Unit 5

Genetic Algorithms

→ adaptive heuristic search algos that belong to the larger part of evolutionary algos.

→ based on ideas of natural selection & genetics.

→ used to generate high-quality sol^{n}s for optimization problems & search problems.

→ Genetic Algos simulate the process of natural selection which means those species who can adapt to changes in their env are able to survive & reproduce & go not next generation.
They simulate "survival of the fittest" among individual of consecutive generation for solving a problem.

→ Each generation consist of a pop^n of individuals and each individual represents a point in search space and possible sol^n.

→ Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

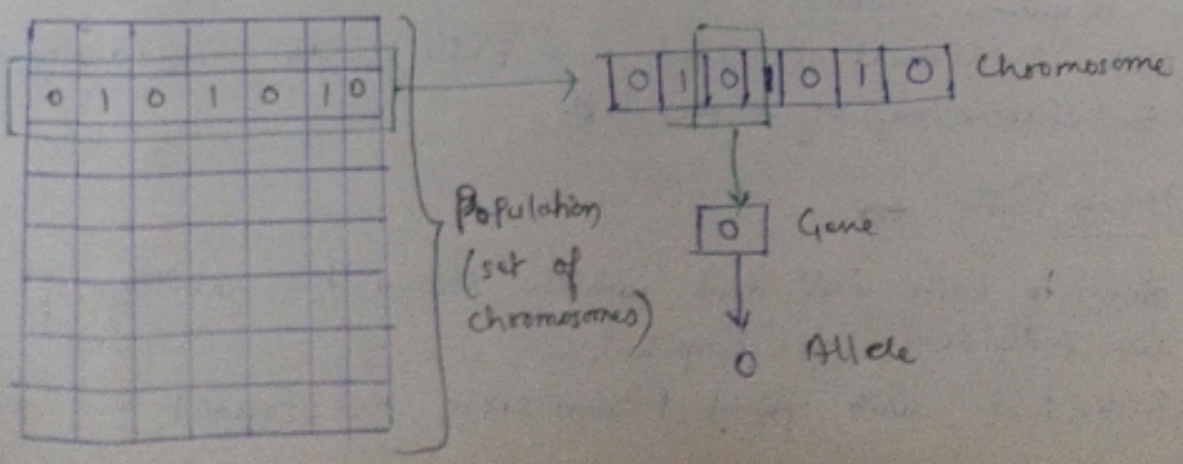
→ A generic structure of GAs is presented in both pseudo-code & graphical form.

Population - subset of all possible (encoded) sol^{n}s to the problem. It is analogous to pop^n for human beings except instead of human beings we have Candidate sol^{n}s.

Chromosomes - one such sol^n to the given problem.

Gene - one element position of a chromosome.

Allele - It is the value a gene takes for a particular chromosome.



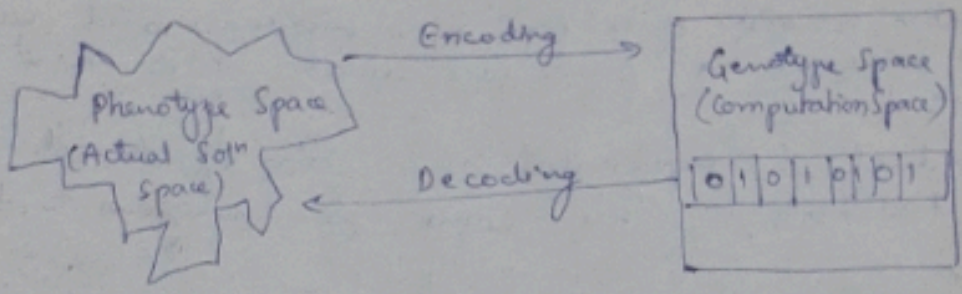
Genotype – popⁿ in the computation space. In the computation space the sol'ⁿs are represented in a way which can be easily understood and manipulated using a computing system.

Phenotype – popⁿ in actual real world solⁿ space in which sol'ⁿs are represented in a way they are represented in real world situations.

