MACHINE INTELLIGENCE

Introduction to Genetic Algorithms

Dr. Arti Arya
Department of Computer Science and Engineering

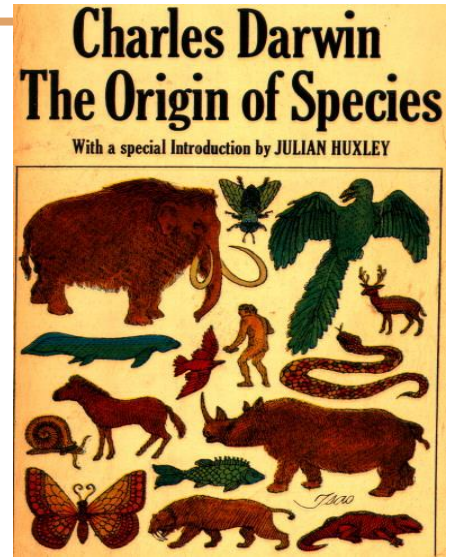
MACHINE INTELLIGENCE

Introduction to Genetic Algorithms

Dr. Arti Arya
Department of Computer Science and Engineering

MACHINE INTELLIGENCE

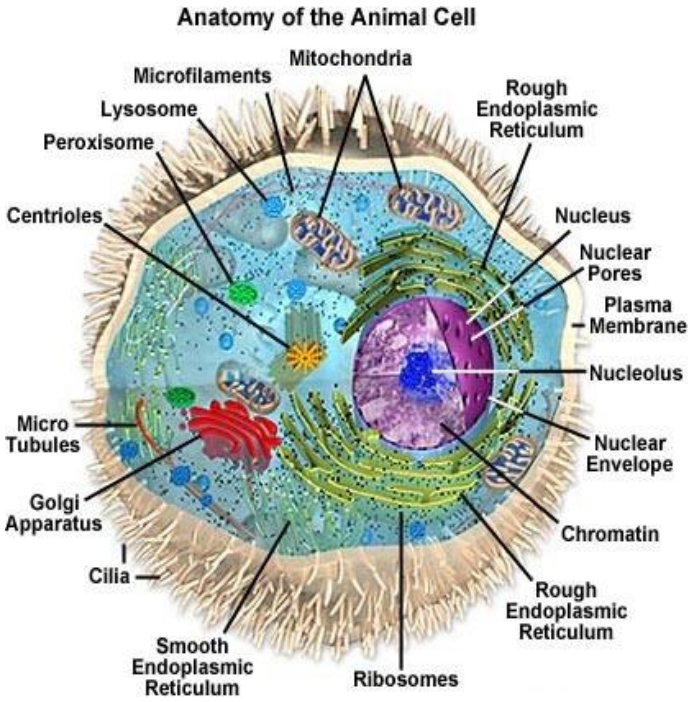
Genetic Algorithms- Introduction

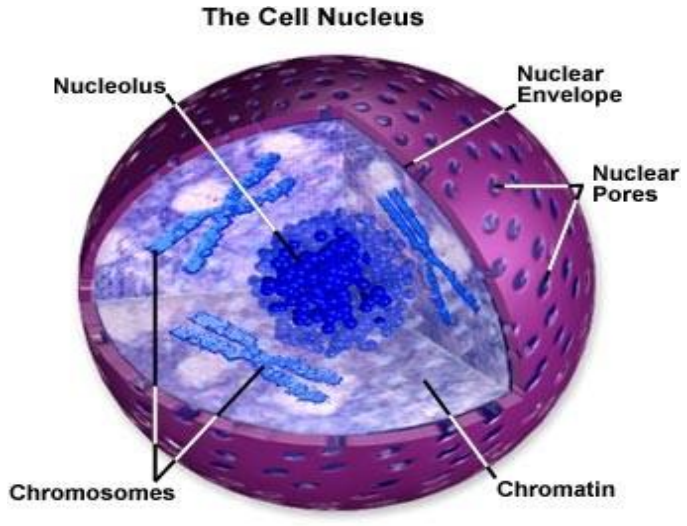


- Genetic Algorithm (GA) is a metaheuristic search-based optimization technique inspired by the principles of **Genetics and Natural Selection** *(Darwin's theory of evolution: survival of the fittest)*.
- Idea was introduced by applying to problem solving by Professor John Holland in 1965.
- It is frequently used to find *optimal or near-optimal solutions* to difficult problems which otherwise would take a lifetime to solve.
- Multiple points in the solution space are simultaneously considered for optimality, ie global perspective of the solution space. This avoids local optima.
- For detailed information you can further refer to [2].

Genetic Algorithms- Biological Background

- The nucleus contains the genetic information.





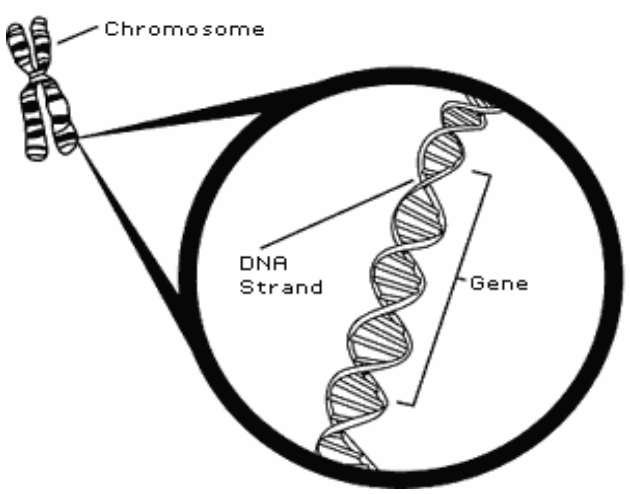


Figure from " Soft and Evolutionary Computing" by N P Padhy.

MACHINE INTELLIGENCE

Connection Of GA with Optimization

- The set of all possible solutions or values of the problem, make up the search space.
- This search space has a point or a set of points which gives the optimal solution.
- The aim of optimization is to find that point or set of points in the search space and *so is the objective of GAs.*
- GA use only objective function information, not derivatives or other auxiliary knowledge.
- Thus GA can deal with the non-smooth, non-continuous and non-differentiable functions which actually exist in a practical optimization problem.

Genetic Algorithms- Motivation

- Because Genetic Algorithms tend to find globally optimal solutions[1], which makes genetic algorithms attractive for solving optimization problems.

- **Need of GAs**

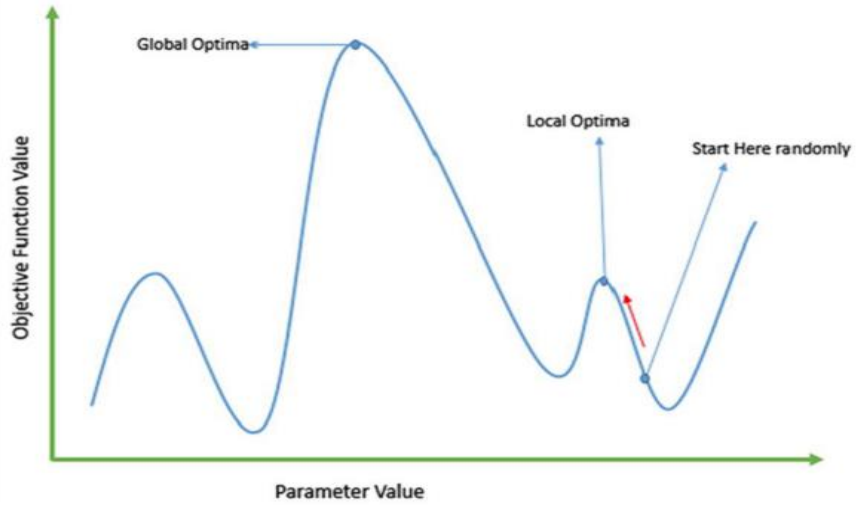
1. Solving Difficult Problems

  - In computer science, there is a large set of problems, which are **NP-Hard**. For solving such problems, even the most powerful computing systems take a very long time (even years!) to solve that problem.

Genetic Algorithms- Motivation

- **Failure of Gradient Based Methods**

- Traditional techniques efficient and works very well for single-peaked (unimodal) objective functions like the cost function in linear regression.
- More complex problems having multi-peaked (multimodal)objective functions come across, which causes such methods to fail, as they suffer from an inherent tendency of getting stuck at the local optima.



