
# Mathematics for Computer Science Engineers

## Optimization in Statistics

## Genetic Algorithm

Team MCSE

Department of Computer Science and Engineering

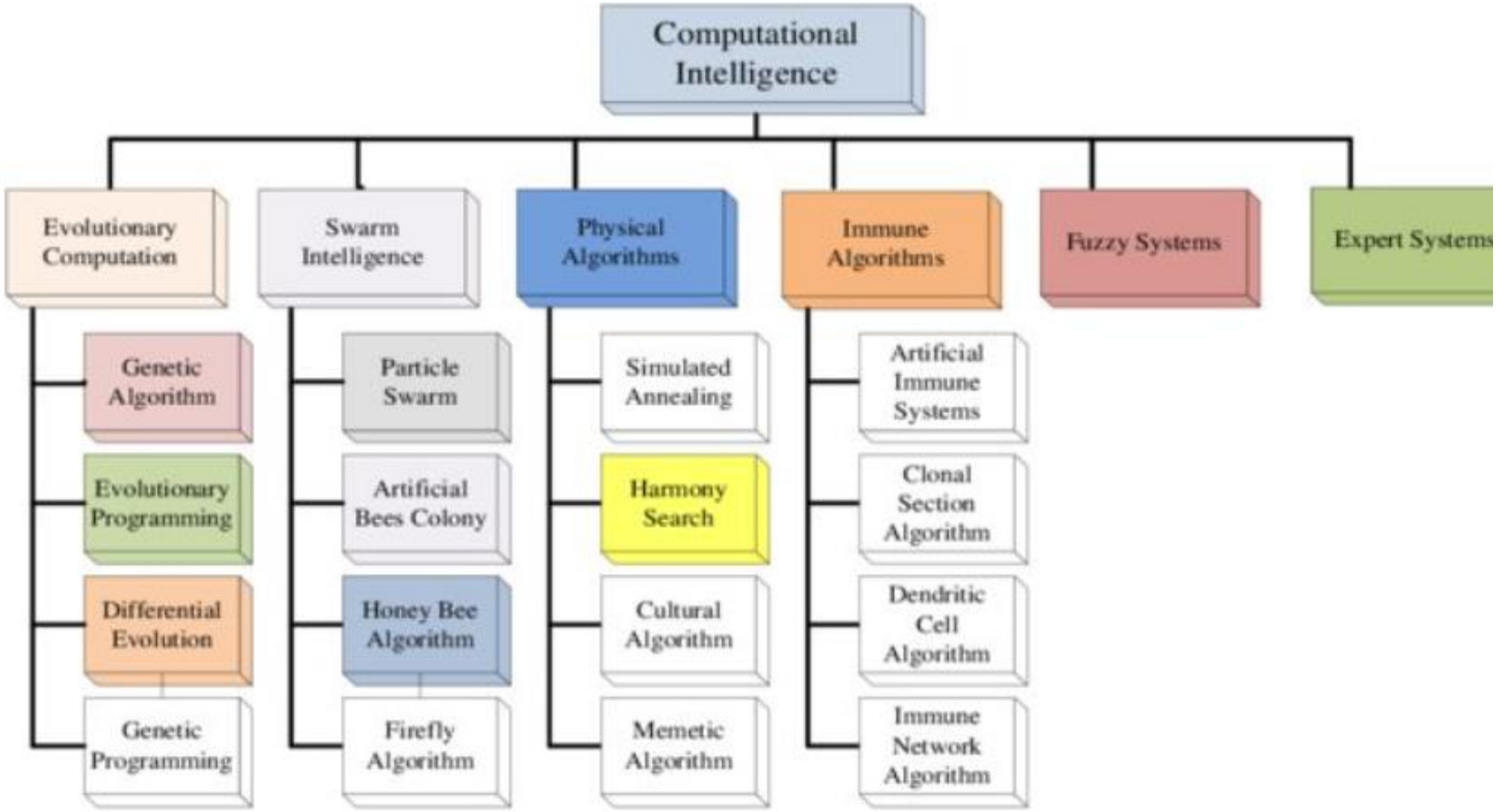
Acknowledgement : Dr. Rajanikanth K (Former Principal MSRIT, PhD(IISc),Board of Studies Member, PESU)
Dr. Uma D, Professor Dept of CSE
Prof. Preet Kanwal, Assoc Professor Dept of CSE

**Computational Intelligence (CI)** usually refers to the ability of a computer to learn a specific task from data or experimental observation.

Generally, **computational intelligence** is a set of **nature-inspired computational methodologies and approaches** to address complex real-world problems to which mathematical or traditional modelling can not be used for a few reasons:

- The processes might be too complex for mathematical reasoning,
- It might contain some uncertainties during the process, or
- The process might simply be stochastic in nature.

# Nature Inspired Algorithms for Optimization



**Evolutionary Computation:** Evolutionary computation is a sub-field of artificial intelligence (AI) and is used extensively in complex optimization problems having too many variables.

Alorithms running on the Evolutionary computation systems are called EA's.

Evolutionary algorithms (EAs) are a subset of optimization techniques inspired by the principles of natural evolution, such as selection, mutation, crossover (recombination), and survival of the fittest. They are used to solve optimization and search problems by iteratively improving a population of candidate solutions.

**Examples:** genetic algorithms, evolutionary programming, genetic programming and swarm intelligence models like ant colony optimization or particle swarm optimization.