Decoding & Encoding – for simple problems, phenotype and genotype spaces are same. But in most case they are diff.

Decoding is a process of transforming a solⁿ from genotype to phenotype space.

Encoding is a process of transforming from phenotype to genotype space.



fitness funcⁿ – funcⁿ which takes the solⁿ as input and produces the suitability of the solⁿ as output.

In some cases fitness funcⁿ & objective funcⁿ may be same while in other they are diff.

Genetic Operators – they alter the genetic composition of the offspring. (crossover, mutation, selection etc)

Search Space

→ popⁿ of individuals are maintained within search space.
→ Each individual represents a solⁿ in search space for a given problem.
→ Each individual is coded as a finite length vector (analogous to chromosome) of components.

Fitness Score

→ given to each individual which shows the ability of an individual to "compete".
→ individual with optimal fitness score are sought.

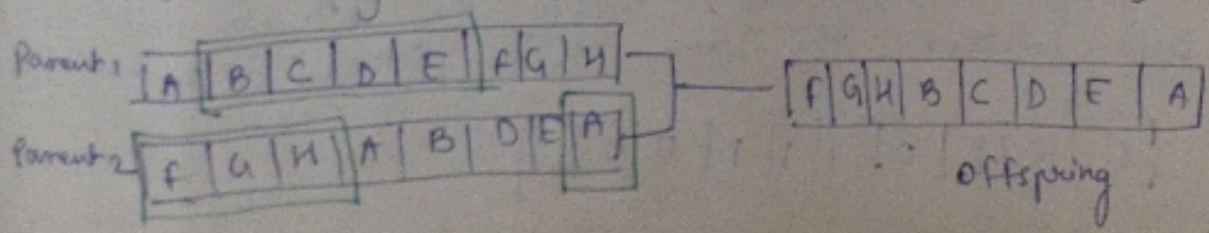
→ The GAs maintain the pop^n of n individuals (chromosome/sol^{n}s) along with their fitness scores.
→ individuals with better fitness scores have more chance to produce their offspring than others.
→ individuals with better fitness scores are selected to produce better offspring by combining chromosomes of parents.
→ pop^n size is static so space for new arrivals has to be created.
→ So some individuals die and get replaced by new arrivals. eventually creating new generation when they does not produce more offspring. It is hoped that over successive generations better sol^{n}s will arrive while least fit die.
→ Each new generation has on avg more "better genes" than the individual (sol^n) of previous generations. Thus each new generations have better "partial sol^{n}s" than previous generations.
→ Once the offspring produced have no significant diff from offspring produced by previous pop^{n}s, the pop^n is converged.
→ The algo is said to be converged to a set of sol^{n}s for the problem.

Operators of GAs

→ Once the initial generation is created, the algo evolves the generation using following operators —

① Selection Operator → idea is to give preference to individuals with good fitness scores and allow them to pass their genes to successive generations.

② Crossover Operator - this represents pairing of individuals to produce offspring.
Two individuals are selected using selection operator and crossover sites are chosen randomly.
Then genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

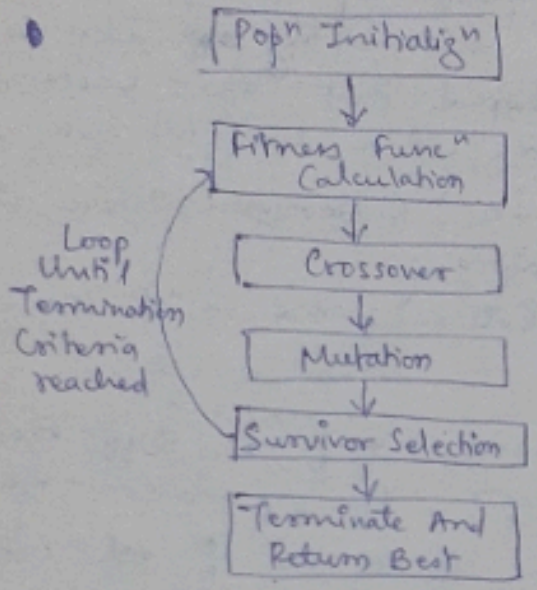


(3) Mutation Operator → idea is to insert random genes in offspring to maintain the diversity in pop^n to avoid premature convergence.