Genetic Algorithms- Motivation

- **Getting a Good Solution Fast**

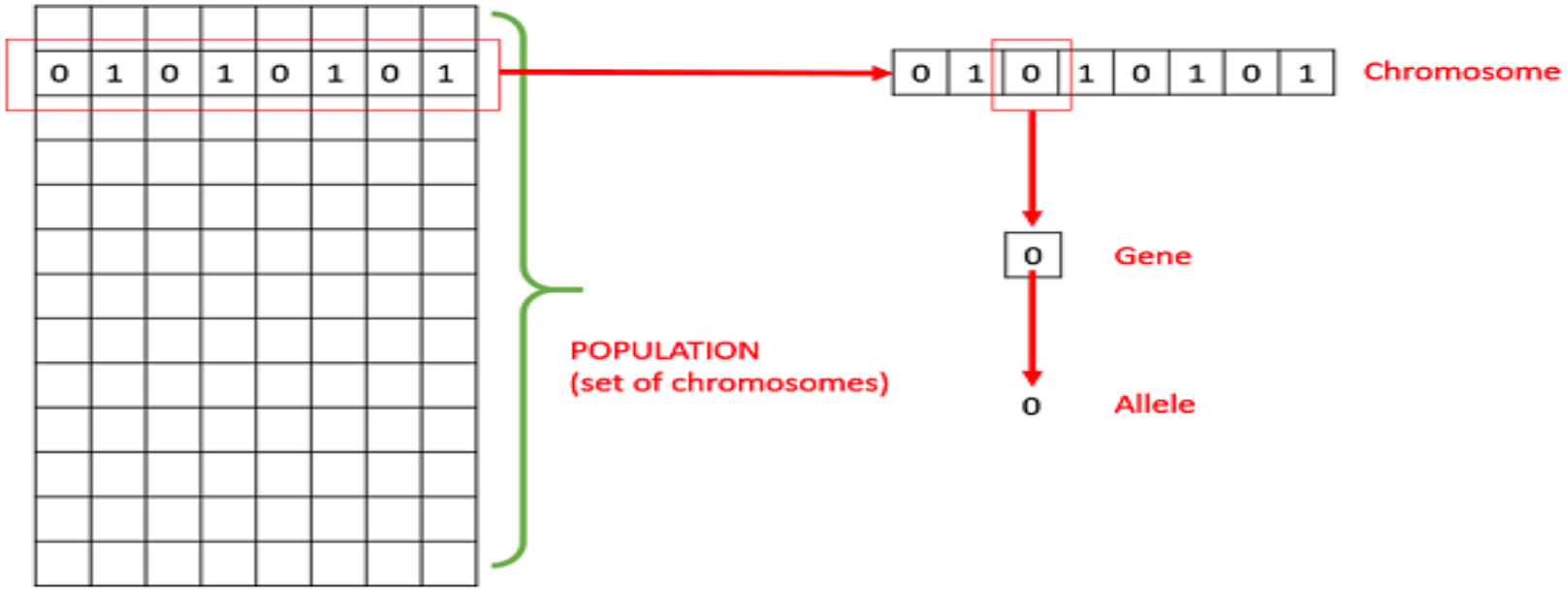
Some difficult problems like the Travelling Salesperson Problem (TSP), have real-world applications like path finding and VLSI Design.

MACHINE INTELLIGENCE

GA- Basic Terminologies

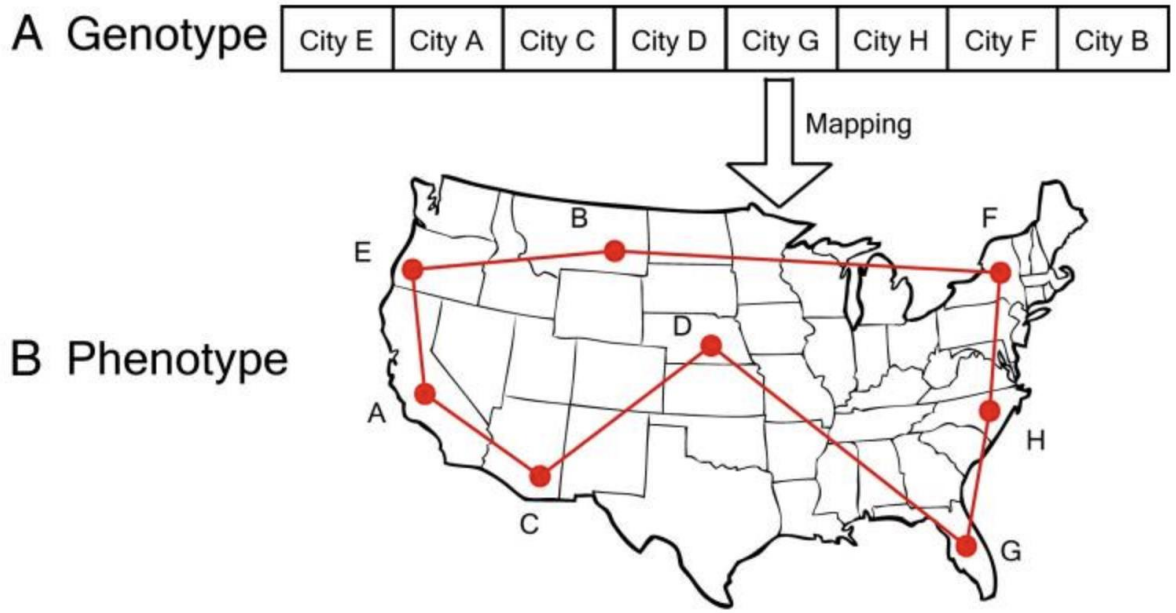
- **Population** – Subset of all the possible (encoded) solutions.
- **Chromosomes** – A chromosome is one such solution to the given problem.

- **Gene** – A gene is one element position of a chromosome.
- **Allele** – It is the value a gene takes for a particular chromosome.



GA- Basic Terminologies

- **Genotype** – The set of genes representing the chromosome in computation space.
- **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.



GA- Basic Terminologies

Decoding and Encoding- Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space.

- **Fitness Function** – A fitness function takes the solution as input and produces the suitability of the solution as the output.

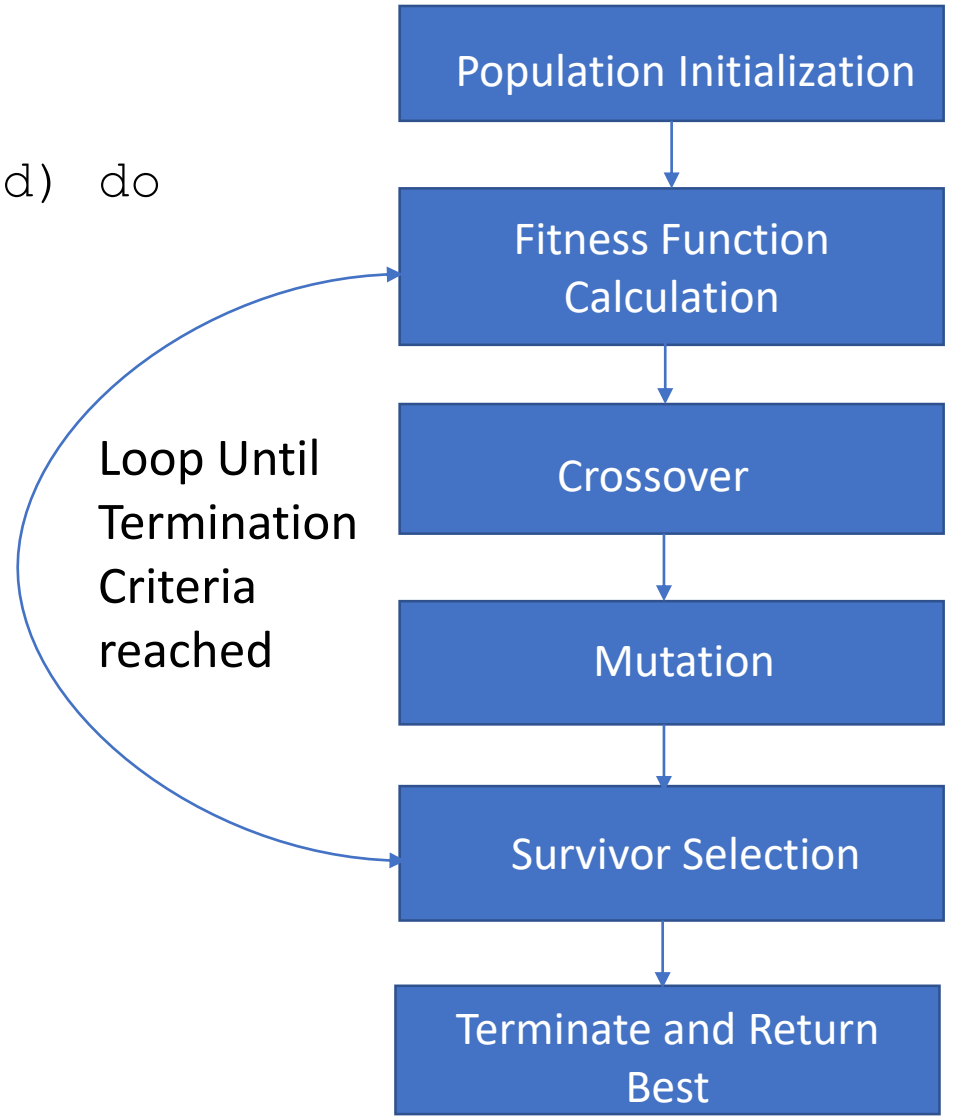
- **Genetic Operators** – These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

