STREAM: ECE
SUBJECT NAME: SOFT COMPUTING
SUBJECT CODE: EC 803 C

Prepared by: Ms. Suranjana Saha

MODULE IV: GENETIC ALGORITHMS

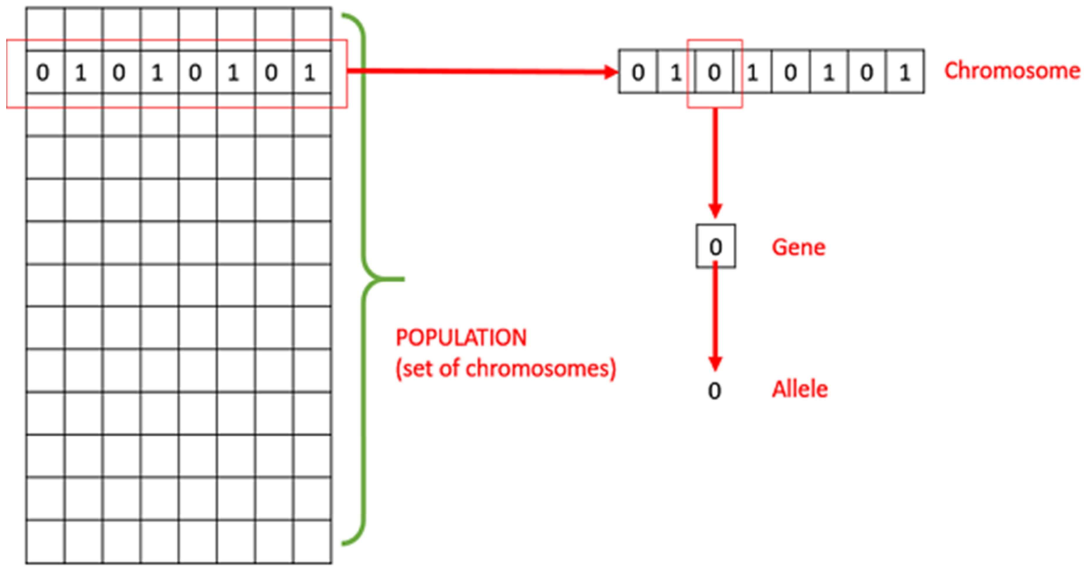
GENETIC ALGORITHMS - FUNDAMENTALS

This section introduces the basic terminology required to understand GAs. Also, a generic structure of GAs is presented in both pseudo-code and graphical forms. The reader is advised to properly understand all the concepts introduced in this section and keep them in mind when reading other sections of this tutorial as well.

Basic Terminology

Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.

- Population − It is a subset of all the possible encoded solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- 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.

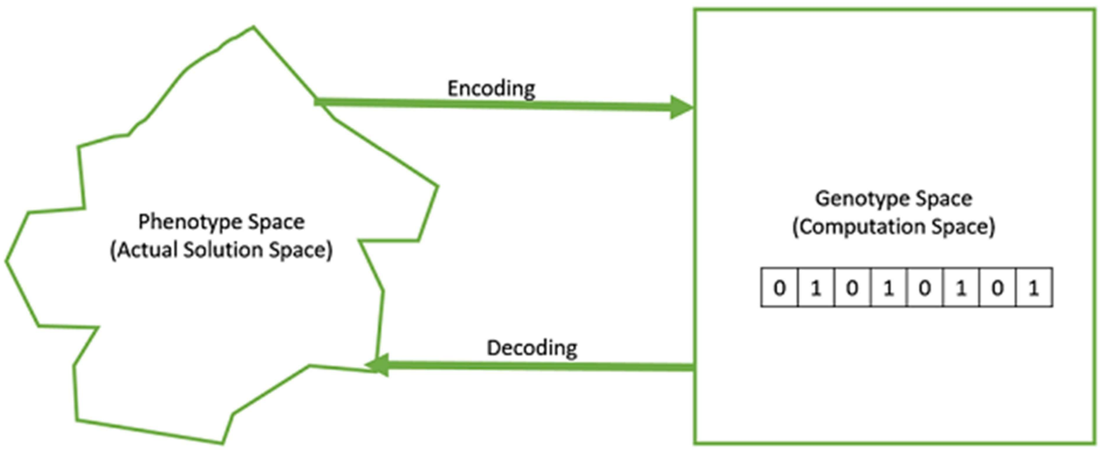


- Genotype − Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.

- 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.
- Decoding and Encoding − For simple problems, the phenotype and genotype spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. 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. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

  For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.

  However, in the genotype space it can be represented as a binary string of length n where "n" is the number of items. A 0 at position x represents that x^{th} item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