Before Mut^n | F | G | H | B | C | D | E | A |

After Mut^n | F | G | M | B | C | D | E | N |

Basic Structure (Procedure)



- start with initial pop^n, select parents from this pop^n to produce their offspring
- Apply crossover and mutation operators on parents to generate new off-spring.
- And these offsprings replace the existing individuals in pop^n & process repeats.
- → This way genetic algos try to mimic human evol^n.

Genotype Representation

→ represen^n used to represent our sol^{n}s.
→ improper rep^n lead to poor performance of GA.
→ proper rep^n having proper definition of mappings betn the phenotype & genotype space is essential for success of a GA.

(1) Binary Rep^n

→ simplest, most widely used
- in problems where sol^n space consists of Boolean decision variables — yes or no, the binary rep^n is natural.

Ex → for a 0/1 Knapsack Problem. If there are n items, we can represent a sol^n by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not(0).

| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

→ The problem here is ~~with kind of encoding~~ that diff bits have diff significance & ∴ mutation and crossover operators have undesired consequences.
This can be solved using Gray coding, as a change in one bit does not have a massive effect on sol^n.

② Real Valued Rep^n

→ for problems where we want to define genes using continuous rather than discrete values.
→ The precision of these real valued or floating pt numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3

③ Integer Rep^n

→ for discrete valued genes, we cannot always limit sol^n space to binary 'yes' or 'no'.
→ Ex → if we encode 4 directions — North, South, East & West, we can encode them as {0, 1, 2, 3}.

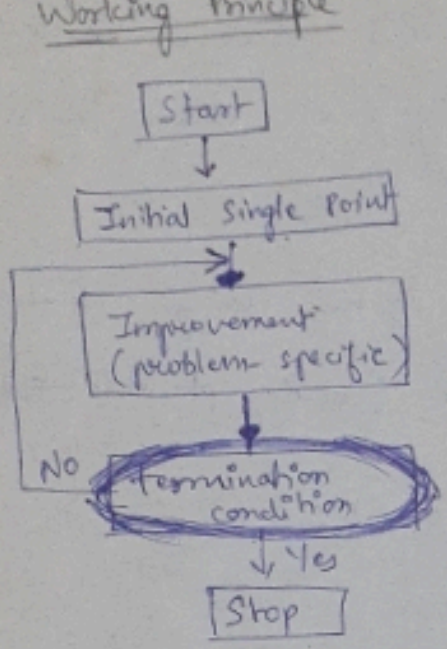
1	2	3	4	3	2	4	1	2	1

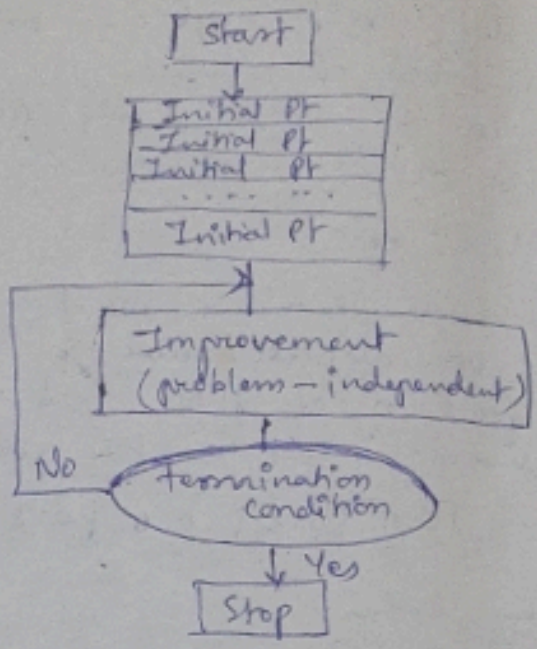
④ Permutation Rep^n

→ In many problems, sol^n is represented by an order of elements.
→ Ex → Travelling ~~Salesman~~ Salesman Problem.
→ Here salesman has to take a tour of all cities, visiting each city exactly once and come back to starting city.
→ Total distance of tour has to be minimized.
→ Sol^n of TSP is an ordering or permutation of all cities and ∴ using a permutation rep^n.