MACHINE INTELLIGENCE

GA- A generalized pseudo-code

```
GA()
        initialize population
        find fitness of population

while (termination criteria is reached) do
        parent selection
        crossover with probability pc
        mutation with probability pm
        decode and fitness calculation
        survivor selection
        find best
return best
```



Steps in GA……

- In GAs,
  - A pool or a population of possible solutions to the given problem is considered.
  - These solutions then undergo recombination and mutation (like in natural genetics), producing new offsprings.
  - And the process is repeated over various generations.
  - Each individual (or candidate solution) is assigned a fitness value and the fitter individuals will get a higher chance to mate and yield more “fitter” individuals.
  - In this way solutions “evolving” over generations, till a stopping criterion is met.

Limitations of Genetic algorithms

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- Convergence may not happen in some cases

Population Models

Steady State

In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as **Incremental GA**.

Generational

In a generational model, we generate ‘n’ off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

Genetic Representation(Encoding)

- *Encoding* is a process of representing chromosomes and thus genes.

- GAs algorithms do not deal with the solutions directly but with encoded representation of it.

- Chromosomal representation of a problem can be done by any of the following ways:
    - Binary string representation
    - Real number coding
    - Integer coding
    - Permutation Representation
    - Random key Representation
    - Representing genes in arrays, trees, lists and other objects

MACHINE INTELLIGENCE

Hypotheses Representation in GA

- Most of the time hypotheses in GAs are represented by bit strings.
- Bit strings are used for encoding bcoz it becomes easy to use genetic operators like crossover and mutation.
- Just as a simple example how bit strings would be used:
- Consider the classical example of " Play Tennis" where Outlook takes 3 values 'rainy', 'sunny' and 'overcast'.
- Say Outlook attribute is represented by a binary string of length 3 then
  - 100 represents Outlook has value "rainy".
  - 110 represents Outlook can acquire "rainy or sunny".

Genetic Representation-Examples

Binary Encoding

The equivalent value for any n-bit string can be obtained by

$$X_i = X_i^l + \frac{X_i^u - X_i^l}{(2^{n_i} - 1)} \times (decoded\ value\ of\ string)$$

Each variable has both the upper and lower limit which can be put in the form as

$$X_i^l \le X_i \le X_i^u$$

An 'n'-bit string can be represented by an integer between 0 to 2^n - 1, which consists of 2^n integers

n = 4, X_i^l=4 , X_i^u=25, $string$ =1010

Decoded value of string = 1010 = 2^3 x 1 + 2^2 x 0 + 2^1 x 1 + 2^0 x 0 = 10

Genetic Representation-Examples

The decoded value of 4 bit string is

$$X_i = 4 + \frac{21}{15} * 10 = 18$$

If 'p' is the precision required for a continuous variable then the string length 'S_l' should be equal to

$$S_i = \log_2 \left[\frac{X^u - X^l}{p} \right]$$

Example : Knapsack problem [4]

The problem: There are things with given value and size. The knapsack has a given capacity. Select things to maximize the value of things in knapsack, but do not extend knapsack capacity.

Encoding: Each bit says, if the corresponding thing is in knapsack.

Genetic Representation

Real Valued Representation

For problems where we want to define the genes using *continuous* rather than discrete variables, the real valued representation is the most natural[3]. The precision of these real valued or floating point numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

Integer Representation

For discrete valued genes, we cannot always limit the *solution space to binary 'yes' or 'no'.* For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{1,2,3,4}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1

Genetic Representation

Permutation representation (*Path Representation* or *Order representation)*

In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.
A classic example of this representation is the travelling salesman problem (TSP).

In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

For example, a tour of 8-city Travelling Salesman Problem(TSP)

3-5-8-1-4-2-6-7 can be represented as [3 5 8 1 4 2 6 7]

Genetic Representation

Random keys representation

Solution is encoded with random numbers from (0, 1) - 8 city TSP

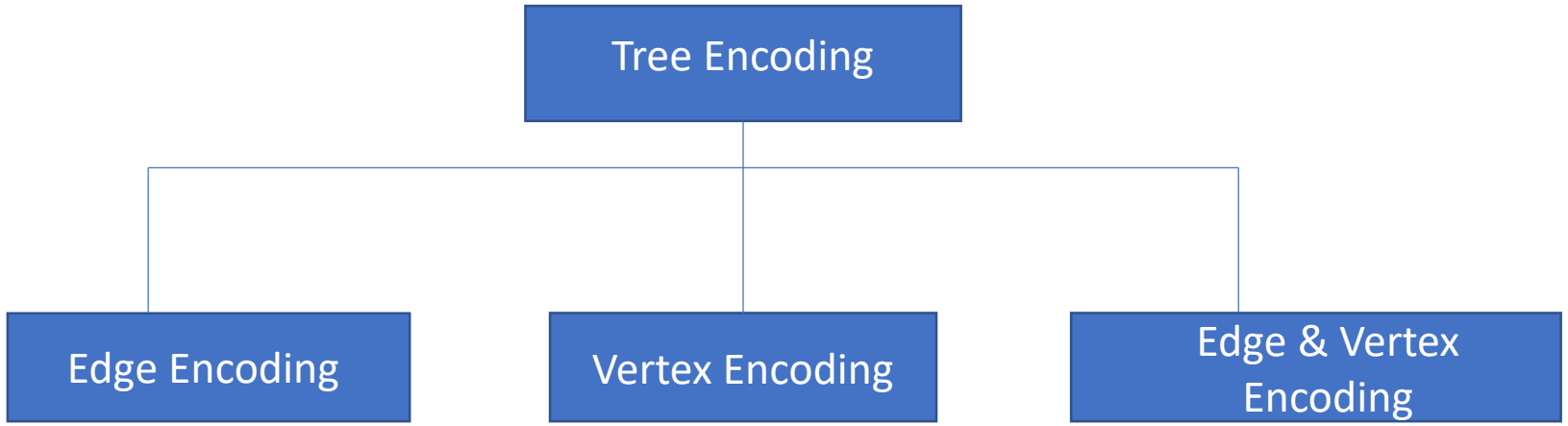
[0.45 0.68 0.91 0.11 0.62 0.34 0.74 0.89] can be represented as 3-5-8-1-4-2-6-7

(machine scheduling- resource allocation- vehicle routing-quadratic assignment problem)

MACHINE INTELLIGENCE

Genetic Representation-Examples

Tree Representation



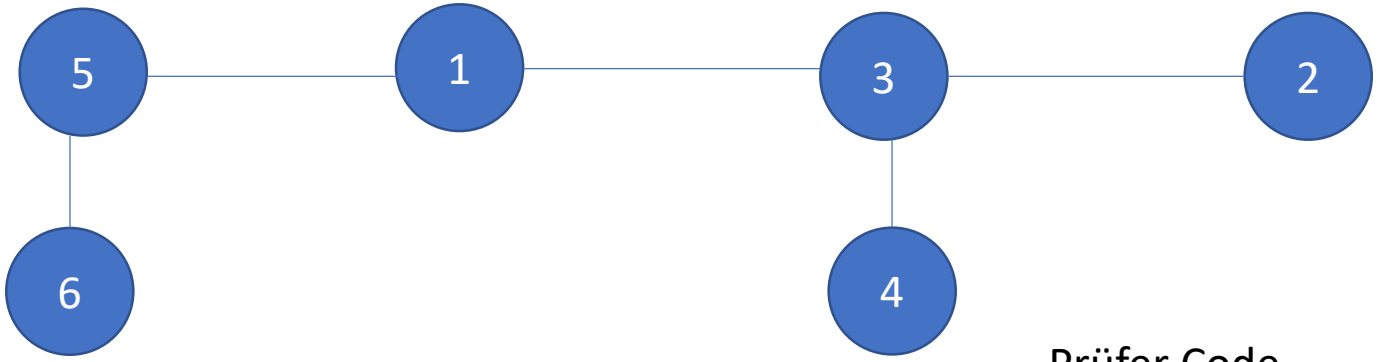
For GA's, one method of encoding input is through Tree Representation i.e. a chromosome may be represented through a tree.