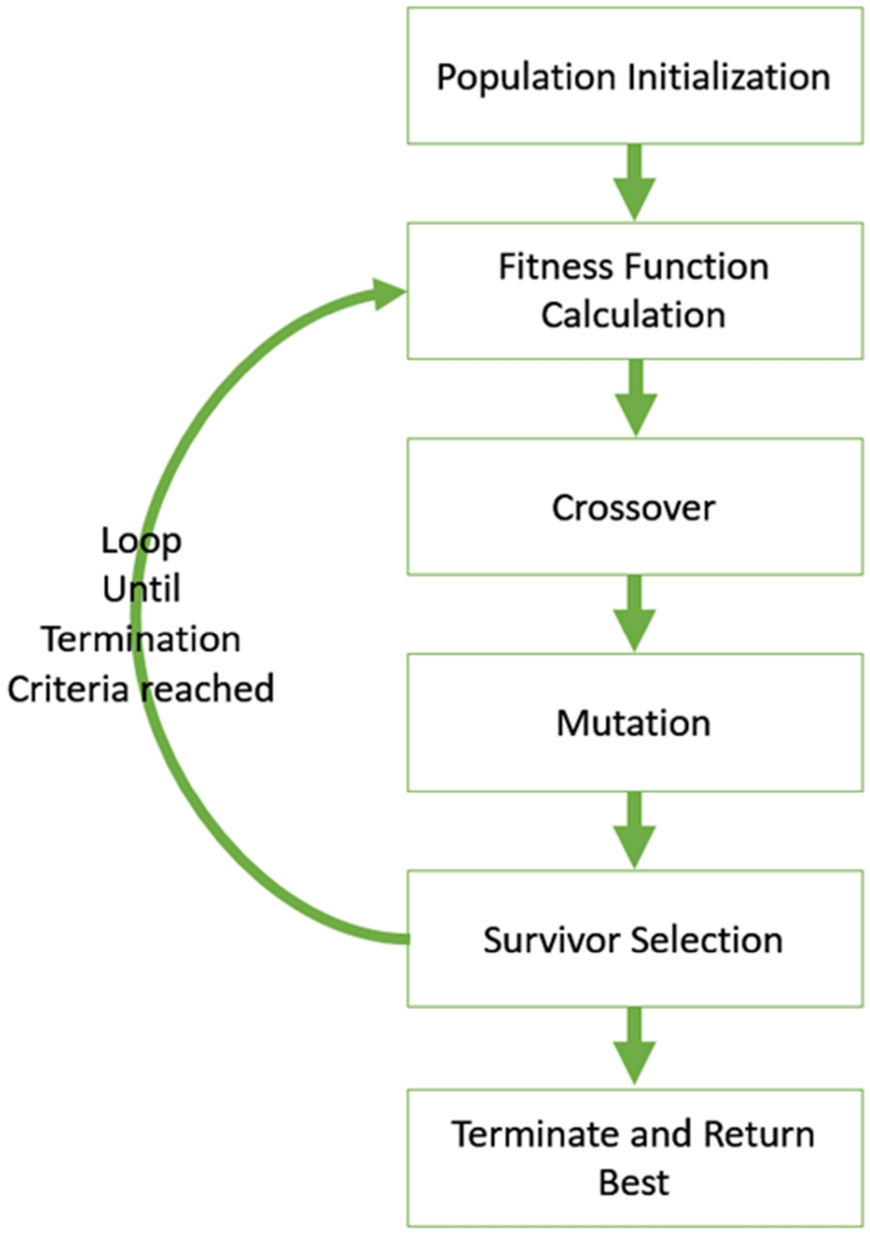


- Fitness Function − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- Genetic Operators − These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc. Basic Structure

The basic structure of a GA is as follows −

We start with an initial population which may be generated at random or seeded by other heuristics, select parents from this population for mating. 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. In this way genetic algorithms actually try to mimic the human evolution to some extent.

Each of the following steps are covered as a separate chapter later in this tutorial.


Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

A generalized pseudo-code for a GA is explained in the following program −

```
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
```

GENOTYPE REPRESENTATION

The GA Algorithm

A Genetic Algorithm can be implemented using the following outline algorithm

1. Initialise a population of chromosomes
2. Evaluate each chromosome (individual) in the population
    2.1. Create new chromosomes by mating chromosomes in the current population (using crossover and mutation)
    2.2. Delete members of the existing population to make way for the new members
    2.3. Evaluate the new members and insert them into the population
3. Repeat stage 2 until some termination condition is reached (normally based on time or number of populations produced)
4. Return the best chromosome as the solution.

This is the basic GA algorithm. As we shall see below there are many parameters that we can use to affect this basic algorithm.

One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

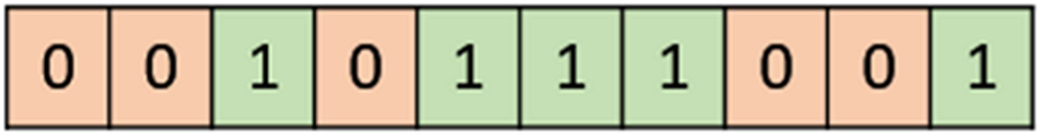
Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

In this section, we present some of the most commonly used representations for genetic algorithms. However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

Binary Representation

This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.

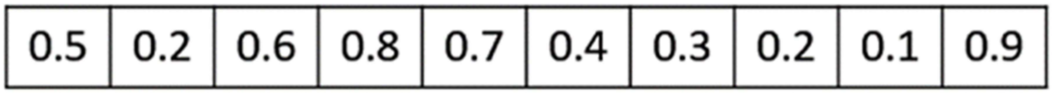
For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked 11 or not 00.



For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be resolved to some extent by using Gray Coding, as a change in one bit does not have a massive effect on the solution.

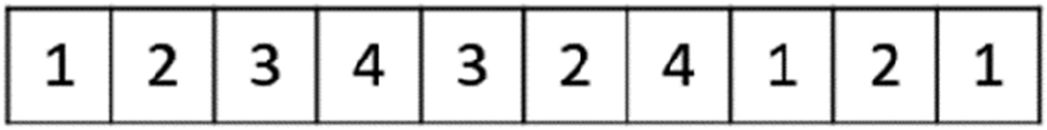
Real Valued Representation

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



Integer Representation

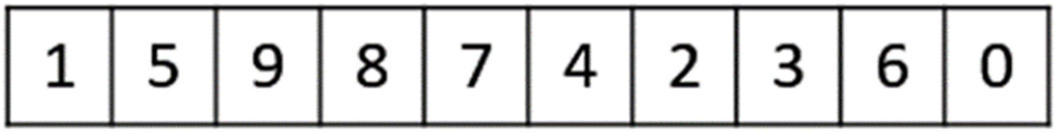
For discrete valued genes, we cannot always limit the solution space to binary ‘yes’ or ‘no’. For example, if we want to encode the four distances – North, South, East and West, we can encode them as {0,1,2,3}. In such cases, integer representation is desirable.



Permutation 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 TSPTSP. 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.



GENETIC ALGORITHMS - POPULATION

Population is a subset of solutions in the current generation. It can also be defined as a set of chromosomes. There are several things to be kept in mind when dealing with GA population −

- The diversity of the population should be maintained otherwise it might lead to premature convergence.
- The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

The population is usually defined as a two dimensional array of – size population, size x, chromosome size.

Population Initialization

There are two primary methods to initialize a population in a GA. They are −

- Random Initialization − Populate the initial population with completely random solutions.
- Heuristic initialization − Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

Population Models

There are two population models widely in use −

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 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" our how "good" the solution is with respect to the problem in consideration.

Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.

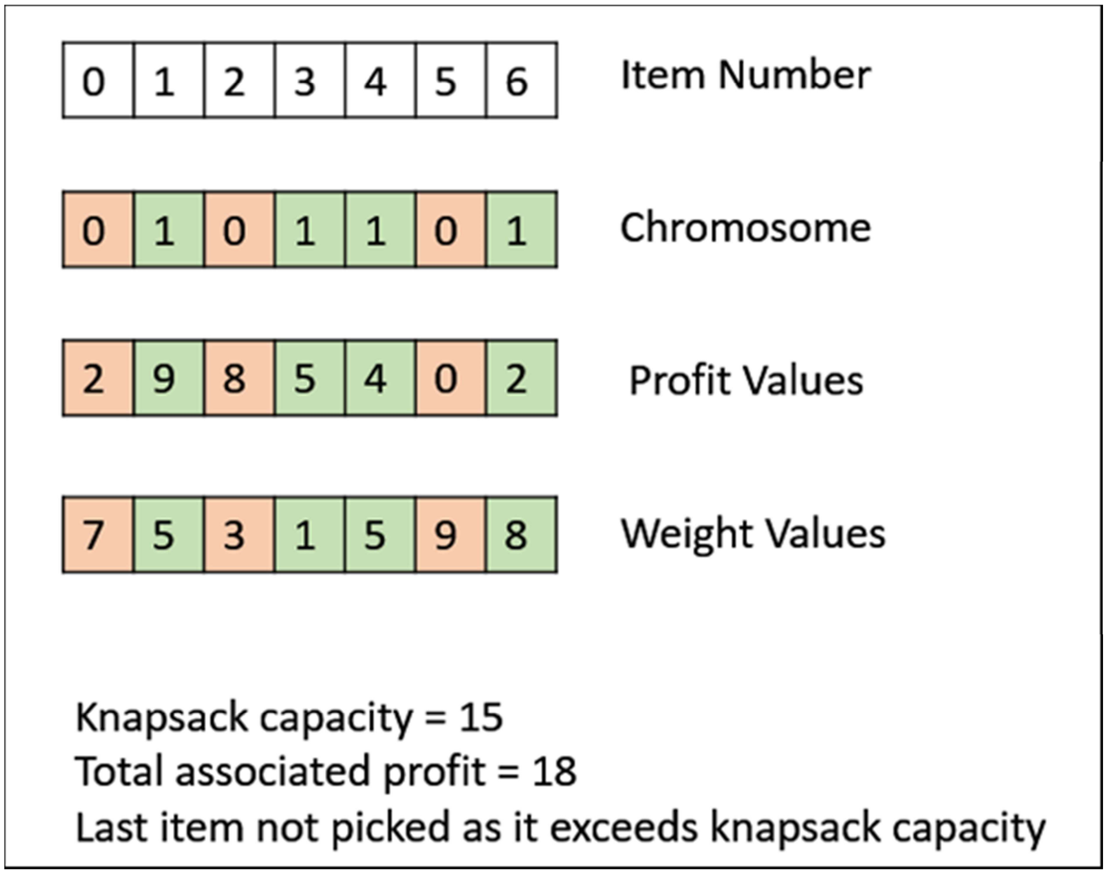
In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an Algorithm Designer might choose to have a different fitness function.