1	5	9	8	7	4	2	3	6	0

Working Principle





Start → generates a random popⁿ of n chromosomes.

Fitness → calculates the fitness f(x) of each chromosome x in the popⁿ

New popⁿ → generates a new popⁿ by repeating following steps until new popⁿ is finished.

Selection → chooses 2 parent chromosomes from popⁿ as per their fitness. The better fitness, higher the probability of getting selected.

Crossover → In cross over probability, cross over the parent to form new offspring. If no crossover was performed, the offspring is the exact copy of parents.

Mutation → In Mutatⁿ prob., mutate new offspring at each locus.

Accepting → It places new offspring in the new popⁿ.

Replace → uses newly generated popⁿ for a further run of the algo.

Test → if end condition satisfied, then it stops and returns the best solⁿ in current popⁿ.

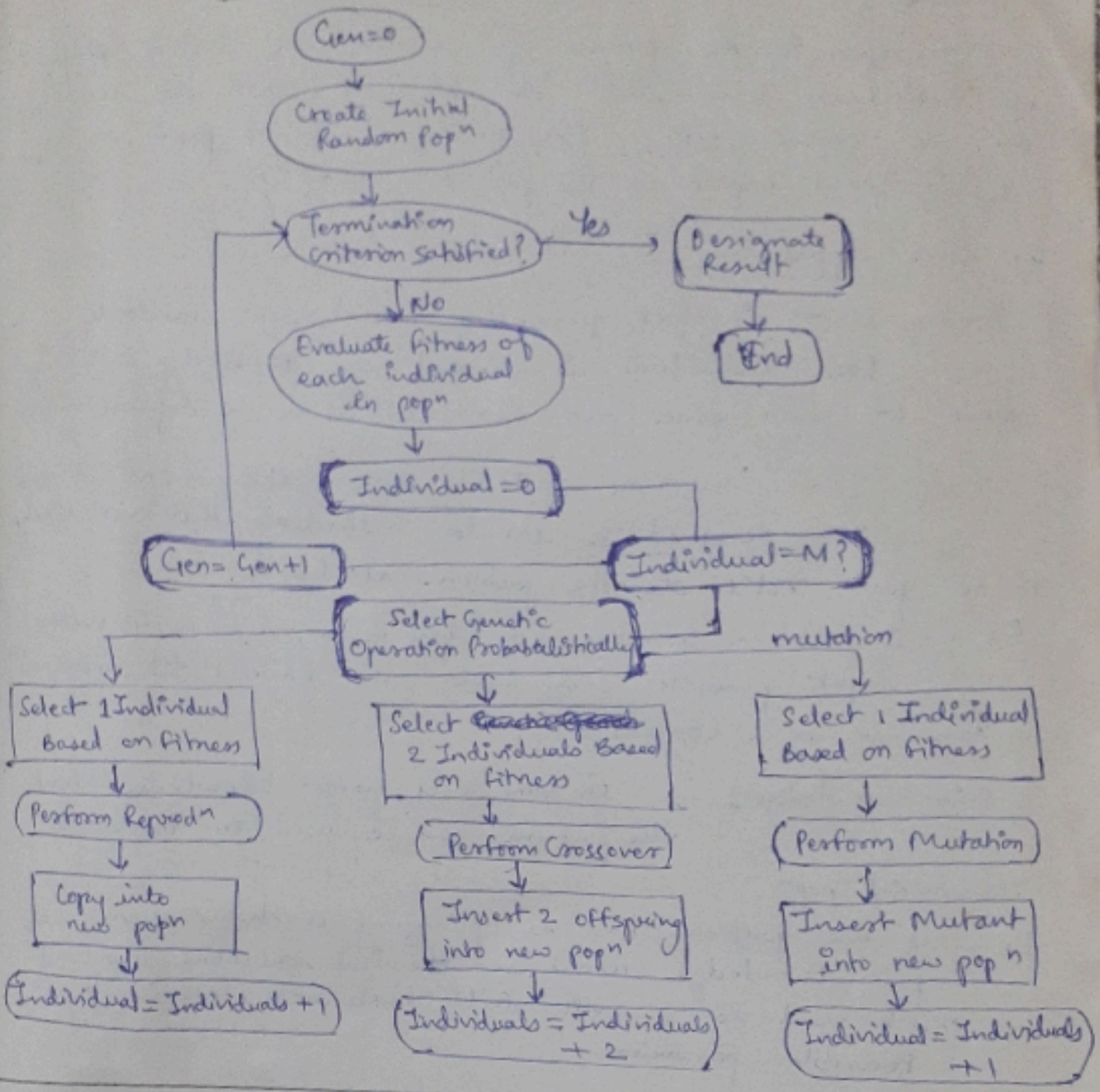
Loop → need to go to 2nd step for fitness evalⁿ.