MACHINE INTELLIGENCE

Genetic Representation-Examples

Vertex Encoding

Prufer's Code: Represents a tree in a particular way.



Prüfer Code

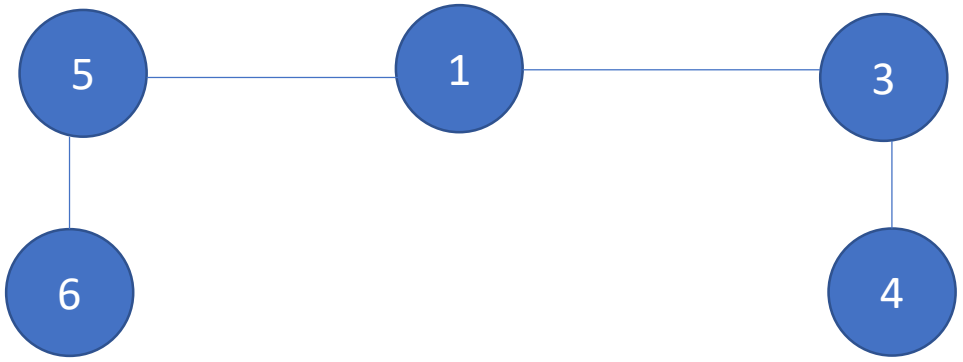
Generate Prüfer Code for the above tree.

The length of Prüfer's Code is given by **n-2,** where n is number of nodes/vertices in the tree.

Step 1: Look for the leaf nodes with **the smallest label**. So, its **2** and edge incident from **node 2 is to node 3**. So **remove node 2** from the tree and **add 3** at the first position in the Prüfer code.

MACHINE INTELLIGENCE

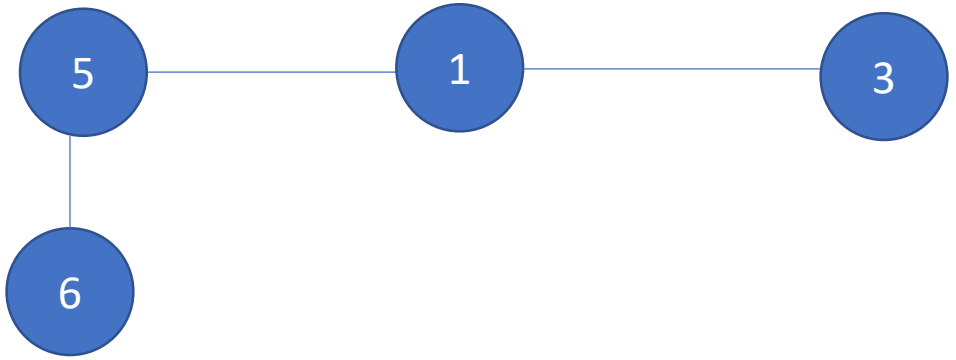
Genetic Representation-Examples



Prüfer Code

3	3		

Step 2: Now look for the leaf nodes with the **smallest label** again and **its node 4** and **edge incident from node 4 is to node 3**. So **remove 4** from the tree and add 3 at the second position in the Prüfer code.



Continuing like this till just one edge is left and we will have Prüfer's Code of length 4.

So,

3	3	1	5

is the final Prüfer's Code corresponding to given tree.

MACHINE INTELLIGENCE

Representing Hypotheses

- Hypothesis in GAs are often represented by bit strings, which are easily manipulated by crossover and mutation.

- Consider sets of if-then rules that can easily be represented by bit strings, by choosing an encoding of rules that allocate specific substrings for each rule precondition and postcondition.

MACHINE INTELLIGENCE

Representing Hypotheses- Example

- Attribute: **Outlook=Sunny, Rainy, Overcast**

- Use a bit string of length 3, where each position corresponds to one of the values. Placing a 1 in some position indicates that the attribute is allowed to take on the corresponding value.

- So 001 represents **Outlook = Overcast** and 011 represents **Outlook = Overcast or Rainy**

Id code	outlook	temp	humidity	windy	play
a	Sunny	high	high	strong	no
b	s	h	h	weak	n
c	overcast	h	h	s	y
d	Rainy	mild	h	s	y
e	r	cool	normal	s	y

More Representations

- Conjunctions of constraints can be represented by concatenation.

  Eg. **011 10** represents **Outlook = Rainy or Overcast and Wind = Strong**

- Postconditions can be represented in the same way **111 10 10** represents **If Wind = Strong then PlayTennis = Yes**. Notice that **111** represents the “don’t care” condition on Outlook

- Sets of rules can be represented by concatenating single rules, but may not be fixed length!

- Some GAs represent hypothesis as symbolic descriptions rather than bit strings.

Genetic Algorithms - Fitness Function

- The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how "fit" or how "good" the solution is with respect to the problem in consideration.

- A fitness function should process the following characteristics:

✓ The fitness function should be sufficiently fast to compute.

✓ It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

Genetic Algorithms - Parent Selection

- Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

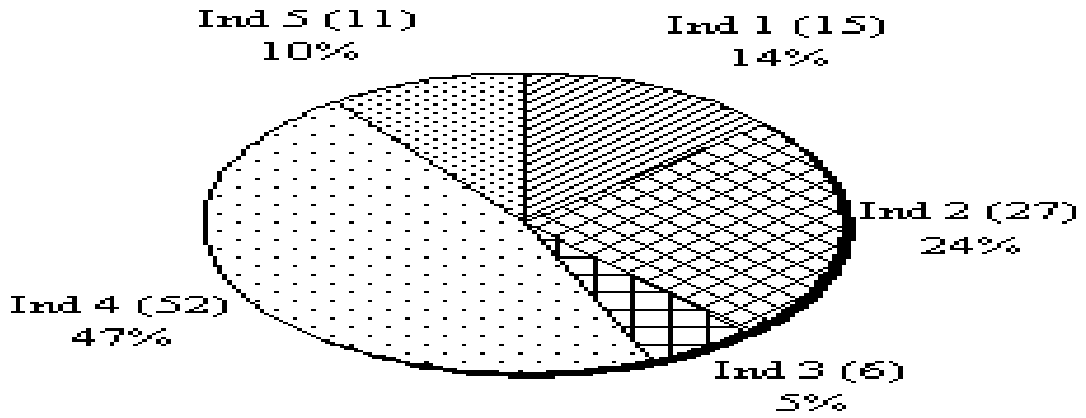
- **Maintaining good diversity** in the population is extremely crucial for the success of a GA.

Genetic algorithms. Fitness Proportionate Selection

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:

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



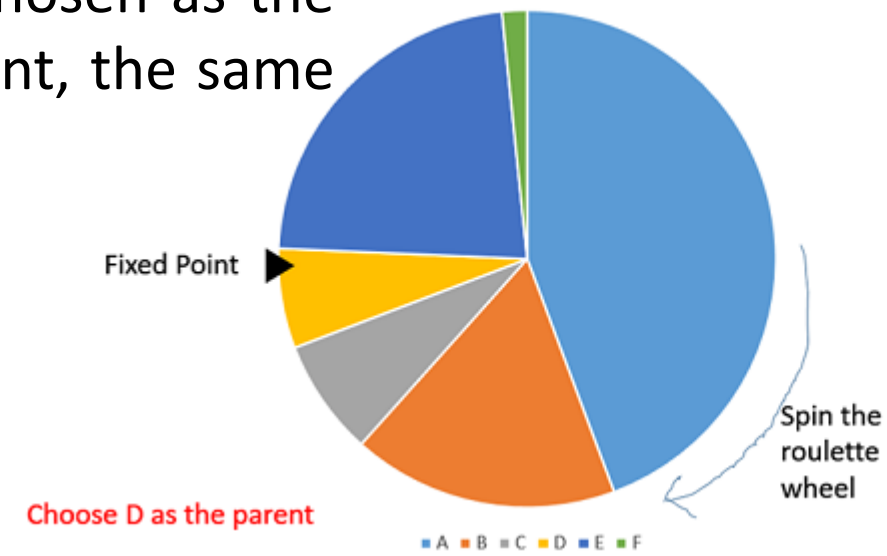
Randomly generated number = 21

Individual 2 is selected

Genetic Algorithms

Fitness Proportionate Selection : 1. Roulette Wheel Selection

- In a roulette wheel selection, the circular wheel is divided as described before.
- A fixed point is chosen on the wheel circumference as shown and the wheel is rotated.
- The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