A fitness function should possess 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.

In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs.

The following image shows the fitness calculation for a solution of the 0/1 Knapsack. It is a simple fitness function which just sums the profit values of the items being picked whichhavea1whichhavea1, scanning the elements from left to right till the knapsack is full.



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.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. Maintaining good diversity in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as premature convergence and is an undesirable condition in a GA.

Fitness Proportionate Selection

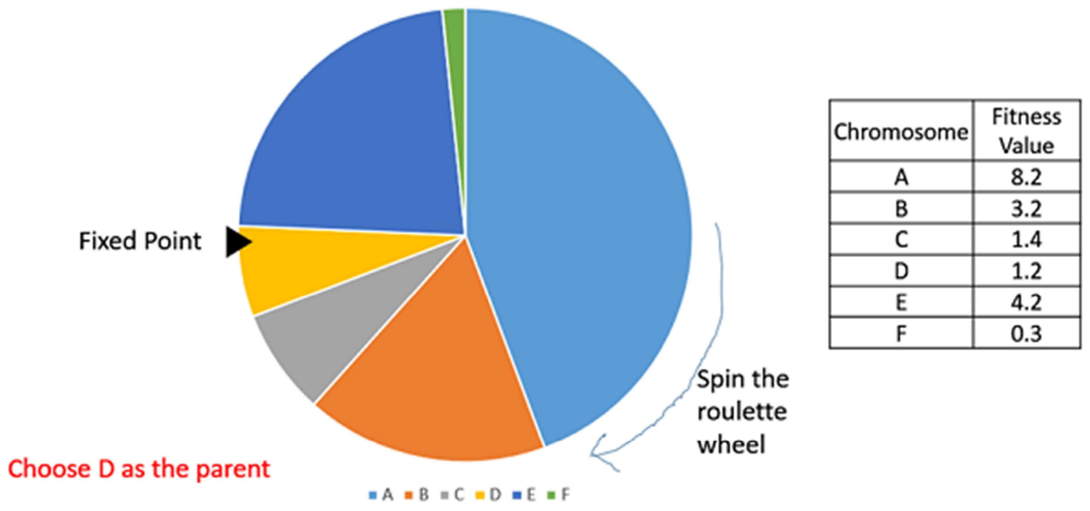
Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

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.

Two implementations of fitness proportionate selection are possible −

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

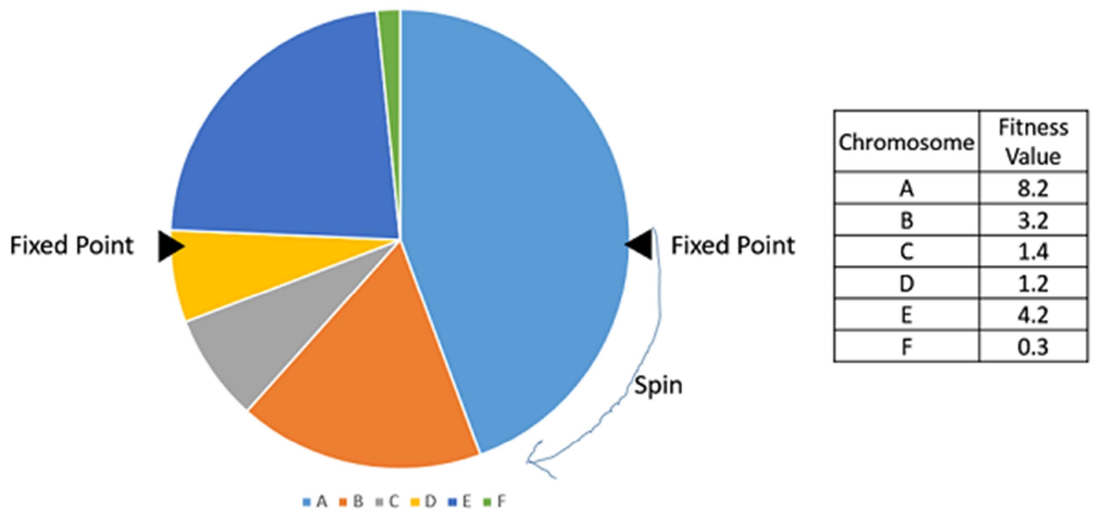
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

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.

Stochastic Universal Sampling SUSSUS

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 following 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.

