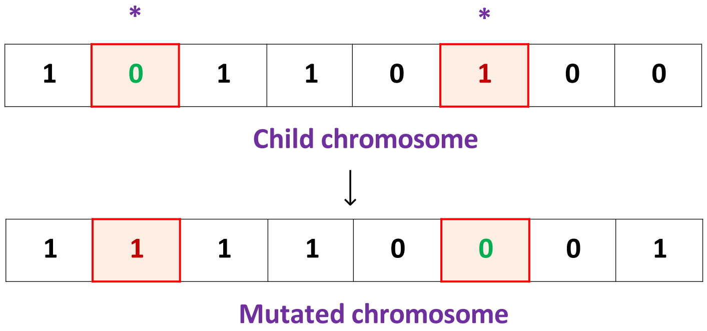
Mutation chromosome

0	0	1	1	0	1	0	0

Mutated offspring

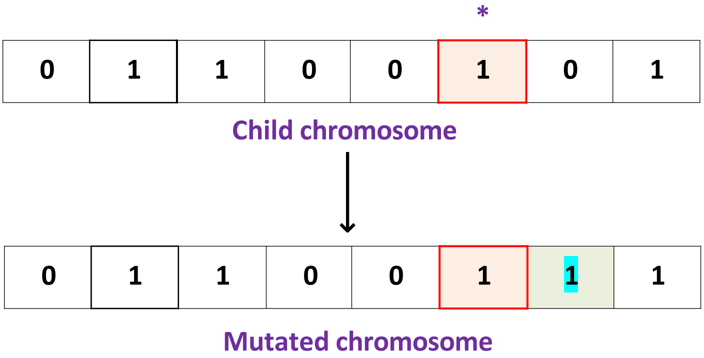
Binary-coded GA : Interchanging

- Two positions of a child's chromosome are chosen randomly and the bits corresponding to those position are interchanged.



Mutation in Binary-coded GA : Reversing

- A positions is chosen at random and the bit next to that position are reversed and mutated child is produced.



Mutation operation in Real-coded GA

Mutation Operation in GAs

Like different crossover techniques in different GAs there are many variations in mutation operations.

- **Binary Coded GA :**
  - Flipping
  - Interchanging
  - Reversing
- **Real-coded GA :**
  - Random mutation
  - Polynomial mutation
- **Order GA :**
- **Tree-encoded GA :**

Mutation in Real-coded GA : Random mutation

- Here, mutated solution is obtained from the original solution using the following rule.

$$P_{mutated} = P_{original} + (r - 0.5) \times \Delta$$

  Where r is a random number lying between 0.0 and 1.0 and Δ is the maximum value of the perturbation decided by the user.

- **Example :**

  $P_{original} = 15.6$

  $r = 0.7$

  $\Delta = 2.5$

  Then, $P_{mutated} = 15.6 + (0.7 - 0.5) \times 2.5 = 16.1$

Mutation in Real-coded GA : Polynomial mutation

It is a mutation operation based on the polynomial distribution. Following steps are involved.

1. Calculate a random number r lying between 0.0 and 1.0
2. Calculate the perturbation factor δ using the following rule

$$\delta = \begin{cases} (2r)^{\frac{1}{q+1}} - 1 & ,\text{if } r < 0.5 \\ 1 - [2(1-r)]^{\frac{1}{q+1}} & ,\text{if } r \geq 0.5 \end{cases}$$

   where q is a exponent (positive or negative value) decided by the user.

3. The mutated solution is then determined from the original solution as follows

$$P_{mutated} = P_{original} + \delta \times \Delta$$

   Where Δ is the user defined maximum perturbation allowed between the original and mutated value.

Mutation in Real-coded GA : Polynomial mutation

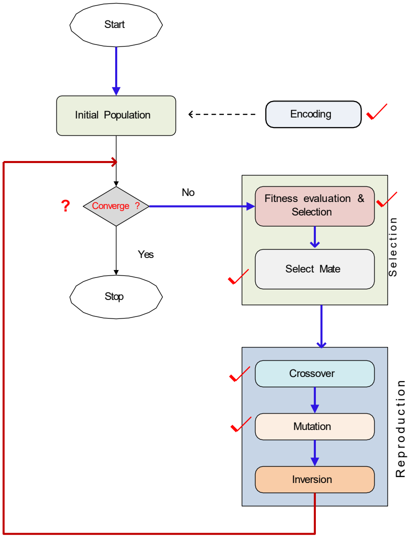
- **Example :**

  $P_{original} = 15.6, r = 0.7, q = 2, \Delta = 1.2$ then $P_{mutated} = ?$

  $\delta = 1 - [2(1-r)]^{\frac{1}{q+1}} = 0.1565$

  $P_{mutated} = 15.6 \times 0.1565 \times 1.2 = 15.7878$

Revisiting the flow in GA



Termination and/or convergence criteria

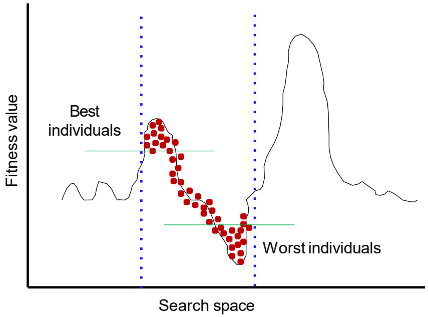
Each iteration should be tested with some convergence test.
Commonly known convergence test or termination conditions are :

1. A solution is found that satisfies the objective criteria.
2. Fixed number of generation is executed.
3. Allocated budget (such as computation time) reached.
4. The highest ranking solution fitness is reaching or has reached a plateau such that successive iterations no longer produce better result.
5. Manual inspection.
6. Combination of the above.