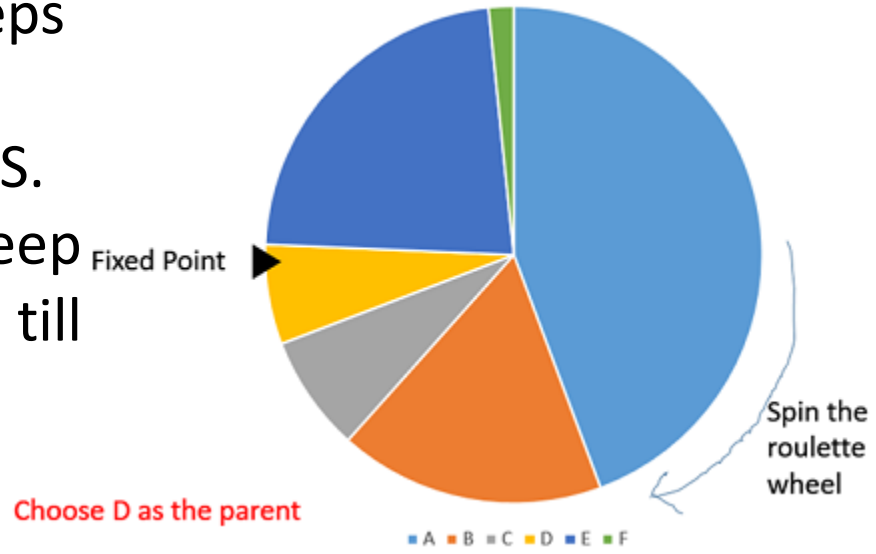
Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Genetic Algorithms

Fitness Proportionate Selection : Roulette Wheel Selection

Implementation wise, we use the following steps

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

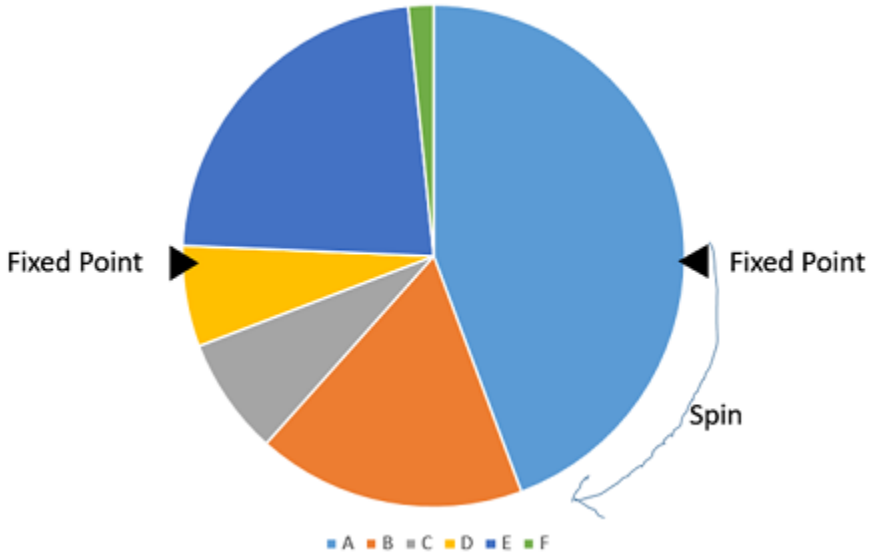


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Fitness Proportionate Selection : Stochastic Universal Sampling (SUS)

- Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the the image.

Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

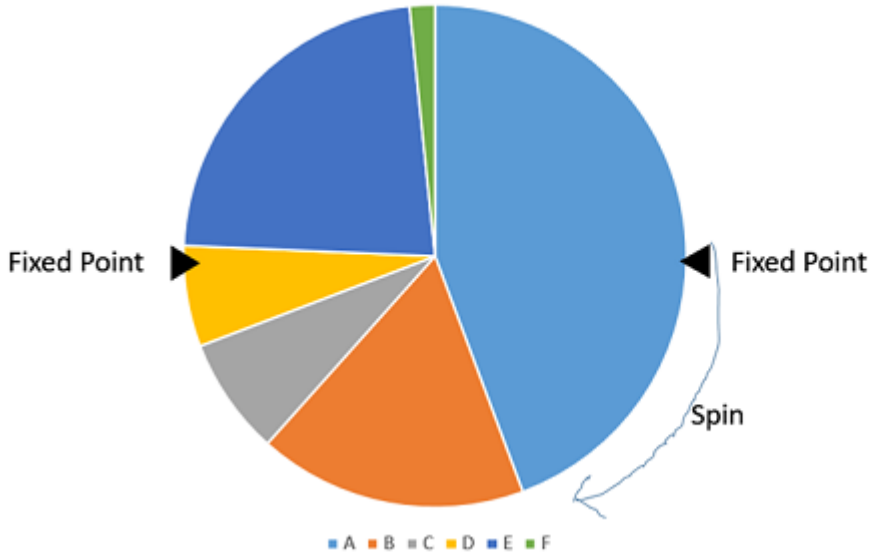


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Genetic Algorithms

Fitness Proportionate Selection : Stochastic Universal Sampling (SUS)

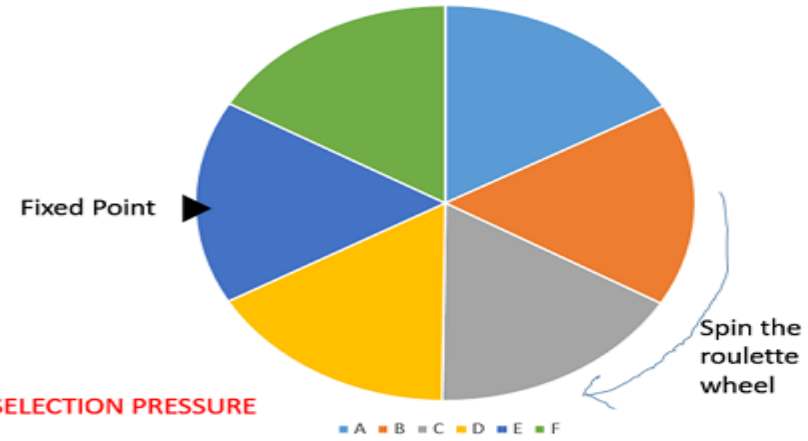
- It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Fitness Proportionate Selection : 3. Rank Selection

- Rank Selection also works with negative fitness values.
- The selection of the parents depends on the rank of each individual and not the fitness.
- The higher ranked individuals are preferred more than the lower ranked ones.

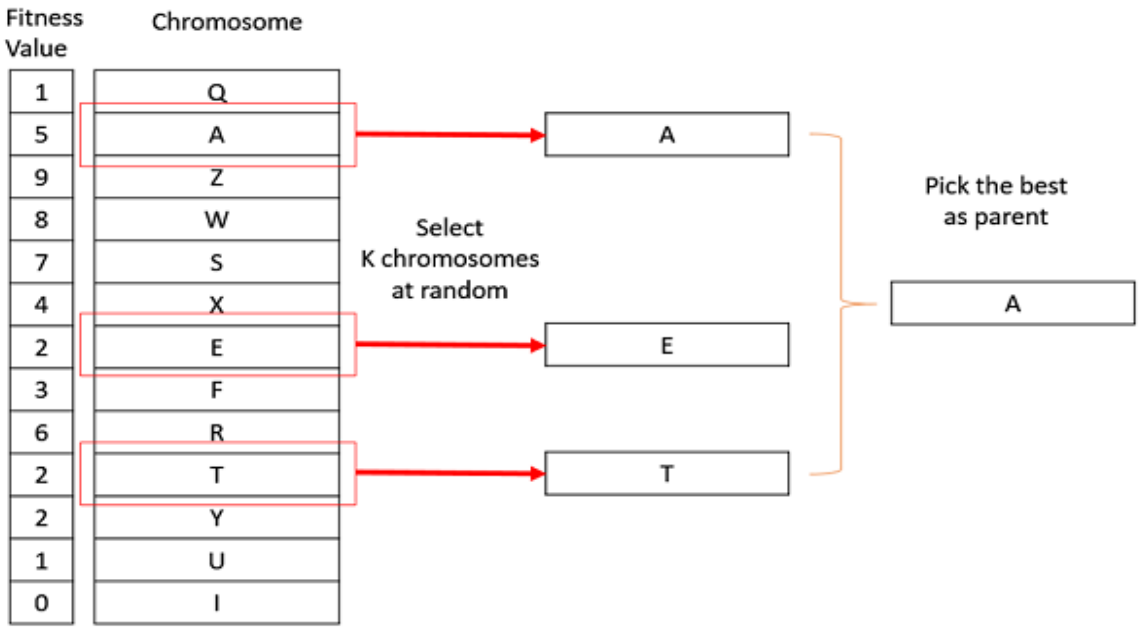


Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Genetic Algorithms

Fitness Proportionate Selection : 2. Tournament Selection

- In tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent.
- It can even work with negative fitness values.