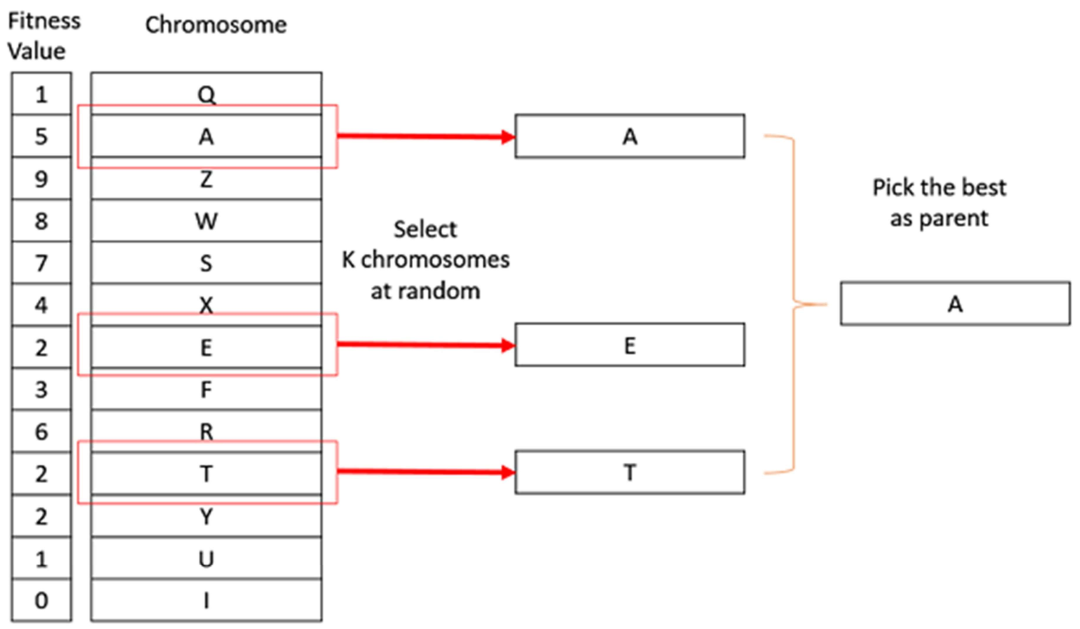


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

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

Tournament Selection

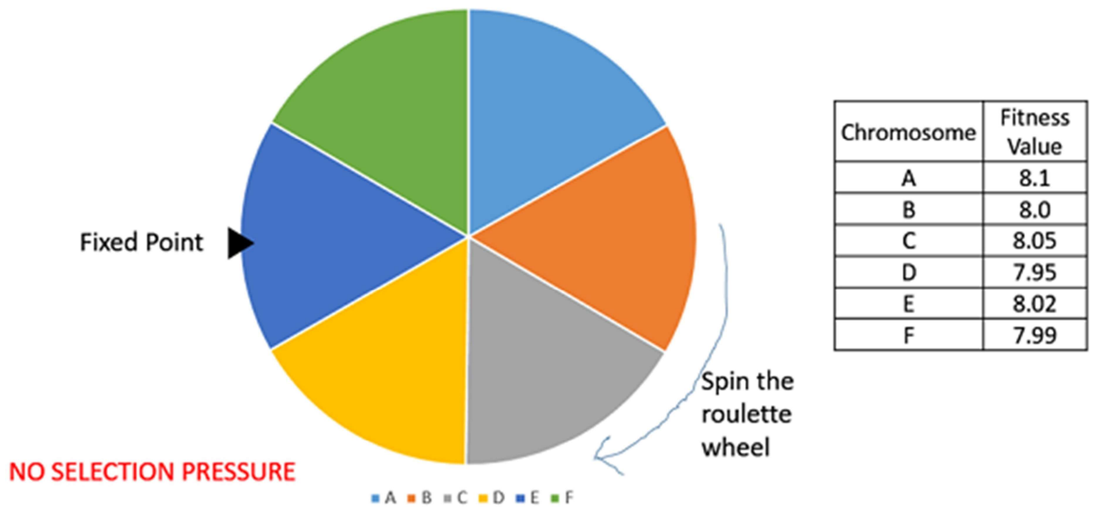
In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Fitness Value	Chromosome
1	Q
5	A
9	Z
8	W
7	S
4	X
2	E
3	F
6	R
2	T
2	Y
1	U
0	I

Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values this happens usually at the end of the run. This leads to each individual having an almost equal share of the pie like incase of fitness proportionate selection as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



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

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. 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	Rank
A	8.1	1
B	8.0	4

C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

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

In this chapter, we will discuss about what a Crossover Operator is along with its other modules, their uses and benefits.

Introduction to Crossover

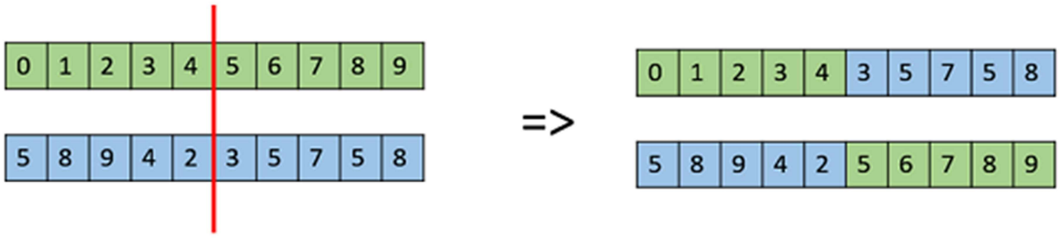
The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – p_c .

Crossover Operators

In this section we will discuss some of the most popularly used crossover operators. It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problemspecific crossover operator as well.

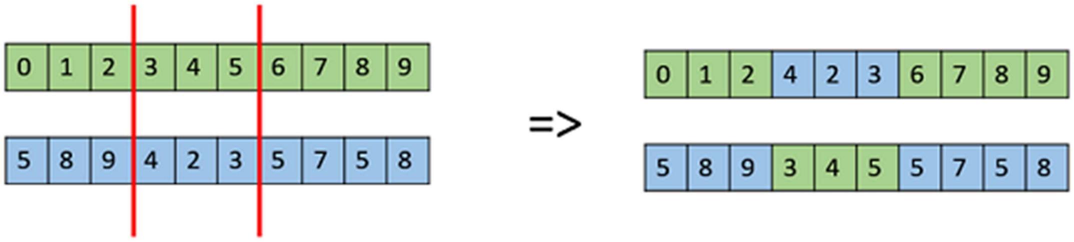
One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



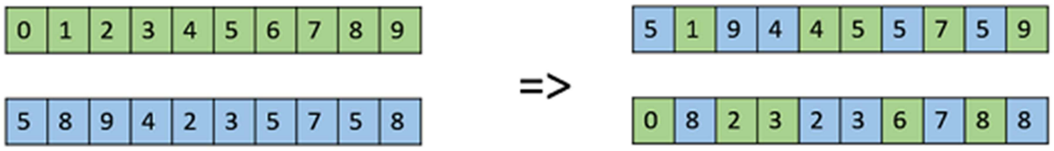
Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



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. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

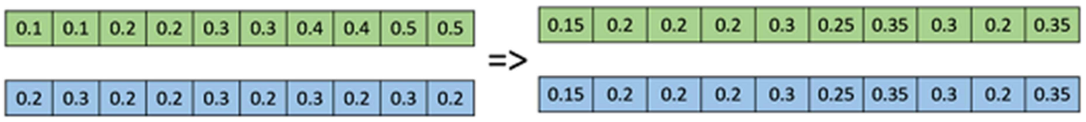


Whole Arithmetic Recombination

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−α1−α.y
- Child2 = α.x + 1−α1−α.y