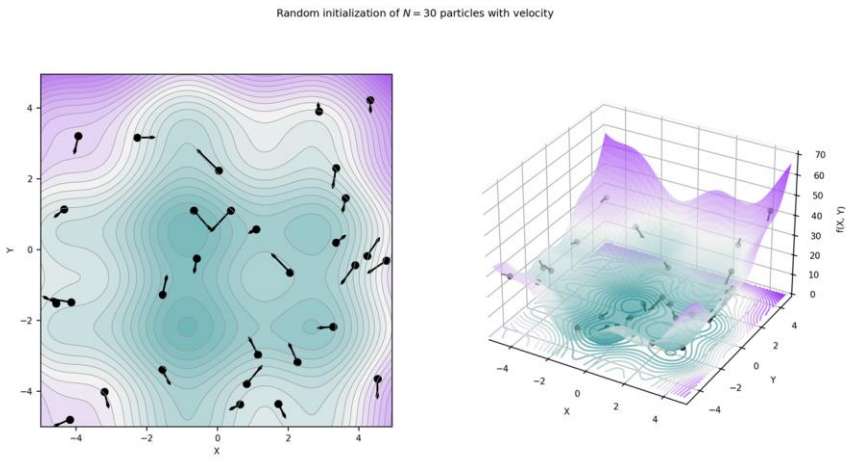
## Bio-Inspired Algorithms

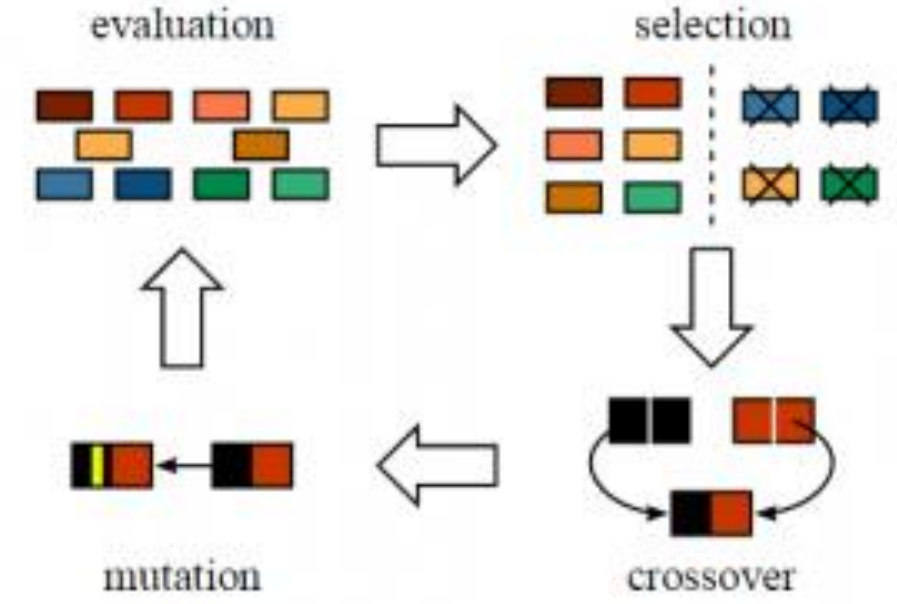
- **Bio-Inspired Computing (BIC)** refers to a multidisciplinary field of computing that develops algorithms, systems, and models inspired by biological processes and mechanisms.
- It seeks to mimic the problem-solving abilities, adaptability, and robustness of natural systems to address complex real-world challenges.
- These systems draw inspiration from various biological phenomena such as evolution, immune systems, neural activity, social behaviors, and cellular interactions.

# Mathematics for Computer Science Engineers

## Bio-Inspired Algorithms

- Some examples of bio-inspired algorithms are:
    a. Genetic algorithm
    b. Neural networks
    c. Particle Swarm Optimization



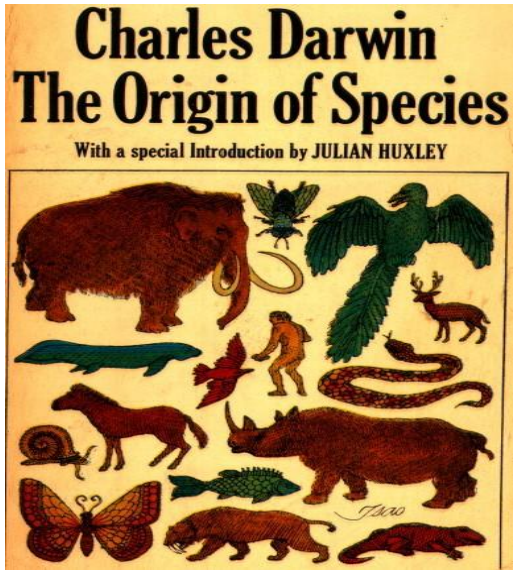



# Genetic Algorithms- Introduction

- **Genetic Algorithm** (GA) is a metaheuristic search-based optimization technique inspired by the principles of **Genetics and Natural Selection** (*Darwin's theory of evolution:* ***survival of the fittest)*****.**

- Idea was introduced by applying to problem solving by Professor John Holland in 1965.

- It is frequently used to find *optimal or near-optimal solutions* to difficult problems which otherwise would take a lifetime to solve.



## Introduction

GAs are inspired by natural genetics and selection, using principles like reproduction, crossover, and mutation to perform the optimization search. Unlike traditional methods, GAs have unique features that enhance their effectiveness:

1. GAs start with a population of points (trial solutions) rather than a single point. For *n decision* variables, the population size typically ranges from *2n* to *4n*. This broad search base reduces the risk of getting trapped in local optima.

1. GAs rely solely on the objective function values, avoiding the need for derivative information.

# PSO Example

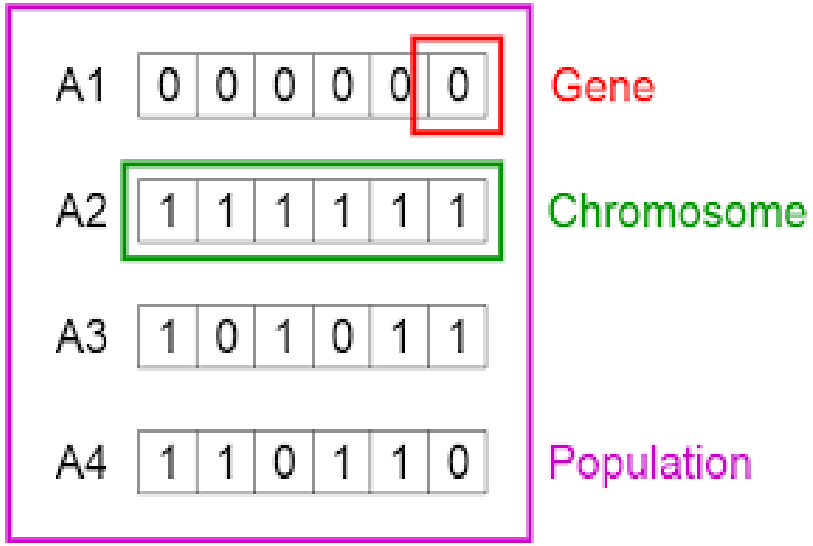
## Basic Terminologies:

**Population** – It is a subset of all the possible(encoded) solutions to the given problem.
Pool of individuals which allows the application of **genetic operators**.