Fitness Proportionate Selection : 4. Random Selection

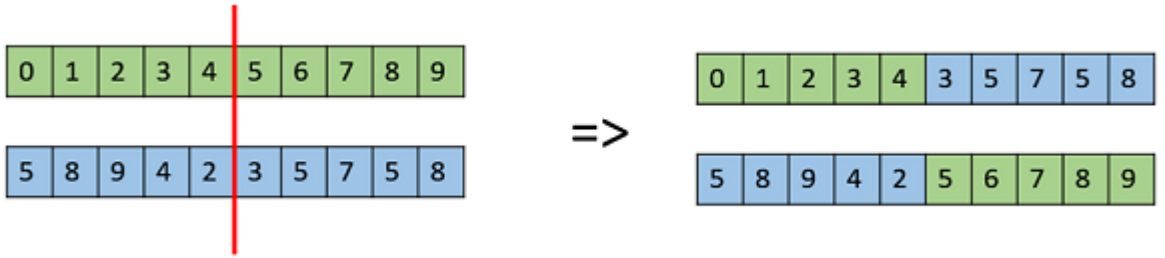
- In this strategy we randomly select parents from the existing population.
- There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

Genetic Algorithms : Crossover

The crossover operator is similar to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.

One Point Crossover

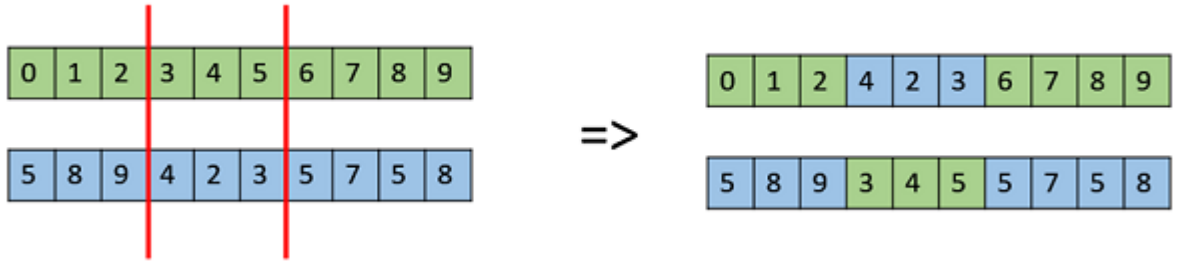
A random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



Genetic Algorithms : Crossover

Multi Point Crossover

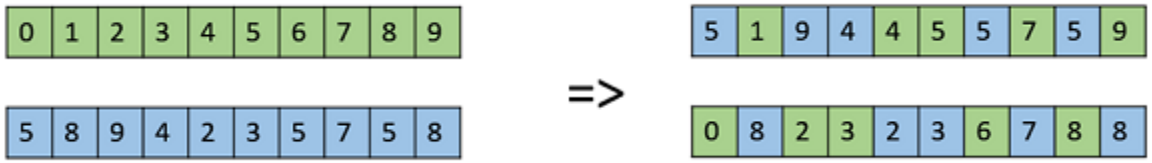
Here alternating segments are swapped to get new off-springs.



Genetic Algorithms : Crossover

Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring.



Genetic Algorithms : Crossover

Arithmetic crossover - some arithmetic operation is performed to make a new offspring

11001011 + 11011111 = 11001001 (AND)

Genetic Algorithms : Crossover

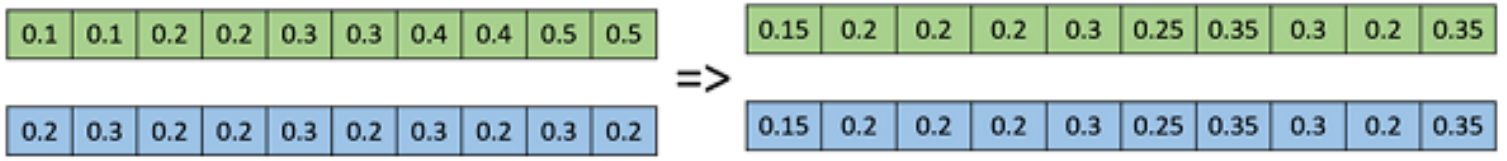
Whole Arithmetic Recombination

This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae −

Child1 = α.x + (1-α).y

Child2 = α.x + (1-α).y

Obviously, if α = 0.5, then both the children will be identical as shown in the following image.

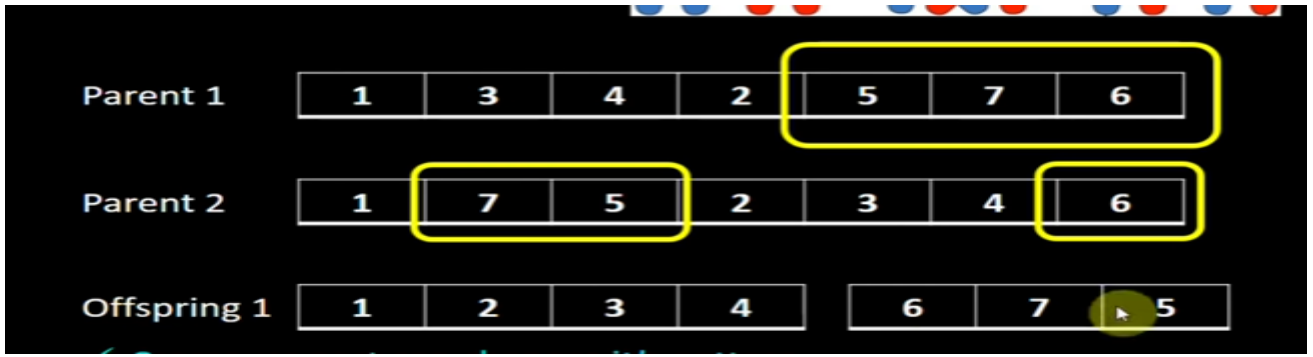


Genetic Algorithms : Crossover

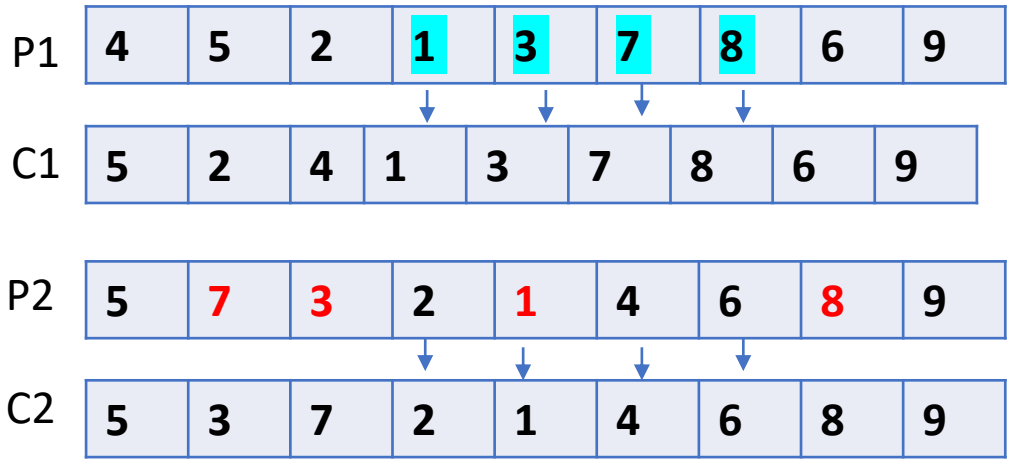
Cross Over for Permutation Encoding

If you need to crossover a 5 in parent 1. ie 5,7,6 are to be inter changed with 3,4,6,

Remove 5,7,6 from parent 2 move the other values . Then move 5,7,6 or some combination of 5,7,6 to the end.



GA- Order Crossover



- Set of randomly selected string of genes from one parent is considered (1-3-7-8 from parent 1)
- Transfer the selected genes of parent 1 from child 1 at the same locations.

- Delete 1-3-7-8 from *parent 2* and remaining genes are 9,6,4,2,5 in P2.
- Starting from the bottom to top of C1, fill the empty strings of C1 with the remaining genes of Parent 2 ie 9,6,4,2,5.

GA- Position Based Crossover

P1	4	5	2	1	3	7	8	6	9
C1	7	5	3	1	2	4	8	6	9
P2	5	7	3	2	1	4	6	8	9
C2	4	5	2	1	3	7	8	6	9

- Set of randomly selected positions from P1 (5-1-8-6) are copied to C1.
- Delete the selected nodes of P1 from P2.