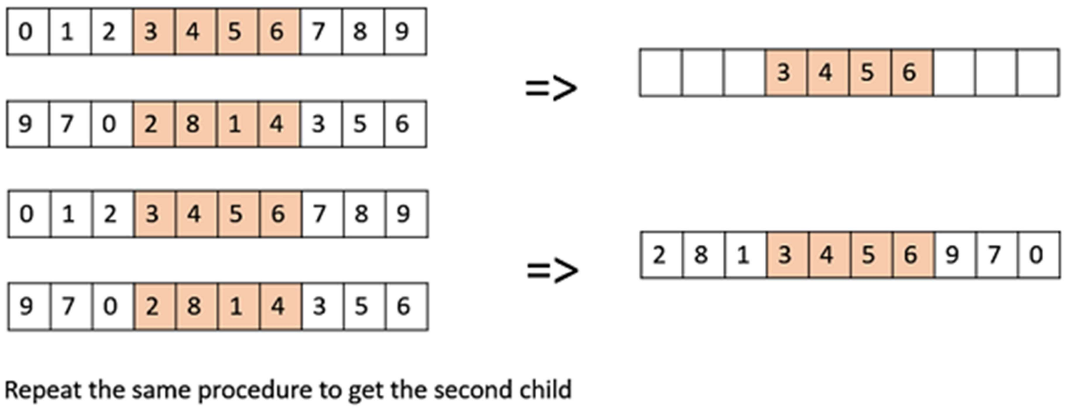
Obviously, if α = 0.5, then both the children will be identical as shown in the following image.



Davis' Order Crossover OX1OX1

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows −

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.



There exist a lot of other crossovers like Partially Mapped Crossover PMXPMX, Order based crossover OX2OX2, Shuffle Crossover, Ring Crossover, etc.

GENETIC ALGORITHMS - MUTATION

Introduction to Mutation

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m. If the probability is very high, the GA gets reduced to a random search.

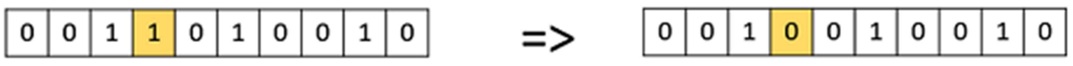
Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

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.

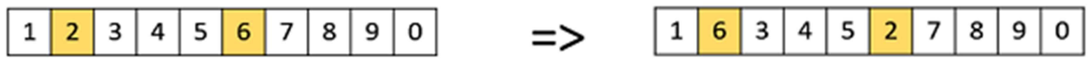


Random Resetting

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

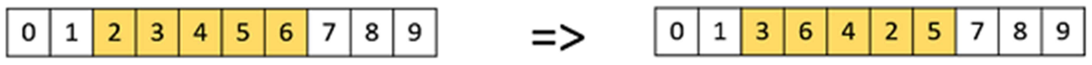
Swap Mutation

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



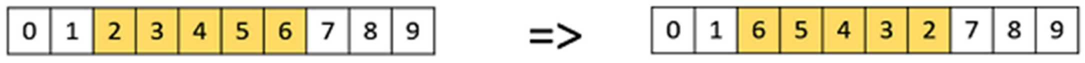
Scramble Mutation

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.



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. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

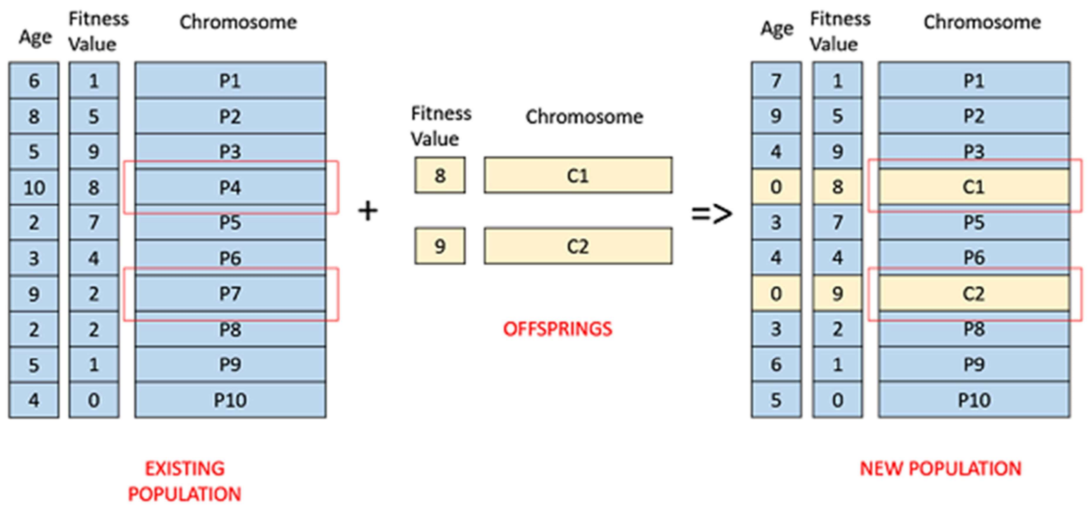
Some GAs employ Elitism. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

In Age-Based Selection, we don't have a notion of a fitness. It is 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.

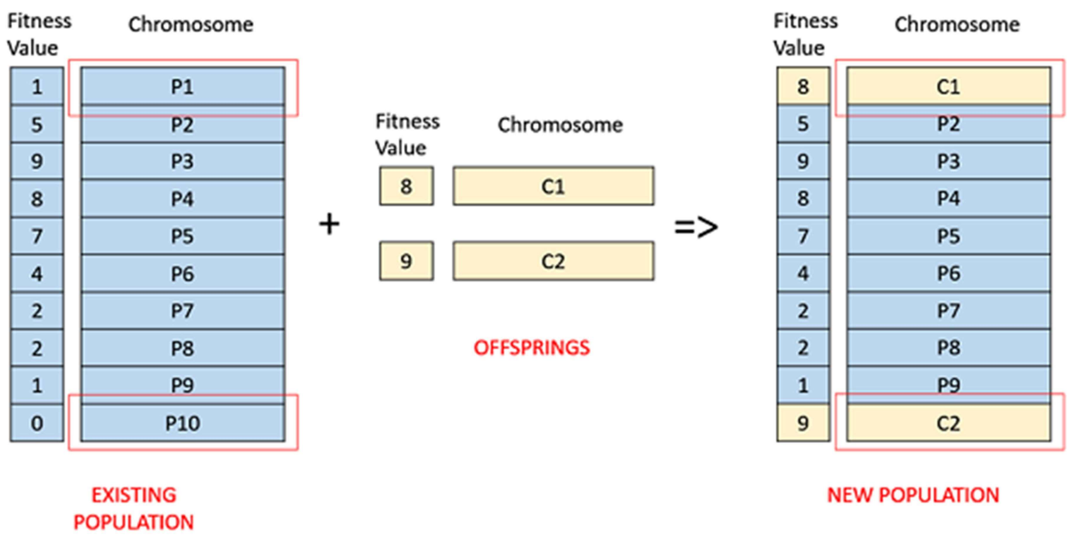
For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.



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 described before – tournament selection, fitness proportionate selection, etc.

For example, in the following image, the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.