**Chromosome / Individual** –A chromosome is one such solution to the given problem.
carrier of the genetic information(**Chromosome**).
It is characterized by its state in the search space, its **fitness** (objective function value).

**Gene** – A gene is **one element position** of a **chromosome**

**Allele** – It is the **value a gene** takes for a particular chromosome



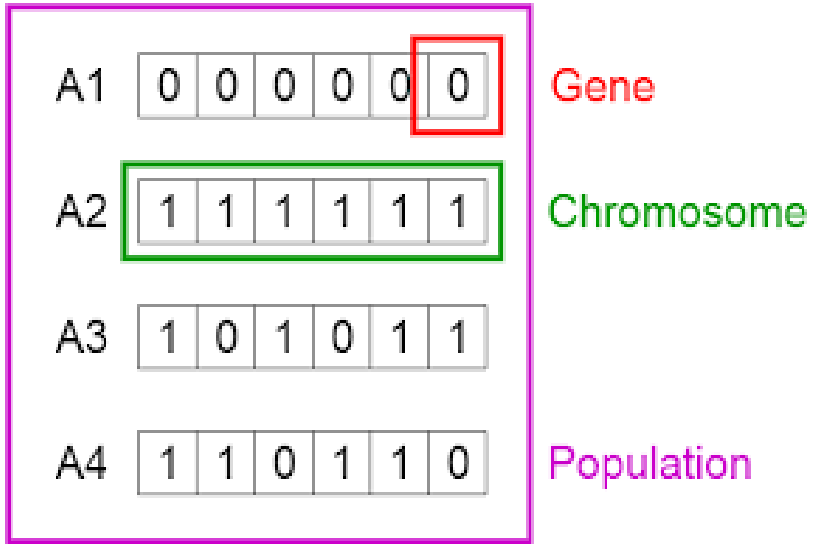
# PSO Example

## Basic Terminologies:

**Fitness Function** – Fitness function takes a solution and produces the suitability of the solution as the output.

- It gives us some idea of "**How good a solution is?**"

Note : The term "fitness function" is used for the **objective function**



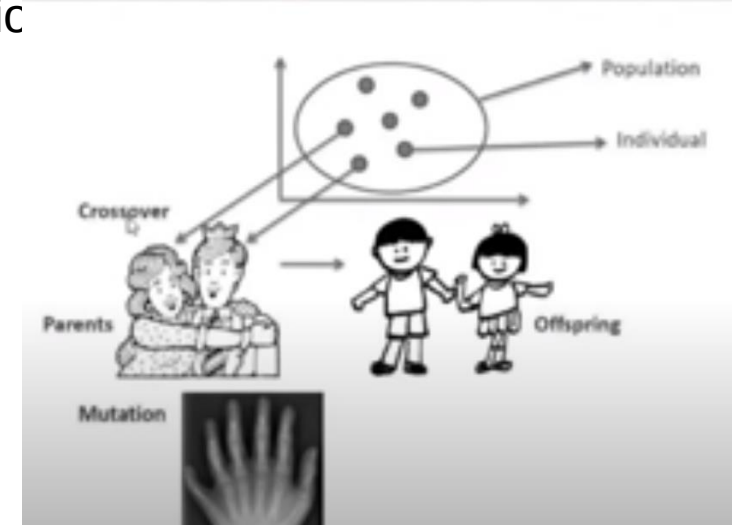
# Introduction to PSO

## Features of Evolutionary Algorithm / Genetic Operators:

- **Selection** – Roulette Wheel, Tournament, Steady State etc.
  Selection of chromosomes to participate in the next generation creation.
  Motivation is to **preserve the best** (make multiple copies) and eliminate the worst.

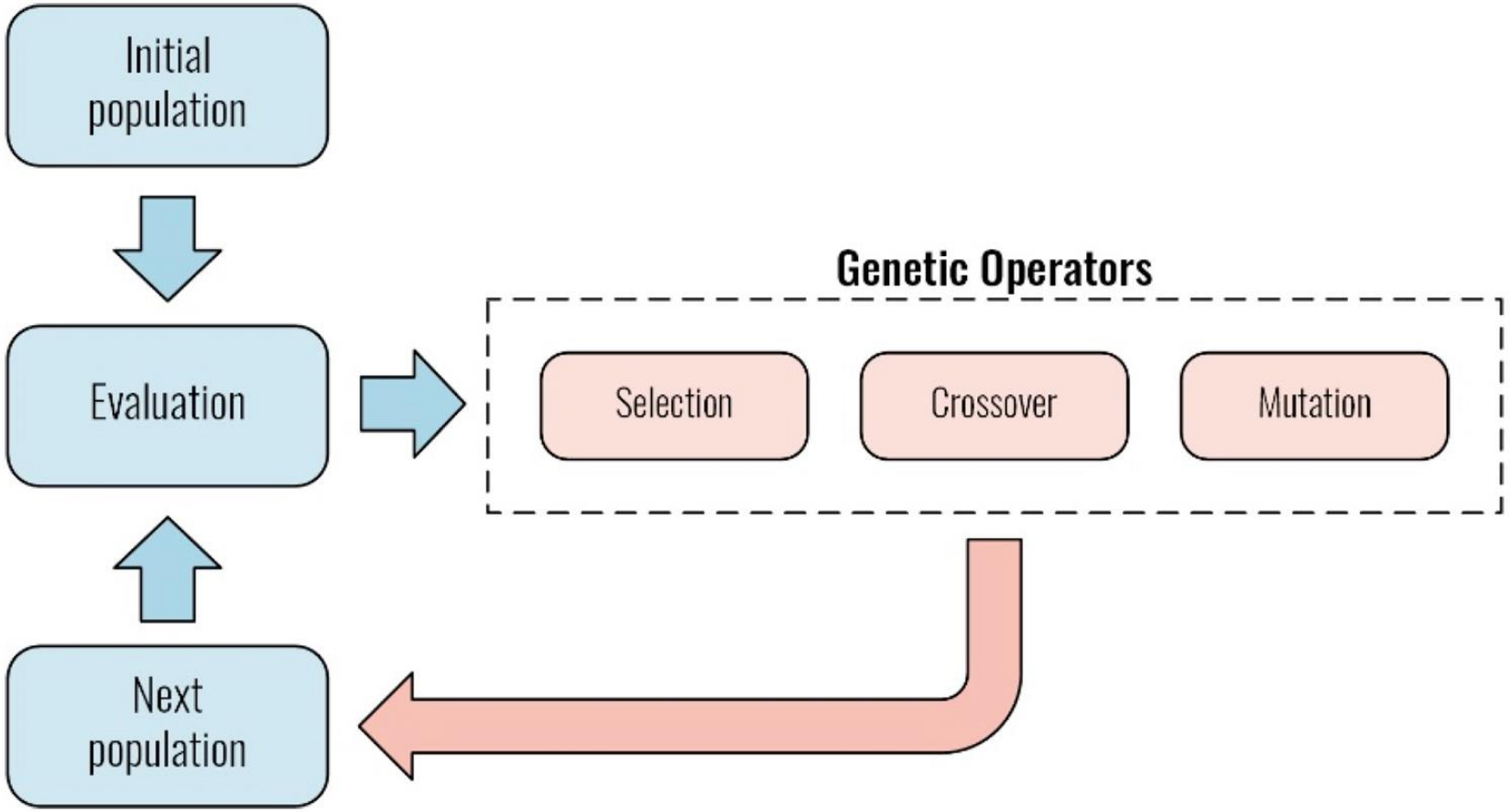
- **Crossover** – single point crossover, simulated binary crossover, linear crossover etc.
  Create **new solutions** by considering more than one individual
  Global search for new and hopefully better solutic