The remaining nodes 9-4-2-3-7 of *parent 2* is transferred to its child 1 from bottom to top.

GA- Mutation

- Mutation is a genetic operator used to maintain *genetic diversity* from one generation of a population of genetic algorithm chromosomes to the next.

- It is analogous to biological mutation. Mutation alters *one or more gene values* in a chromosome from its initial state.

- Mutation helps preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. So, it helps the algorithm to *avoid local extreme*

- Mutation allows the *development of un-inherited characteristics* --- it promotes diversity by allowing an offspring to also evolve in ways not solely ***determined by inherited traits***.

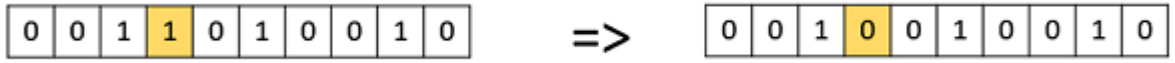
GA- Mutation

- After crossover, the strings are subjected to mutation.
- This process enables GAs to jump out of local or suboptimal solutions to avoid premature convergence.
- Mutation plays the *role of recovering the lost genetic material* as well as *for randomly distributing genetic info.*

Genetic Algorithms : Mutation Operators

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.



Random Resetting

Random Resetting is an extension of the bit flip for the integer/numerical representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

Genetic Algorithms : Mutation Operators

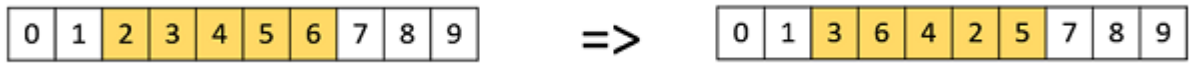
Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.

Genetic Algorithms : Mutation Operators

Scramble Mutation

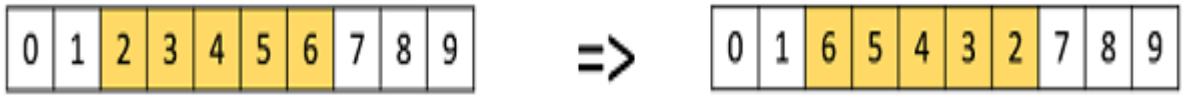
Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



Genetic Algorithms : Mutation Operators

Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.

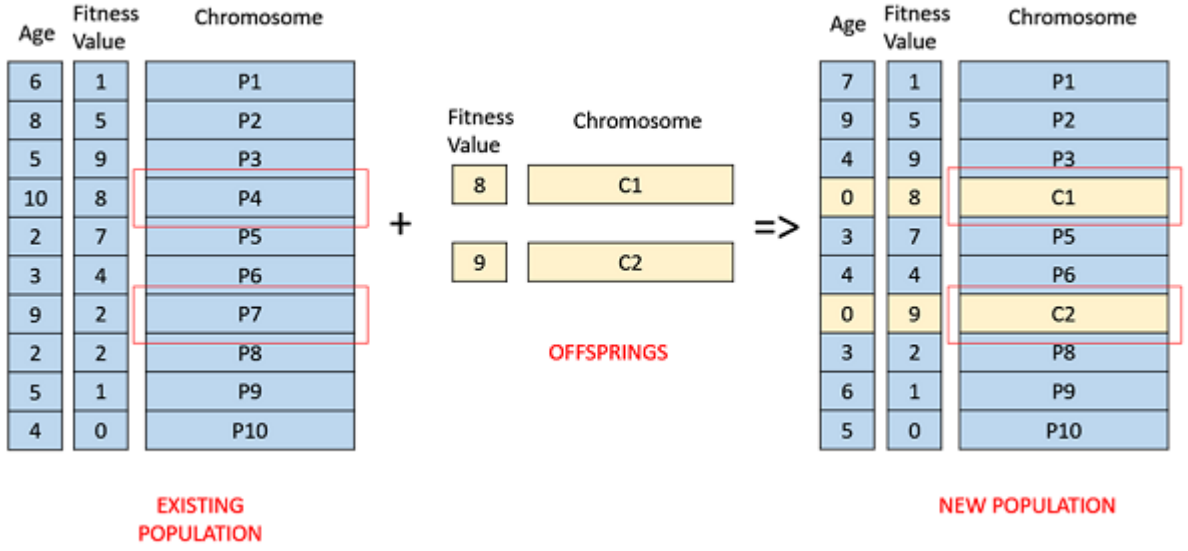


Genetic Algorithms : Survivor Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation.

Genetic Algorithms : Age Based Selection

- In Age-Based Selection, we don't have a notion of a fitness.
- Based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce
- After that, it is kicked out of the population no matter how good its fitness is.



- **Elitist Selection (Elitism):**
  - Ensures that the best chromosomes are retained in the next generation, if not selected by any other method.

Genetic Algorithms : Fitness Based Selection

- In this fitness based selection, the children tend to replace the least fit individuals in the population.
- The selection of the least fit individuals may be done using a variation of any of the selection policies.

Genetic Algorithms : Termination Conditions

- When there has been no improvement in the population fitness for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value (some threshold has been reached).

GA- Crossover and Mutation related terms

Crossover Probability:
Measure of likeliness that the selected chromosomes will be undergo crossover operation.

Crossover rate:
The total no. of crossovers that will be performed on the selected chromosomes.

Mutation Probability:
Measure of likeliness that random elements of the chromosomes will be changed with something else.

Mutation rate:
The no. of chromosomes that will undergo the mutation out of a total population.

Genetic Algorithms : Numerical Example

Given an equation, a+2b+3c+4d=30 use genetic algorithms to find the value of a, b, c, d that satisfy the above equation.

Consider an initial population of 6 chromosomes with values of a,b,c and d initialized randomly as follows

Chromosome[1] = [a;b;c;d] = [12;05;23;08]
Chromosome[2] = [a;b;c;d] = [02;21;18;03]
Chromosome[3] = [a;b;c;d] = [10;04;13;14]
Chromosome[4] = [a;b;c;d] = [20;01;10;06]
Chromosome[5] = [a;b;c;d] = [01;04;13;19]
Chromosome[6] = [a;b;c;d] = [20;05;17;01]

Consider the objective function f(x) = (a+2b+3c+4d-30) to evaluate the fitness value of chromosomes.

Probability of choosing a chromosome i is is formulated by: P[i] = Fitness[i] / Total

Each chromosome has a random value R associated with it as follows

R[1] = 0.191 R[2] = 0.259 R[3] = 0.760 R[4] = 0.006 R[5] = 0.159 R[6] = 0.340.

Chromosome number k will be selected for crossover if random generated R for Chromosome k is below cross over rate. Consider the cross over rate to be 25%

Genetic Algorithms :
Numerical Example

We compute the objective function value for each chromosome produced in initialization step:

F_obj[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)
= Abs((12 + 10 + 69 + 32) - 30)
= Abs(123 - 30)
= 93

F_obj[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)
= Abs((02 + 42 + 54 + 12) - 30)
= Abs(110 - 30)
= 80

F_obj[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
= Abs((10 + 08 + 39 + 56) - 30)
= Abs(113 - 30)
= 83

F_obj[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
= Abs((20 + 02 + 30 + 24) - 30)
= Abs(76 - 30)
= 46

F_obj[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30)
= Abs((01 + 08 + 39 + 76) - 30)
= Abs(124 - 30)
= 94

F_obj[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)
= Abs((20 + 10 + 51 + 04) - 30) = Abs(85 - 30)
= 55

Genetic Algorithms : Numerical Example

The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome. To avoid divide by zero problem, the value of F_obj is added by 1.

Fitness[1] = 1 / (1+F_obj[1])
= 1 / 94
= 0.0106

Fitness[2] = 1 / (1+F_obj[2])
= 1 / 81
= 0.0123

Fitness[3] = 1 / (1+F_obj[3])
= 1 / 84
= 0.0119

Fitness[4] = 1 / (1+F_obj[4])
= 1 / 47
= 0.0213

Fitness[5] = 1 / (1+F_obj[5])
= 1 / 95
= 0.0105

Fitness[6] = 1 / (1+F_obj[6])
= 1 / 56
= 0.0179

Total = 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179
= 0.0845

Genetic Algorithms : Numerical Example

The probability for each chromosomes is formulated by: P[i] = Fitness[i] / Total

P[1] = 0.0106 / 0.0845 = 0.1254

P[2] = 0.0123 / 0.0845 = 0.1456

P[3] = 0.0119 / 0.0845 = 0.1408

P[4] = 0.0213 / 0.0845 = 0.2521

P[5] = 0.0105 / 0.0845 = 0.1243

P[6] = 0.0179 / 0.0845 = 0.2118

From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes

Genetic Algorithms : Numerical Example

For the
selection process we use roulette wheel, for that we should compute the cumulative probability values:

C[1] = 0.1254

C[2] = 0.1254 + 0.1456

= 0.2710

C[3] = 0.1254 + 0.1456 + 0.1408

= 0.4118

C[4] = 0.1254 + 0.1456 + 0.1408 + 0.2521

= 0.6639

C[5] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243

= 0.7882

C[6] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118

= 1.0

Genetic Algorithms : Numerical Example

The process is to generate random number R in the range 0-1 as follows.