GENETIC ALGORITHMS - TERMINATION CONDITION

The termination condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions −

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

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don’t generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

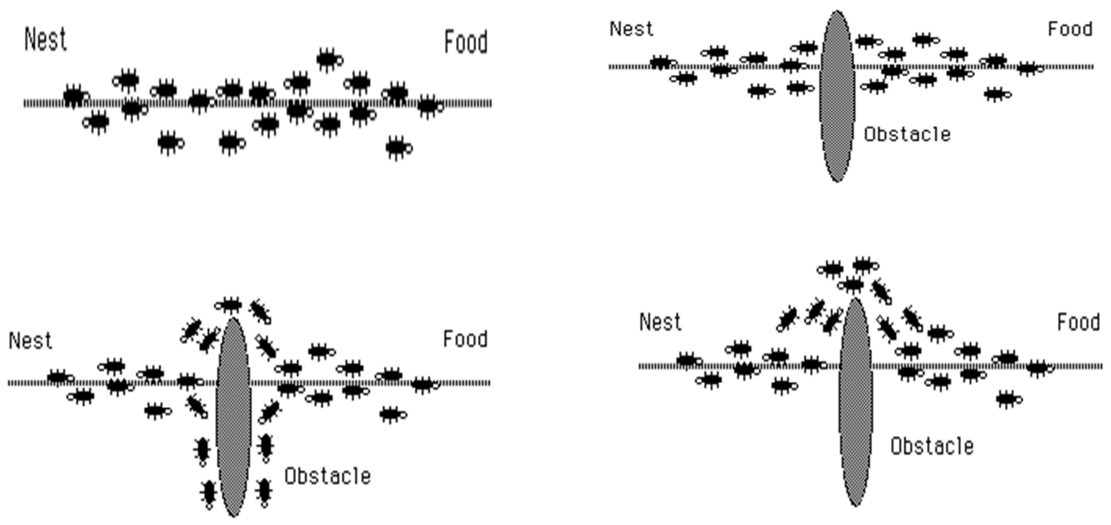
Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

Module V : Other Soft Computing Techniques

Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) studies artificial systems that take inspiration from the behavior of real ant colonies and which are used to solve discrete optimization problems.” First introduced by Marco Dorigo in 1992. Originally applied to Traveling Salesman Problem. Natural behavior of ants have inspired scientists to mimic insect operational methods to solve real-life complex optimization problems. By observing ant behavior, scientists have begun to understand their means of communication. Ant-based behavioral patterns to address combinatorial problems - first proposed by Marco Dorigo.

Ants secrete pheromone while traveling from the nest to food, and vice versa in order to communicate with one another to find the shortest path



Ants are forced to decide whether they should go left or right, and the choice that is made is a random decision. Pheromone accumulation is faster on the shorter path. The difference in pheromone content between the two paths over time makes the ants choose the shorter path.

The more ants follow a trail, the more attractive that trail becomes for being followed. Different optimization problems have been explored using a simulation of this real ant behavior

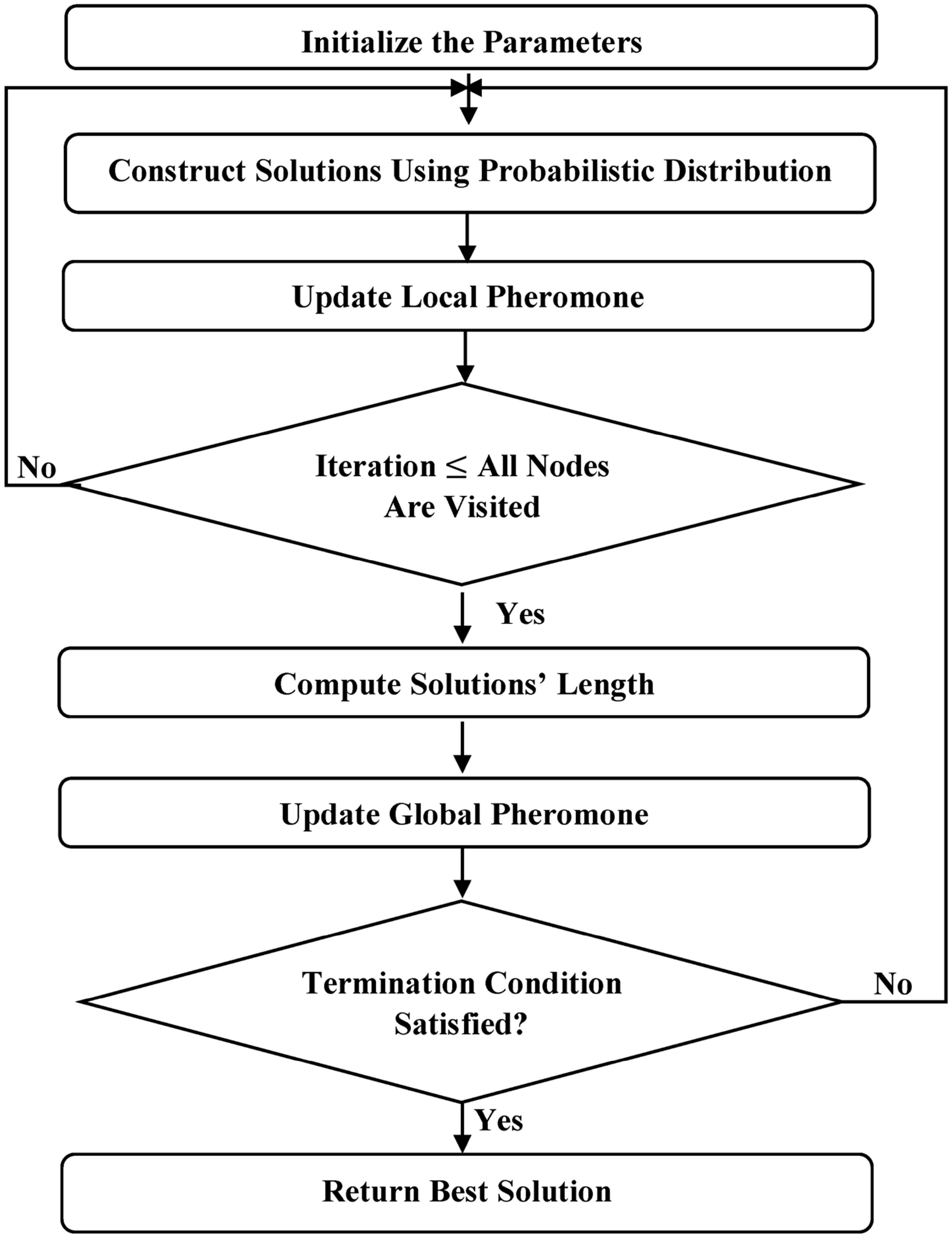

Initialize the Parameters
Construct Solutions Using Probabilistic Distribution
Update Local Pheromone
Iteration ≤ All Nodes Are Visited
No
Yes
Compute Solutions' Length
Update Global Pheromone
Termination Condition Satisfied?
No
Yes
Return Best Solution

Fig.1 ACO Algorithm

ACO Algorithm

1. Randomly place ants at the cities
2. For each ant:
3. Choose a not yet visited city until a tour is completed
4. optimize the tour
5. Update pheromone
6. Evaporate Pheromone
7. Update pheromone
8. Evaporate Pheromone

Advantages of the Ant Colony Optimization
1.Inherent parallelism
2.Positive Feedback accounts for rapid discovery of good solutions
3. Efficient for Traveling Salesman Problem and similar problems
4.Can be used in dynamic applications (adapts to changes such as new distances, etc)

Disadvantages of the Ant Colony Optimization
1.Theoretical analysis is difficult
2.Sequences of random decisions (not independent)
3.Probability distribution changes by iteration
4.Research is experimental rather than theoretical
5.Time to convergence uncertain (but convergence is guaranteed!)