→ Chromosomes are encoded solⁿs to a problem.

→ Crossover includes 2 chromosomes swapping chunks of data.

→ Mutation introduces slight changes into a little extent of popⁿ, and is representative of evolutionary step.

Flowchart for Genetic Programming



Traditional Algos

- → selects next pt in series by a deterministic computation
- → creates an individual pt at each iteration. The sequence of pts approaches an optimal solⁿ.
- → Advancement in each iteration is problem specific.

Genetic Algos

- → selects next popⁿ by computation which utilizes random no generators.
- → creates a popⁿ of pts at every iteration. The best pt in the popⁿ approaches an optimal solⁿ
- → Concurrence in each iteration is a problem independent.

Initialization in GAs

→ It refers to the process of creating an initial pop^n of individuals from which the GA can start the search for an optimal sol^n. The quality of initial pop^n can have a significant impact on the performance of GA.

→ Some approaches to initialization include -

① Random I. — Involves generating initial pop^n randomly. Each individual is created by randomly selecting values for each gene (parameter) within a predefined range.

② Heuristic I. — Heuristic methods are problem-specific and are designed to create individuals that are likely to be good sol^{n}s to the problem at hand.

Ex → If problem involves scheduling tasks, a heuristic method might generate individuals that have tasks grouped together in a logical way.

③ Prior Knowledge I. — In some cases, prior knowledge about the problem can be used to generate the initial pop^n.

Ex → if the problem involves optimizing a chemical process, prior knowledge about the chemical reactions involved can be used to generate individuals with chemically feasible parameters.

④ Re-seeding - If GA is not making progress, or has converged to a suboptimal sol^n, re-seeding can be used to introduce new genetic material into the pop^n. Re-seeding involves creating a new pop^n from scratch, using one of the above methods.

→ Choice of initialz^n method depends on the problem at hand and avl knowledge about the problem.

→ In general, a good initz^n method should generate a diverse initial pop^n that covers a large portion of search space.

Selection in GAs

→ Selection is a key operator in genetic algos that determines which individuals from current pop" will be chosen for reprod" and create next generation.

→ Selection process aims to simulate natural selection process of evol" by favoring individuals with higher fitness values, i.e. those that are better adapted to the problem at hand.

→ Some approaches to selection in GAs –

① Roulette wheel Selection – here individuals are selected with a probability proportional to their fitness. The fitness of each individual is normalized by dividing it by the total fitness of the pop". Then, a roulette wheel is spun, and individuals are selected based on where the wheel stops.

② ~~Tournament~~ Rank-Based Selection – here individuals are ranked based on their fitness values, and selection probability is determined based on the rank. Typically selection probability decreases with rank, so that the fittest individuals have a higher probability of being selected.

③ Tournament Selection – here a subset of individuals is randomly selected from pop", and individual with highest fitness value in that subset is selected. This process is repeated until desired no. of individuals is selected.

④ Stochastic ~~Selection~~ Universal Sampling – here individuals are placed on a roulette wheel, similar to roulette wheel S. method. But here instead of spinning the wheel multiple times, a fixed no. of equally spaced pointers are placed around the wheel, and the individuals closest to the pointers are selected.