- **Mutation** – random mutation, polynomial mutation etc.
  keep **diversity** in the population
  010110 → 010100( bit wise mutation)



## Genetic Operators

**Genetic Operators** − These alter the genetic composition of the offspring.



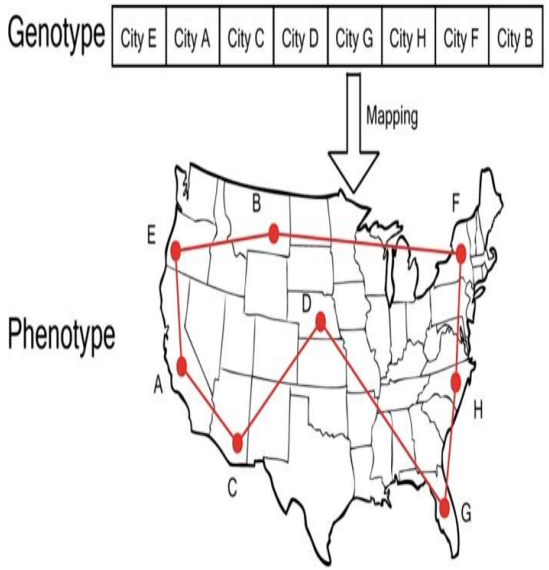
## PSO Example

**Basic Terminologies:**

**Genotype** − Genotype is the population in the computation space.
In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.

**Phenotype** − Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.

**Generation** – time unit of the Evolutionary Algorithm, an **iterative step** of an evolutionary algorithm.



# Basic Terminologies

## Basic Structure of GA:

- **Fitness Function**
- **Genetic Operators**
  - Selection (Survivor + Parent)
  - Crossover
  - Mutation
- **Termination Condition**

## Applications

**1. Traveling Salesman Problem (TSP)**

Find the shortest route for a salesman to visit n cities exactly once and return to the starting point.

**2. Function Optimization:** Maximize or minimize mathematical functions.

3. **Knapsack Problem:** Select items to maximize total value without exceeding a weight limit.

**4. Neural Network Weight Optimization**

Optimize the weights of a neural network to improve performance on a dataset.

**5. Robot Path Planning**

Find the optimal path for a robot to navigate an environment while avoiding obstacles.

**6. Scheduling Problem**

Optimize task scheduling to minimize completion time or maximize resource utilization.

**7. Evolving Game Strategies**

Optimize strategies for agents in games (e.g., chess, tic-tac-toe, etc.).

## Introduction

- **Decision variables in GAs are encoded as binary strings,** similar to genetic chromosomes, making them ideal for discrete and integer problems.
- For continuous variables, string length can be adjusted to achieve the desired precision.

- The objective function value of each decision vector serves as a "fitness" measure, guiding the evolutionary process.

- Each new generation of solutions is created by selecting parents and applying crossover, using randomization in a structured way to explore promising solutions while improving overall fitness.

GAs are not merely random searches; they leverage existing knowledge to systematically evolve solutions with higher fitness or improved objective values.

## Representation of Decision Variables

In GAs, decision variables are represented as binary strings of 0s and 1s.

For example, if a design variable $x_i$ is represented by a string of four bits as 0 1 0 1, its integer (decimal) equivalent is calculated as:

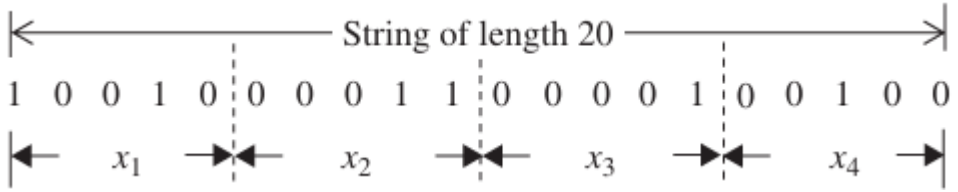
(1) $2^0 + (0)\,2^1 + (1)\,2^2 + (0)\,2^3 = 1 + 0 + 4 + 0 = 5$ .

(2) If each decision variable $x_i$, i = 1,2,...,n, is coded with a string of length $q$, a decision vector can be represented by a string of total length $nq$.

For instance, if each variable is represented by a string of length 5, then a string of length 20 will represent a decision vector with n = 4.

The following 20-bit binary string denotes the vector

$x_1 = 18$, $x_2 = 3$, $x_3 = 1$, $x_4 = 4$:



## Representation of Objective Function and Constraints

- Because GAs are based on nature's survival-of-the-fittest principle, they aim to maximize a function known as the fitness function.
- This makes GAs naturally suited for solving unconstrained maximization problems.
- The fitness function, *F(X)*, can be set equal to the objective function *f(X)* in an unconstrained maximization problem, such that *F(X) = f(X)*.

## Representation of Objective Function and Constraints

- For minimization problems, a common approach is to transform them into equivalent maximization problems before applying GAs.
- Typically, the fitness function is chosen to be nonnegative.
- A commonly used transformation to convert an unconstrained minimization problem to a fitness function is:

$$F(X) = 1/(1 + f(X))$$

## Representation of Objective Function and Constraints

This transformation does not change the location of the minimum of f(X) but effectively turns the minimization problem into a maximization one.