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a technique used to explore the search space of a given problem to find the settings or parameters required to maximize a particular objective. This technique, first described by James Kennedy and Russell C. Eberhart in 1995 [1], originates from two separate concepts: the idea of swarm intelligence based off the observation of swarming habits by certain kinds of animals (such as birds and fish); and the field of evolutionary computation.

This short tutorial first discusses optimization in general terms, then describes the basics of the particle swarm optimization algorithm.

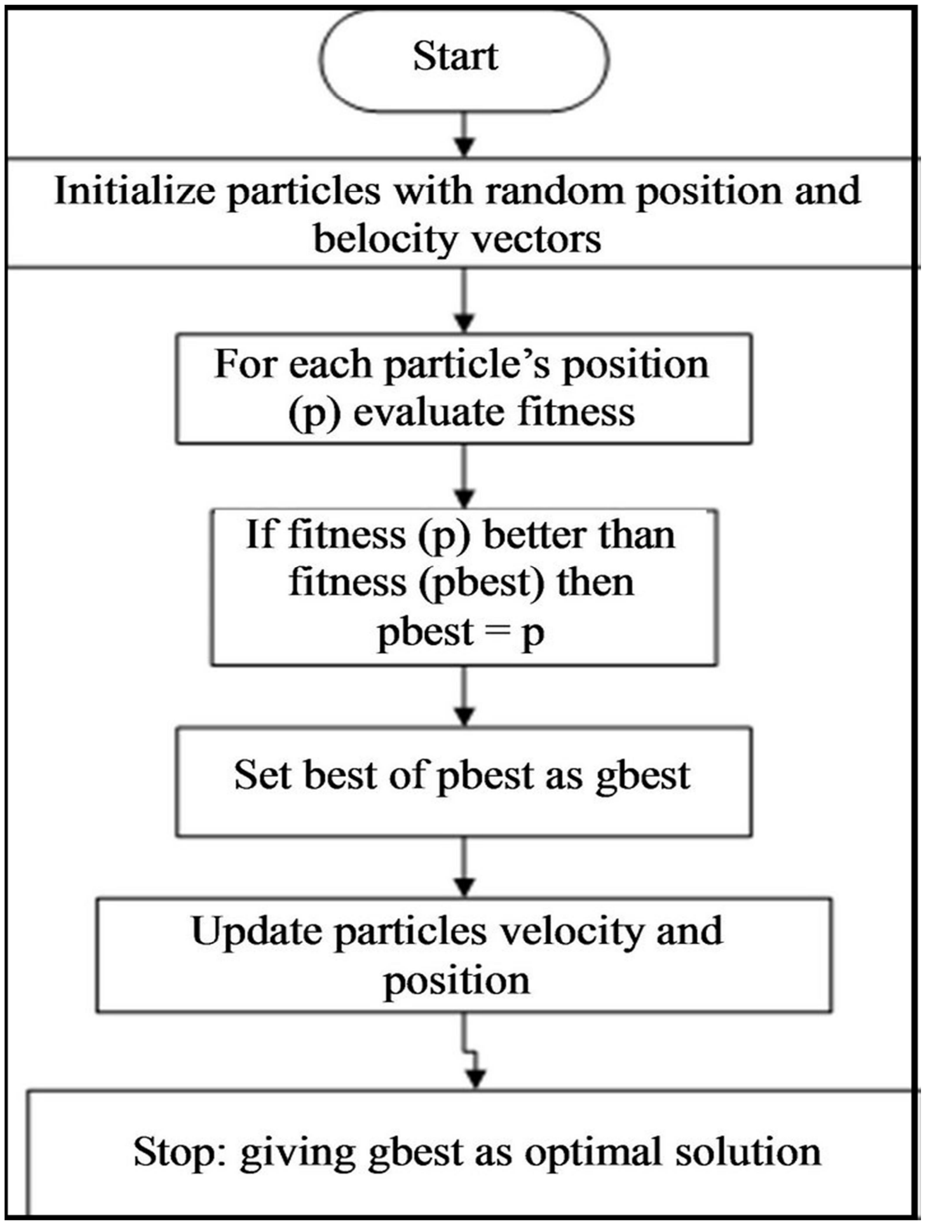

Start
Initialize particles with random position and belocity vectors
For each particle's position (p) evaluate fitness
If fitness (p) better than fitness (pbest) then pbest = p
Set best of pbest as gbest
Update particles velocity and position
Stop: giving gbest as optimal solution

Fig.1 PSO Algorithm

The PSO algorithm works by simultaneously maintaining several candidate solutions in the search space. During each iteration of the algorithm, each candidate solution is evaluated by the objective function being optimized, determining the fitness of that solution. Each candidate solution can be thought of as a particle "flying" through the fitness landscape finding the maximum or minimum of the objective function.
Initially, the PSO algorithm chooses candidate solutions randomly within the search space. Figure 2 shows the initial state of a four-particle PSO algorithm seeking the global maximum in a one-dimensional search space. The search space is composed of all the possible solutions along the x-axis; the curve denotes the objective function. It should be noted that the PSO algorithm has no knowledge of the underlying objective function, and thus has no way of knowing if any of the candidate solutions are near to or far away from a local

or global maximum. The PSO algorithm simply uses the objective function to evaluate its candidate solutions, and operates upon the resultant fitness values.