→ Choice of selection method depends on problem at hand and desired balance both exploration (diversity) and exploitation (convergence) of search space.

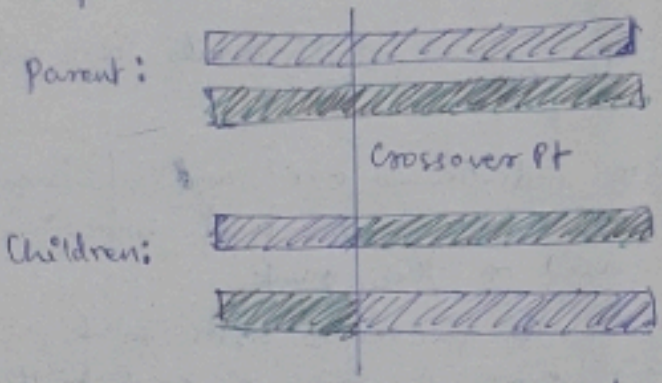
Crossover

→ used to vary the programming of a chromosome or chromosomes from one generation to the next.

→ Two strings are picked at random to crossover in order to produce superior offspring.

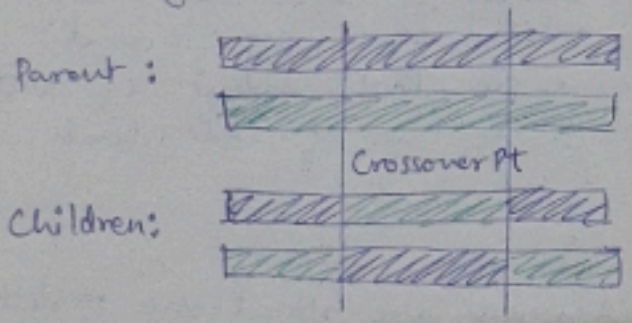
Types →

① Single Point Crossover → A crossover pt on parent organism string is selected. All data beyond that pt in the organism string is swapped bet the 2 parent organisms.
Strings are characterized by Positional Bias.



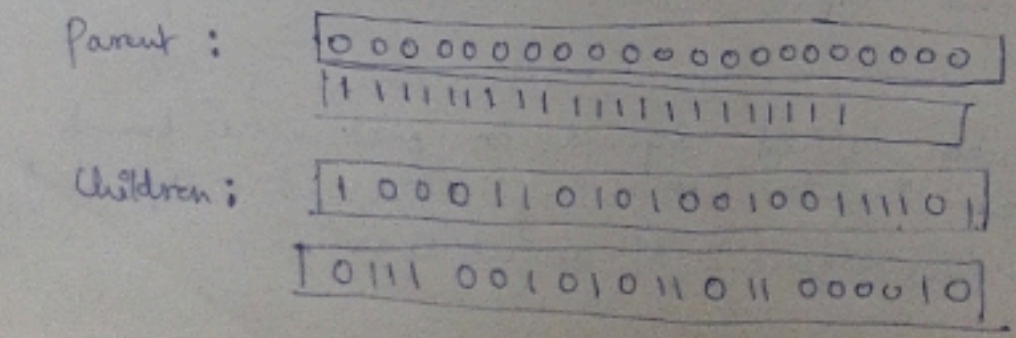
Chromosome1	1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0
C2	1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0
Offspring1	1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 0
OS 2	1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0

② Two Point Crossover → 2 random pts are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these pts.



C1	1 1 0 1 1 | 0 0 1 0 0 | 1 1 0 1 0
C2	1 0 1 0 1 | 1 1 0 0 0 | 0 1 1 1 0
OS1	1 1 0 1 1 | 1 1 0 0 0 | 1 1 0 1 0
OS2	1 0 1 0 1 | 0 0 1 0 0 | 0 1 1 1 0

③ Uniform Crossover → Each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes.
Ex → tossing of a coin

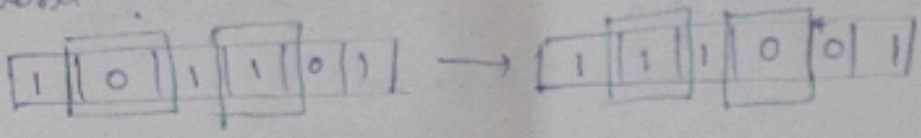


Mutation

→ It is a unary operator and needs only one parent to work on. It does so by selecting a few genes from our selected chromosome & apply the desired algo.