A general constrained minimization problem can be formulated as follows:

**Minimize f(X)**

**subject to:**

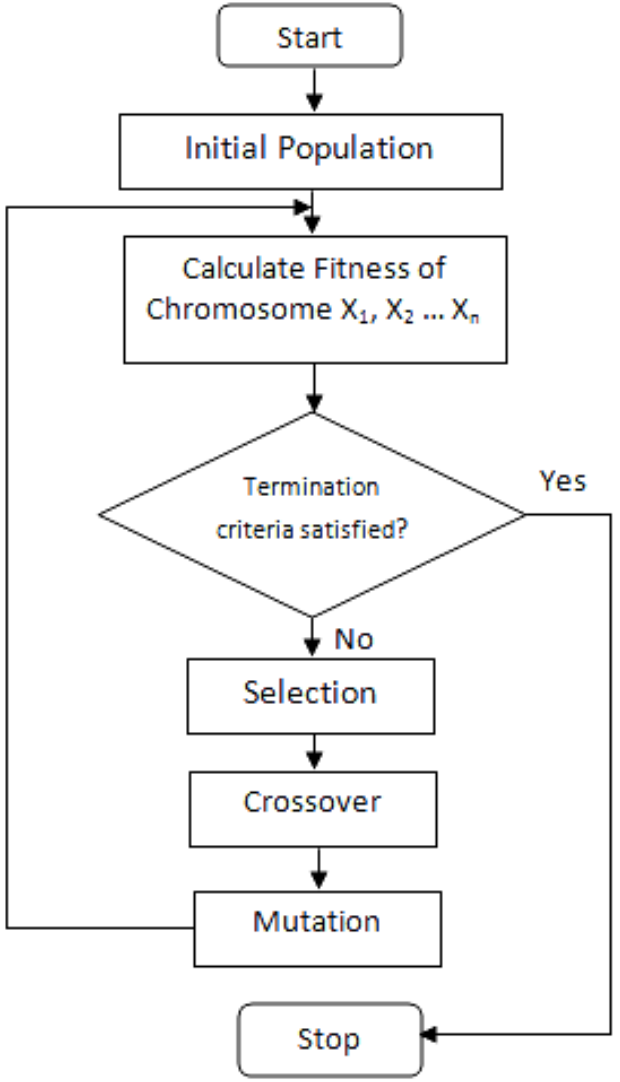
$g_i(X) <= 0, \quad i = 1,2,....,m$

$h_j(X) = 0, \quad j = 1,2,....,p$

## Genetic Algorithms – Basic Structure

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for crossover.

Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats.



## Outline of basic Genetic Algorithm

1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Create a new population:
    a. Selection: According to the selection method select 2 chromosomes from the population.
    b. Crossover: Perform crossover on the 2 chromosomes selected.
    c. Mutation: Perform mutation on the chromosomes obtained.
4. Replace: Replace the current population with the new population.
5. Test: Test whether the Termination condition is satisfied. If so, stop. If not, return the best solution in current population and go to Step 2.

Each iteration of this process is called generation.

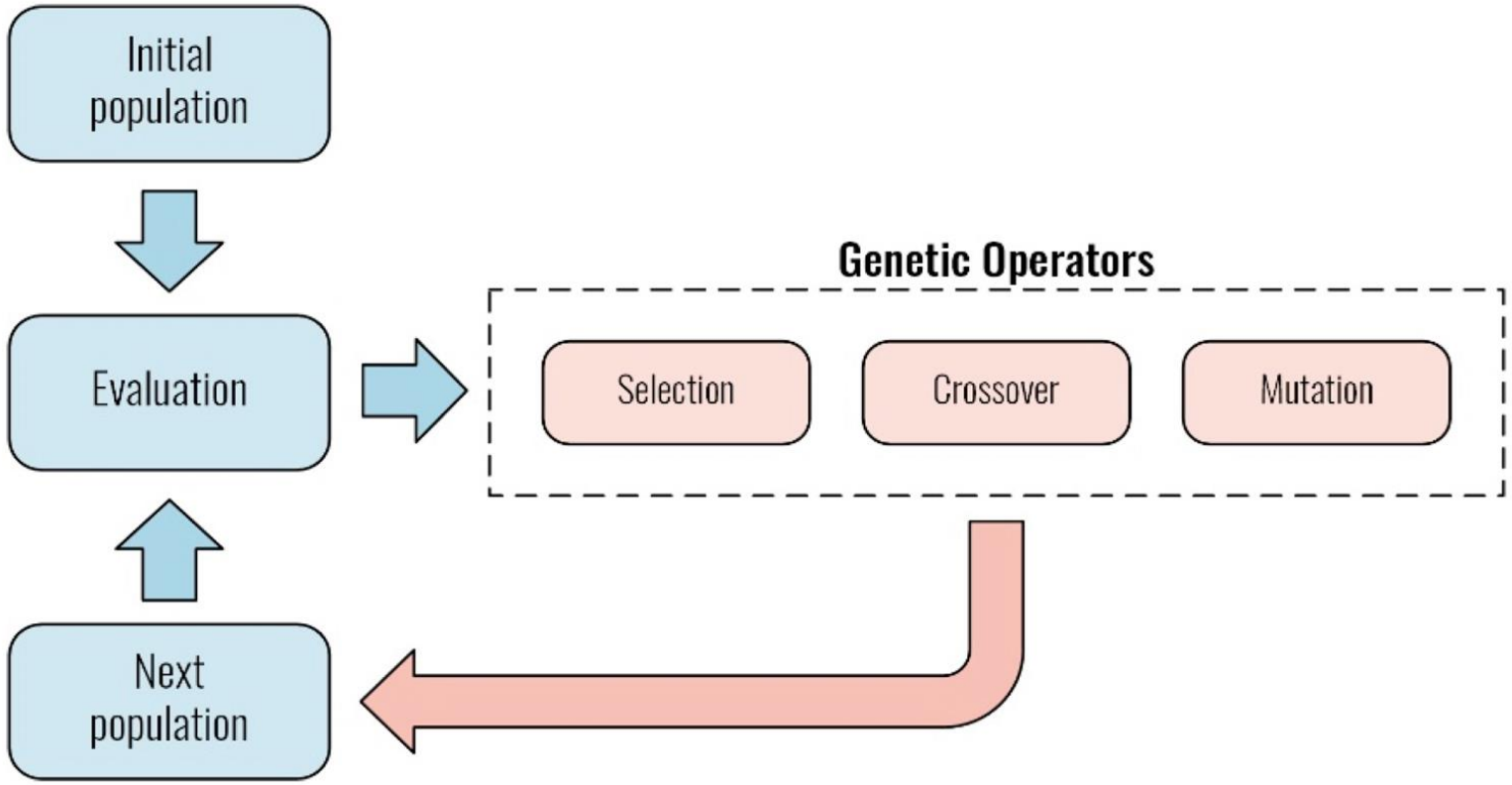
# Hyperparameters of Genetic Algorithm

There are various decisions/choices that a designer has to make based on the problem and domain knowledge when working with Genetic Algorithms.