R[1] = 0.201
R[2] = 0.284
R[3] = 0.099
R[4] = 0.822
R[5] = 0.398
R[6] = 0.501

Genetic Algorithms : Numerical Example

If random number R[1] is greater than C[1] and smaller than C[2] then select Chromosome[2] as a chromosome in the new population for next generation:

NewChromosome[1] = Chromosome[2]
NewChromosome[2] = Chromosome[3]
NewChromosome[3] = Chromosome[1]
NewChromosome[4] = Chromosome[6]
NewChromosome[5] = Chromosome[3]
NewChromosome[6] = Chromosome[4]

Chromosomes in the population thus became:

Chromosome[1] = [02;21;18;03]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;08]
Chromosome[4] = [20;05;17;01]
Chromosome[5] = [10;04;13;14]
Chromosome[6] = [20;01;10;06]

Genetic Algorithms : Numerical Example

Lets say we generate some random numbers and those that are less than the cross over rate we use for pairing

R[1] = 0.191
R[2] = 0.259
R[3] = 0.760
R[4] = 0.006
R[5] = 0.159
R[6] = 0.340

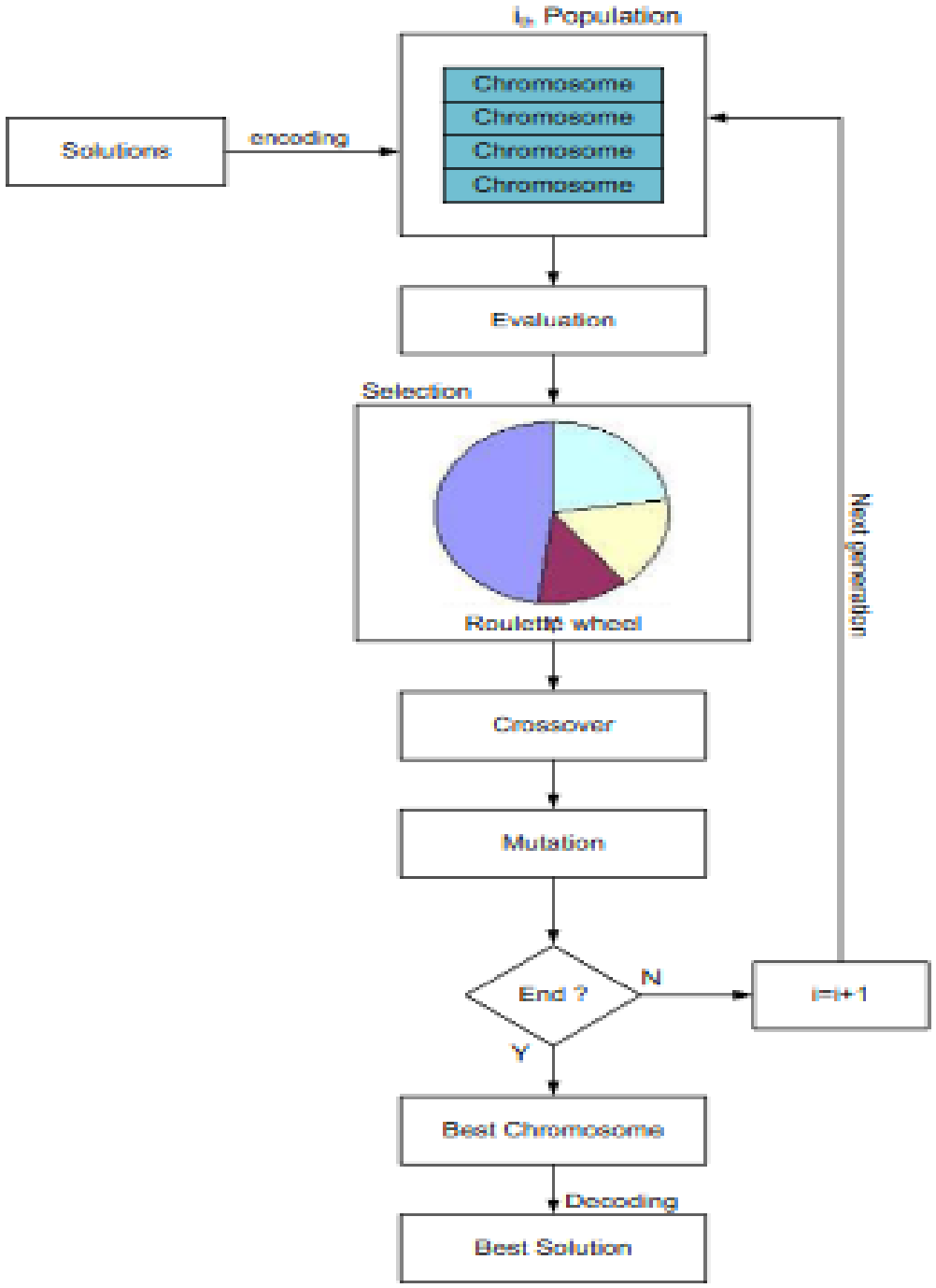
For random number R above, parents are Chromosome[1], Chromosome[4] and Chromosome[5] will be selected for crossover.

Chromosome[1] >< Chromosome[4]
Chromosome[4] >< Chromosome[5]
Chromosome[5] >< Chromosome[1]

Genetic Algorithms : Numerical Example

You then apply cross over and mutation for the rest of the solutions.

Please see this document for the solution https://arxiv.org/ftp/arxiv/papers/1308/1308.4675.pdf



Genetic Operators

Broadly 2 categories of Operators

Crossover (recombination)

Exchanging genetic material between the individuals of the chromosomal population

Mutation

A genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next

Note: A genetic operator which works well for one problem may not work well for another problem. [1]

GA- Mutation

- Mutation can be implemented as *One's complement Operator.*
- Let C= [01101011] be a chromosome, then one's complement operator gives C=[10010100].

- **Inversion**
  - Two positions are randomly selected within a chromosome. Then the substring between these positions is inverted.

P_X=[2 5 4 9 6 8 1 3 7]

C_X=[2 5 8 6 9 4 1 3 7]

GA- Mutation

- **Insertion**

➢ A node is randomly selected and inserted in a random position.

- P_X=[2 5 4 9 6 8 1 3 7]
- C_X=[2 6 5 8 9 4 1 3 7]

- **Heuristic Mutation**

➢ It is based on neighborhood technique

P_X=[1 2 3 4 5 6 7 8 9]

Neighbors are formed with these nodes.

N_X=[1 2 3 4 5 8 7 6 9]

N_X=[1 2 8 4 5 3 7 6 9]

N_X=[1 2 8 4 5 6 7 3 9]

N_X=[1 2 6 4 5 8 7 3 9]

N_X=[1 2 6 4 5 3 7 8 9]

After evaluating all the neighbors the best neighbor is selected.

MACHINE INTELLIGENCE

Genetic Algorithm vs Genetic Programming (FYI)

Genetic Algorithm	Genetic Programming
1.Often produces invalid states.	1. Special case of GAs, where each individual is a computer program
2. GA's are probabilistic search algorithms which mimic the process of natural evolution.	2. GP explores the algorithmic search space and evolve computer program to perform a defined task.
3. In GA's, each individual is a candidate solution.	3. GP is an application of GA. O/p of GP is another program.
4. GA has an o/p which is a quantity	

References

[1] Xin Yao, An empirical study of genetic operators in genetic algorithms, Microprocessing and Microprogramming, Volume 38, Issues 1–5, 1993, Pages 707-714, ISSN 0165-6074.

[2] Hermawanto D. Genetic algorithm for solving simple mathematical equality problem. arXiv preprint arXiv:1308.4675. 2013 Aug 16.

[3] An empirical study of genetic operators in genetic algorithms X Yao Microprocessing and Microprogramming, 1993 - Elsevier

THANK YOU

Dr. Arti Arya

Department of Computer Science

artiarya@pes.edu

+91 9972032451 Extn 029