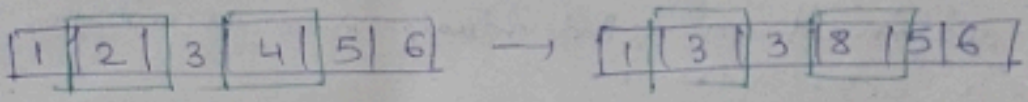
Types →

① Bit Flip Mutation — mainly used for bit string manipulation.
→ Here we select one or more genes (array indices) and flip their values i.e. we change 1s to 0s and vice versa.

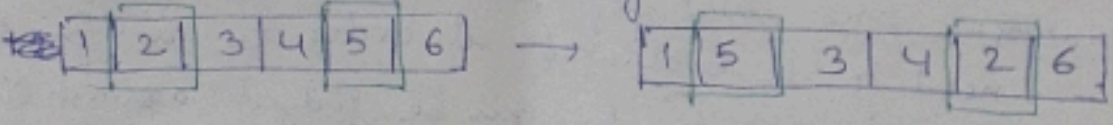


② Randomly Resetting Mutation — Here we select one or more genes and replace their values with another random value from their given ranges.

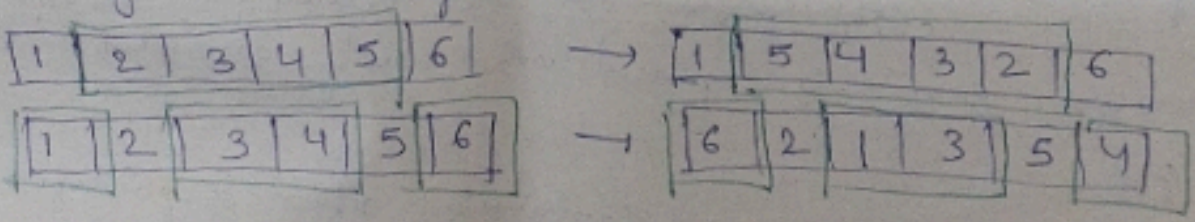
Ex → a[i] ranges from [1, 6] then random resetting mutation will select one value from [1, 6] and replace a[i]'s value with it.



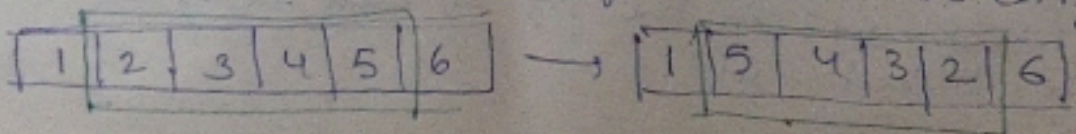
③ Swap Mutation → we select 2 genes from our chromosome & interchange their values.



④ Scramble Mutn → we select a subset of our genes and scramble their value. The selected genes may not be contiguous.