- Encoding of Solution / Chromosome length
- Generation of Initial Population
- Population Size
- Fitness function
- Selection Methods (Elitism rate, Generation Gap)
- Crossover rate
- Crossover Technique
- Mutation rate
- Mutation Technique
- Termination Condition (Target value or Maximum number of iterations)

## Genetic Operators

**Genetic Operators** − These alter the genetic composition of the offspring.



## Genetic Operators

- In GAs, optimization begins with a random population of strings representing decision vectors, with a fixed population size.
- Each string is evaluated for its fitness value, and three key operations—reproduction, crossover, and mutation—are applied to generate a new population.
- This process of operations and fitness evaluation, called a generation, is repeated iteratively.

## Genetic Operators

If the convergence criterion is not met, the cycle continues through successive generations until convergence is achieved, at which point the process stops.

Reproduction is the first step in Genetic Algorithms, focusing on selecting "fit" strings (decisions) from the population to create a mating pool.

This process, also known as selection, favors above-average strings by using a probabilistic approach based on fitness.

## Genetic Operators

Each string is chosen with a probability proportional to its fitness,

$$p_i = \frac{F_i}{\sum_{j=1}^{n} F_j}; \quad i = 1, 2, \ldots, n$$

where $F_i$ is the fitness of the $i$th string.

A common method to implement this is the roulette wheel selection:

the wheel's circumference is divided into segments proportional to each string's fitness, and by "spinning" the wheel multiple times, a mating pool is created that reflects each string's fitness probability.

# Fitness proportionate Selection

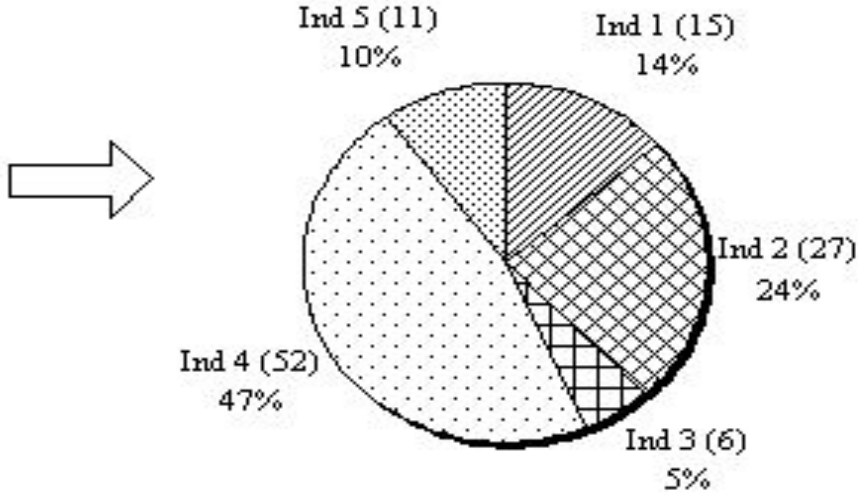
## Example 2

Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

A popular implementation of fitness proportionate selection is:

$$\frac{f_i}{\sum_{k=1}^{P} f_k}$$

| *Population* | *Fitness* |
|---|---|
| Individual 1 | 15 |
| Individual 2 | 27 |
| Individual 3 | 6 |
| Individual 4 | 52 |
| Individual 5 | 11 |




Randomly generated number = 21

Individual 2 is selected

## Roulette Wheel Selection

### Example 2

The population size is assumed to be 6, with the fitness values of the strings (1, 2, 3, 4, 5, and 6) given as 12, 4, 16, 8, 36, and 24, respectively. Since the fifth string (individual) has the highest fitness value, it is expected to be selected most often (36% of the time, probabilistically) when the roulette wheel is spun n times (n = 6 in this case).

This selection scheme, based on spinning the roulette wheel, can be implemented numerically as follows:

The probabilities of selecting different strings based on their fitness values are calculated. These probabilities are then used to determine the cumulative probability $P_i$ of string i being copied to the mating pool by summing the individual probabilities of strings 1 through i:

$$P_i = \sum_{j=1}^{i} p_j$$

## Roulette Wheel Selection

Thus, the roulette-wheel selection process can be implemented by associating the cumulative probability range $(P_{i-1} - P_i)$ to the *i-th string*. This approach generates the mating pool by defining selection ranges for each string based on cumulative probabilities.

Total Fitness=12+4+16+8+36+24=100

$$P(i) = \frac{\text{Fitness of string i}}{\text{Total Fitness}}$$

| String number $i$ | Fitness value $F_i$ | Probability of selecting string $i$ for the mating pool, $p_i$ | Cumulative probability value of string $i$, $P_i = \sum_{j=1}^{i} p_j$ | Range of cumulative probability of string $i$, $(P_{i-1}, P_i)$ |
|---|---|---|---|---|
| 1 | 12 | 0.12 | 0.12 | 0.00–0.12 |
| 2 | 4 | 0.04 | 0.16 | 0.12–0.16 |
| 3 | 16 | 0.16 | 0.32 | 0.16–0.32 |
| 4 | 8 | 0.08 | 0.40 | 0.32–0.40 |
| 5 | 36 | 0.36 | 0.76 | 0.40–0.76 |
| 6 | 24 | 0.24 | 1.00 | 0.76–1.00 |

## Roulette Wheel Selection

The results of the roulette wheel simulation for n=6 spins:

Random numbers generated for the spins:

- [0.305,0.729,0.612,0.493,0.667,0.302]

- Selected strings based on the random numbers:

[3,5,5,5,5,3]

The fifth string is selected most frequently, as expected based on its highest fitness value.