Fitness Scaling in GA

Issue with fitness values

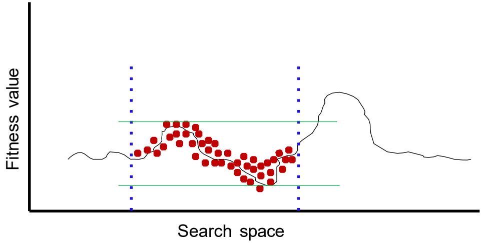
Let us look into a scenario, which is depicted in the following figure.



- Here, the fitness values are with wider range of values.
- It then highly favors the individuals with large fitness values and thus stuck at local optima/premature termination/inaccurate result.

Issue with fitness values

Now, let us look into another scenario, which is depicted in the following figure.



- Here, the fitness values are with narrower range of values.
- In this case, successive iterations will not show any improvement and hence stuck at local optima/premature termination/inaccurate result.

Summary of observations

It is observed that

- If fitness values are too far apart, then it will select several copies of the good individuals and many other worst individual will not be selected at all.
- This will tend to fill the entire population with very similar chromosomes and will limit the ability of the GA to explore large amount of the search space.
- If the fitness values are too close to each other, then the GA will tend to select one copy of each individual, consequently, it will not be guided by small fitness variations and search scope will be reduced.

Why fitness scaling?

- As a way out we can think for crossover or mutation (or both) with a higher fluctuation in the values of design parameter.
  - This leads to a chaos in searching.
  - May jump from one local optima to other.
  - Needs a higher number of iterations.
- As an alternative to the above, we can think for **fitness scaling** strategy.

Fitness scaling is used to scale the raw fitness values so that the GA sees a **reasonable** amount of difference in the scaled fitness values of the best versus worst individuals.

Approaches to fitness scaling

In fact, fitness scaling is a sort of discriminating operation in GA.

Few algorithms are known for fitness scaling:

- Linear scaling
- Sigma scaling
- Power law scaling

- **Note:**
  The fitness scaling is useful to avoid premature convergence, and slow finishing.

Linear scaling

Algorithm

- **Input :** $F = \{f_1, f_2 \cdots f_N\}$ is a set of raw fitness values of N individuals in a population (initial).
- **Output :** $F^r = \{f_1^r, f_2^r \cdots f_N^r\}$ is a set of fitness values after scaling

Steps :

1. Calculate the average fitness value
$$\bar{f} = \frac{\sum_{i=1}^{N} f_i}{N}$$
2. Find $f_{max} = MAX(F)$, Find the maximum value in the set F.
3. Find $f_{min} = MIN(F)$, Find the minimum value in the set F.

Linear scaling

4. Calculate the following,

$$a = \frac{\bar{f}}{f_{max} - \bar{f}},$$
$$b = \frac{\bar{f} * f_{min}}{f_{min} - \bar{f}}$$

5. For each $f_i \in F$ do

$$f_i^{\mathrm{r}} = a \times f_i + b$$
$$F^{\mathrm{r}} = F^{\mathrm{r}} \cup f_i^{\mathrm{r}}$$

   where F^{r} is initially empty.

6. End

Linear scaling

Note :

1. For better scaling it is desirable to have $\bar{f} = \bar{f}^{r}$
2. In order not to follow dominance by super individuals, the number of copies can be controlled with $f^{r}_{max} = C \times \bar{f}^{r}$ where $C = \frac{f_{max} - f_{min}}{\bar{f} - f_{min}}$

Sigma scaling

Algorithm

- **Input :** $F = \{f_1, f_2 \cdots f_N\}$ is a set of row fitness values of N individuals in a population.
- **Output :** $F^r = \{f_1^r, f_2^r \cdots f_N^r\}$ is a set of fitness values after scaling.

Steps :

1. Calculate the average fitness value
$$\bar{f} = \frac{\sum_{i=1}^{N} f_i}{N}$$
2. Determine reference worst-case fitness value f_w such that
$$f_w = \bar{f} + S * \sigma$$
Where $\sigma = STD(F)$, is the standard deviation of the fitness of population and

   S is a user defined factor called sigma scaling factor (Usually $1 \leq S \leq 5$)

Sigma scaling

3. Calculate f_i^r as follows

```
For each $f_i \in F$ do
    $f_i^r = f_w - f_i$, if $(f_w > f_i)$
Else $f_i^r = 0$
```

4. Discard all the individuals with fitness value 0
5. Stop

Note :

- Linear scaling (only) may yield some individuals with negative fitness value.
- Hence, Linear scaling is usually adopted after Sigma scaling to avoid the possibility of negative fitness individuals.

Power law scaling

In power law scaling, the scaled fitness value is given by

$$f_i^{\mathrm{r}} = f_i^k$$

where k is a problem dependent constant.