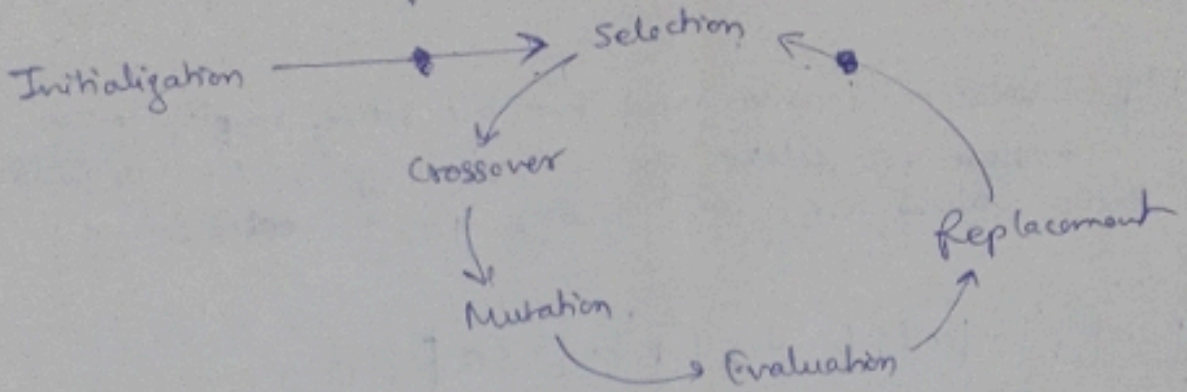


⑤ Inversion Mutn → we select a subset of our genes & reverse their order. Here genes have to be contiguous.



Generational Cycle of GAs

→ to augment or replace pop^ns in a generational manner in order to improve the overall fittest sol^n.



① Initializ^n

② Selection

③ Variation – Once parent pop^n is fully populated via selection process, a child pop^n is created which form the basis of next generation.
This child pop^n is generated by variation operators, which are performed on individuals from parent pop^n.
(Ex→ crossover, mutn)

④ Crossover

⑤ Mutation

⑥ Evaluation – once child pop^n has been created, all children need to be evaluated in order to assign a fitness value by which they can be judged against their peers.

⑦ Replacement – replace the old "N" pop^n with new "N+1" child pop^n. In one method the entire previous gen^n is replaced with newly generated child pop^n.

⑧ Termination – GA will terminate after a predefined no of generations or some stopping criterion is met (Ex→ fitness above some threshold) The fittest sol^n in the pop^n is then returned as the overall best sol^n.

Optimization Problems in GAs

→ Involves finding optimal soln or set of sol"s to a given problem. This is done by simulating the process of natural selection & evoln in a popn of potential sol"s.

→ Basic steps in solving an optimizn problem are -

① Encoding the potential sol"s - 1st step is to represent the potential soln as chromosomes or strings of genetic info. The chromosomes are typically binary strings, but they can also be represented in other forms, such as real-valued vectors or permutations.

② Generating initial popn - To generate an initial popn of potential sol"s it can be done by using heuristics or other optimizn techniques.

③ Evaluating fitness of each soln - evaluated using fitness func". The fitness func" measures how well a particular soln solves the problem.

④ Selecting the fittest individuals - to serve as parents for next gen", done by using selection techniques.

⑤ Applying genetic operators - Crossover, mutn applied to selected individuals to create new offspring.

⑥ Evaluating fitness of offspring - using fitness func"

⑦ Creating next gen" - fittest individual from previous gen" along with their offspring, create next gen" of potential sol"s.

⑧ Repeating the process - until a satisfactory soln is found or for a fixed no. of ~~generations~~ gen".

→ Goal is to find optimal soln or set of sol"s that maximize or minimize the fitness func". The process of natural selⁿ & evolution helps in it.

Applications

(1) Optimiz^{n} Problems - can be used to solve opt^{n} problems in various fields such as engg, economics & logistics. They can be used to optimize parameters for complex systems such as flight control systems, traffic management systems, & supply chain management systems.

(2) Machine Learning - to optimize parameters of ML models.
↳ to improve accuracy of classifi^{n}, regression & clustering algos.

(3) Robotics - to optimize design of robots & control systems.
↳ to optimize config^{n} of robotic arms, path planning of mobile robots, & control of multi^{n} - robot systems.

(4) Computational Biology - to analyze biological data and simulate biological processes.
↳ to analyze gene expression data, simulate evol^{n}, and optimize molecular structures.

(5) Game Theory - to solve game theory problems.

(6) Computer Networks - to optimize config^{n} of computer networks, such as routing of data packets and allocation of network resources.

(7) Financial Modeling - to optimize trading strategies & portfolio management.
↳ to identify patterns in financial data and make predictions about future market trends.

Himanshi Rajput