## Example 3

Consider six strings with fitness values 12, 4, 16, 8, 36, and 24, and the corresponding roulette wheel as shown before. We need to find the contribution of each string to the mating pool using the roulette-wheel selection process with the following 12 random numbers:
0.41, 0.65, 0.42, 0.80, 0.67, 0.39, 0.63, 0.53, 0.86, 0.88, 0.75, and 0.55.

Solution: If the given random numbers represent cumulative probabilities, the string numbers to be copied to the mating pool can be determined from the cumulative probability ranges listed in the last column of the Table. The results are as follows:

## Example 2

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Random number (cumulative probability of the string to be copied) | 0.41 | 0.65 | 0.42 | 0.80 | 0.67 | 0.39 | 0.63 | 0.53 | 0.86 | 0.88 | 0.75 | 0.55 |
| String number to be copied to the mating pool | 5 | 5 | 5 | 6 | 5 | 4 | 5 | 5 | 6 | 6 | 5 | 5 |

This indicates that the mating pool consists of:

- 1 copy of string 4
- 8 copies of string 5
- 3 copies of string 6

## Example

- This shows that less fit individuals (strings 1, 2, and 3) did not contribute to the next generation (or "died") because they could not contribute to the mating pool.
- String 4, although having a smaller fitness value, still contributed 1 copy based on the random selection process.

## Crossover

- Crossover is the next operation after reproduction in GAs.
- Its purpose is to create new strings (decisions) by exchanging portions of information between selected strings from the mating pool.

Different Crossover approaches

One Point Crossover
Two-Point Crossover
Multipoint Crossover
Uniform Crossover
Arithmetic/Logical Crossover
Whole arithmetic Recombination
Matrix Crossover

## Crossover

- In the single-point crossover method, two parent strings are randomly selected from the mating pool, and a crossover site is chosen randomly along the string length.
- The portions of the strings to the right of this site are swapped to generate two child strings.

**For example, if two parent strings are:**

$$\begin{array}{lll} \text{(Parent 1)} & \mathbf{X}_1 = \{0 \quad 1 \quad 0 \mid 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1\} \\ \text{(Parent 2)} & \mathbf{X}_2 = \{1 \quad 0 \quad 0 \mid 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0\} \end{array}$$

## Crossover

And the crossover site is at position 3, the resulting child strings will be:

$$\text{(Offspring 1)} \quad \mathbf{X}_3 = \{0 \quad 1 \quad 0 \mid 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0\}$$
$$\text{(Offspring 2)} \quad \mathbf{X}_4 = \{1 \quad 0 \quad 0 \mid 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1\}$$

The crossover process aims to combine the good characteristics of both parent strings, and ideally, the child strings should have better fitness values.

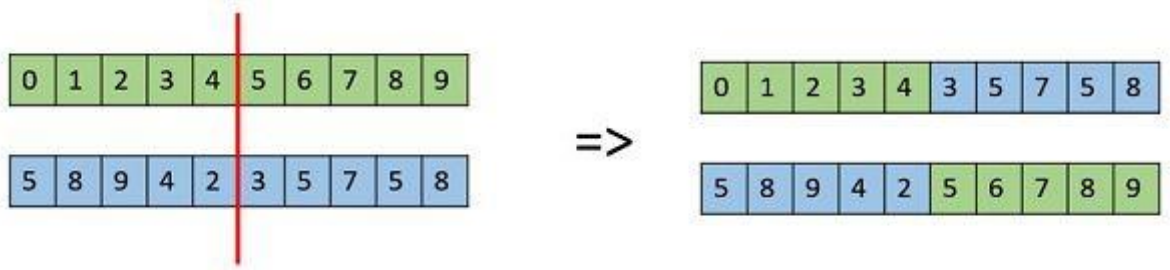
However, the effectiveness of the crossover depends on the random selection of the crossover site, which might result in children that are either better, worse, or equal in fitness compared to their parents.

**Results after crossover**

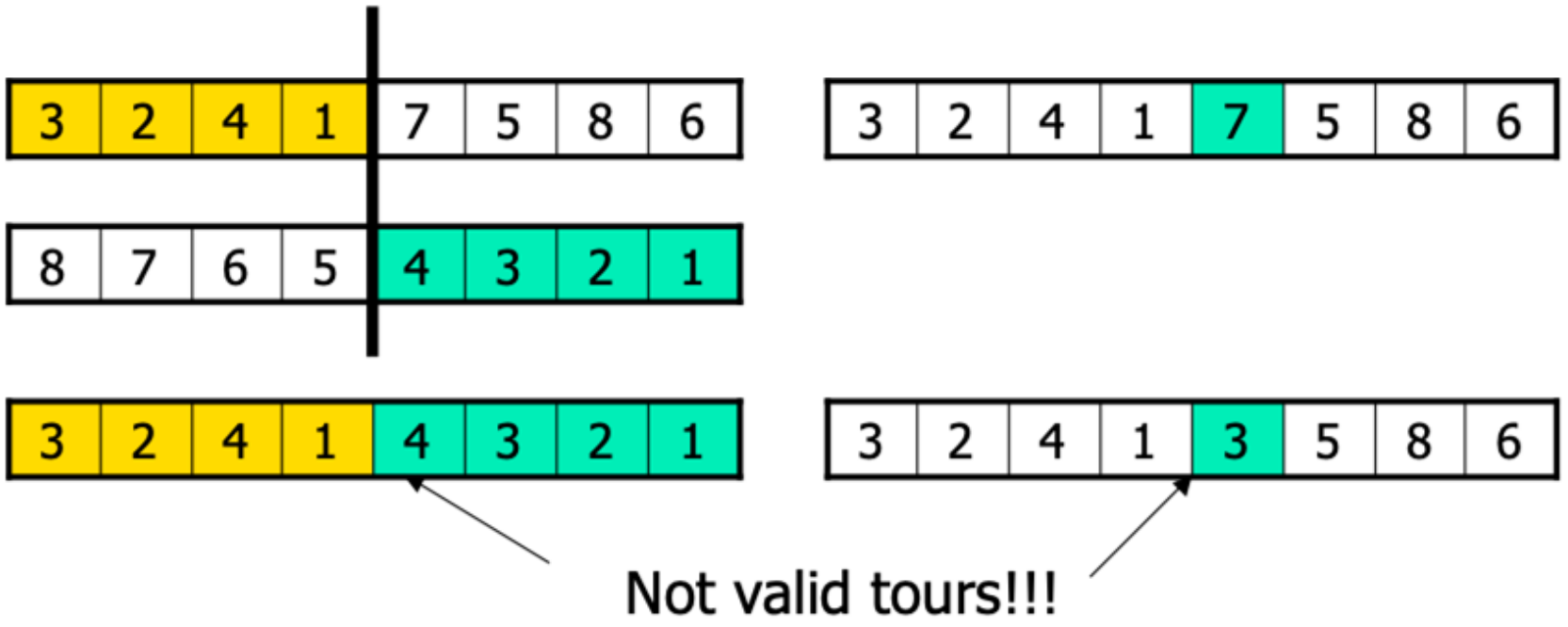
The crossover at position 3 would yield:

- $X_3 = \{0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\}$
- $X_4 = \{1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\}$

**Example**



## Problem with TSP (Permutation representation while Crossover)



## Mutation

- Mutation GA operator applied after crossover to introduce small, random changes in the offspring strings.
- This is done with a small mutation probability ($pm$), typically to prevent the algorithm from stagnating or losing important genetic material (Ex local minima).
- The mutation changes a binary digit (0 to 1 or 1 to 0) at a randomly selected site in the string.

There are two common mutation methods:

- Single-point mutation: A random mutation site is selected, and the binary digit at that site is flipped with probability $pm$.
- Bit-wise mutation: Each binary digit in the string is considered one at a time, and flipped with probability $pm$.

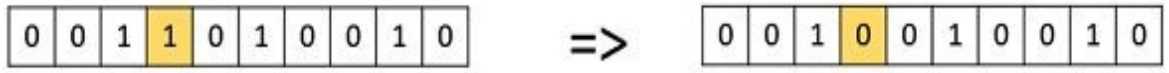
## Mutation

- The purpose of mutation is to explore the search space locally, maintain diversity, and prevent premature convergence to suboptimal solutions.
- For example, if all strings have the same bit at a particular position, mutation allows for introducing diversity by changing that bit.
- The mutation ensures that important features of the population are not lost and that the GA explores new solutions beyond what is found through reproduction and crossover alone.

## Mutation

Different Approaches for Mutation:

**Bit Flip**




**1's complement Mutation** Let C= [01101011] be a chromosome, then one's complement operator gives C'=[10010100].

**Random Resetting/ Uniform Mutation**

- Boundary Mutation
- Swap
- Scramble(shuffle)
- Inversion
- Insertion
- Heuristic Mutation

## Example

As an example, consider the following population of size n = 5 with a string length 10:

1 0 0 0 1 0 0 0 1 1

1 0 1 1 1 1 0 1 0 0

1 1 0 0 0 0 1 1 0 1

1 0 1 1 0 1 0 0 1 0

1 1 1 0 0 0 1 0 0 1

- Here, all five strings have a 1 in the position of the first bit.
- The true optimum solution of the problem requires a 0 as the first bit.
- The required 0 cannot be created by either the reproduction or the crossover operators.
- However, when the mutation operator is used, the binary number will be changed from 1 to 0 in the location of the first bit with a probability of $p_m$.

## Replacement and Termination

•Replace the old population with the new generation of individuals. This can be done partially (elitism) or fully.

•Example: Keep the top 2 individuals from the current population and replace the rest with new offspring.

**Termination**

•Repeat steps 2-6 until a stopping criterion is met:

- A maximum number of generations.
- A solution meets a predefined fitness threshold.

•Example: If after 50 generations the best solution is sufficiently close to the global minimum of f(x), stop.

## Example

To solve the optimization problem $f(x)=x^2$ using a genetic algorithm, let's break it down step-by-step with an example:

Solution

We want to maximize (x) = x^2.

Assume: x is an integer within a range, e.g., $0 \leq x \leq 31$.

•Represent x using binary strings (e.g., 5→00101).

Step 1:

**Initialization**

Generate an initial population of binary strings randomly. Assume a population size of 4:

Population: {00101,11001,00011,10100}

These correspond to: x={5,25,3,20}

Fitness values: f(x)={25,625,9,400}

## Example

**Step 2: Selection**

Use a method like **roulette wheel selection** to select parents. The probability of selection is proportional to fitness:

Probabilities: {25/1059,625/1059,9/1059,400/1059}

Higher fitness (e.g., 625) will have a higher chance of being selected.

**Step 3: Crossover**

Perform a **single-point crossover** on the selected parents.

Assume:

- Parent 1: 11001 (x=25)
- Parent 2: 10100 (x=20)

Choose a crossover point (e.g., after the 3rd bit):

- Offspring 1: 11000 (x=24)
- Offspring 2: 10101 (x=21)

## Example

**Step 4: Mutation**

Introduce small random changes by flipping a bit(Apply change to both the offsprings).

For example:

- Offspring 1: 11000→11010(x=26)
- Offspring 2: 10101→10111 (x=23)

**Note: Bit change can be applied to only one offspring**

**Step 5: Replacement**

Replace the old population with the new offspring. The new population is:

{11010,10111,00011,10100} ----------Random replacement

Fitness values are recalculated: f(x)={676,529,9,400}

**Step 6: Termination**

Repeat steps 2–5 until:

- A maximum number of generations is reached.
- The fitness value stops improving significantly.

## Example

**Example Conclusion**

After several generations, the algorithm may converge to the optimal solution:

- $x=31$, $f(x)=31^2=961$
- This shows that the genetic algorithm can effectively maximize f(x)=x^2

## More Example

1. Minimize $f(x,y)=x^2+y^2$
2. We are given N items where each item has some weight and profit associated with it. We are also given a bag with capacity W, [i.e., the bag can hold at most **W** weight in it]. The target is to put the items into the bag such that the sum of profits associated with them is the maximum possible.
3. Find the shortest route for a salesman to visit n cities exactly once and return to the starting point.A salesman needs to visit 5 cities (A, B, C, D, E), and the distance between each pair of cities is given in the following table:

| City | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 10 | 15 | 20 | 25 |
| B | 10 | 0 | 35 | 25 | 30 |
| C | 15 | 35 | 0 | 30 | 20 |
| D | 20 | 25 | 30 | 0 | 15 |
| E | 25 | 30 | 20 | 15 | 0 |


# THANK YOU

## Mathematics for Computer Science Engineers