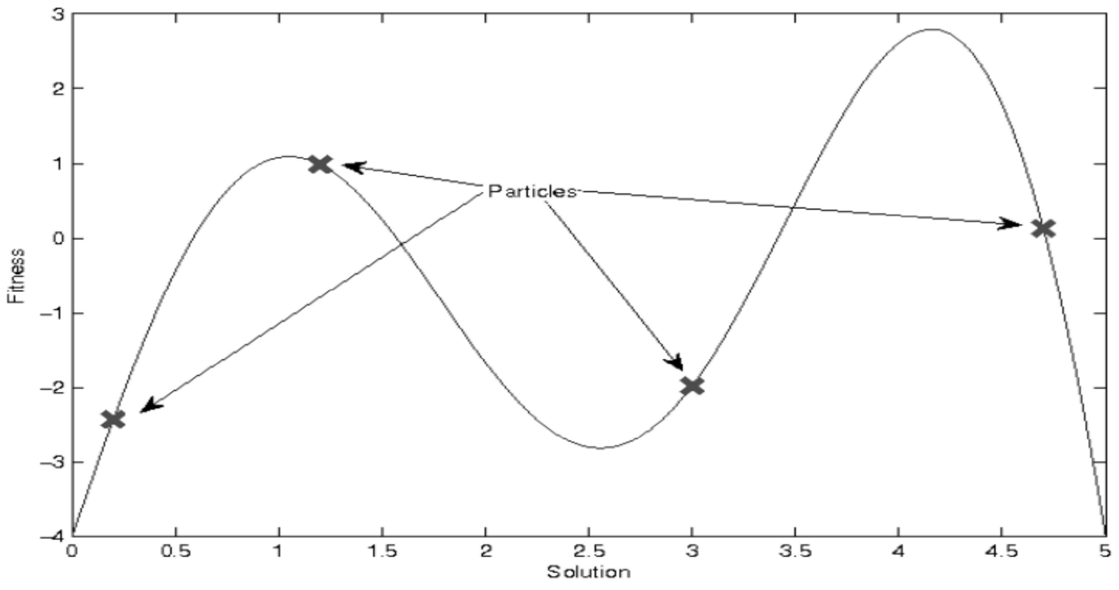


Fig.2 Initial PSO state

Each particle maintains its position, composed of the candidate solution and its evaluated fitness, and its velocity. Additionally, it remembers the best fitness value it has achieved thus far during the operation of the algorithm, referred to as the individual best fitness, and the candidate solution that achieved this fitness, referred to as the individual best position or individual best candidate solution. Finally, the PSO algorithm maintains the best fitness value achieved among all particles in the swarm, called the global best fitness, and the candidate solution that achieved this fitness, called the global best position or global best candidate solution.

The PSO algorithm consists of just three steps, which are repeated until some stopping condition is met:

1. Evaluate the fitness of each particle
2. Update individual and global best fitness and positions
3. Update velocity and position of each particle

The first two steps are fairly trivial. Fitness evaluation is conducted by supplying the candidate solution to the objective function. Individual and global best fitness and positions are updated by comparing the newly evaluated fitness against the previous individual and global best fitness, and replacing the best fitness and positions as necessary.

Advantages of the basic particle swarm optimization algorithm:
(1)PSO is based on the intelligence. It can be applied into both scientific research and engineering use.
(2)PSO have no overlapping and mutation calculation. The search can be carried out by the speed of the particle. During the development of several generations, only the most optimist particle can transmit information onto the other particles, and the speed of the researching is very fast.
(3)The calculation in PSO is very simple. Compared with the other developing calculations, it occupies the bigger optimization ability and it can be completed easily.
(4) PSO adopts the real number code, and it is decided directly by the solution. The number of the dimension is equal to the constant of the solution.

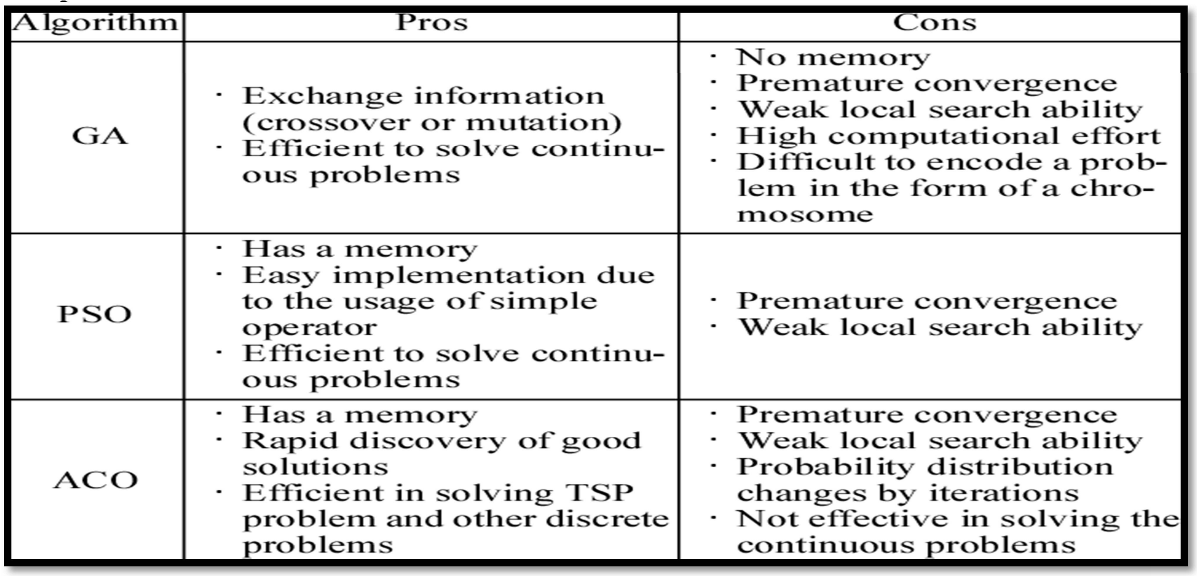
Disadvantages of the basic particle swarm optimization algorithm:
(1)The method easily suffers from the partial optimism, which causes the less exact at the regulation of its speed and the direction.
(2)The method cannot work out the problems of scattering and optimization.
(3)The method cannot work out the problems of non-coordinate system, such as the solution to the energy field and the moving rules of the particles in the energy field

APPLICATIONS OF ACO AND PSO

Ant colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to fold protein or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. It has also been used to produce near-optimal solutions to the travelling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches of similar problems when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems.

The first practical application of PSO was in the field of neural network training and was reported together with the algorithm itself (Kennedy and Eberhart 1995). Many more areas of application have been explored ever since, including telecommunications, control, data mining, design, combinatorial optimization, power systems, signal processing, and many others. To date, there are hundreds of publications reporting applications of particle swarm optimization algorithms. For a review, see (Poli 2008). Although PSO has been used mainly to solve unconstrained, single-objective optimization.

Comparison of GA, PSO and ACO



Algorithm	Pros	Cons
GA	· Exchange information (crossover or mutation) · Efficient to solve continuous problems	· No memory · Premature convergence · Weak local search ability · High computational effort · Difficult to encode a problem in the form of a chromosome
PSO	· Has a memory · Easy implementation due to the usage of simple operator · Efficient to solve continuous problems	· Premature convergence · Weak local search ability
ACO	· Has a memory · Rapid discovery of good solutions · Efficient in solving TSP problem and other discrete problems	· Premature convergence · Weak local search ability · Probability distribution changes by iterations · Not effective in solving the continuous problems

Sample Questions

1. What is meant by servival of the fittest? Explain.
2. What is genotype and phenotype? Explain.

3. Write down the steps of Genetic Algorithm.
4. Write down the flowchart of Genetic Algorithm.
5. What is fitness function?
6. Explain the term Selection, Crossover and Mutation.
7. Write down the steps/flowchart of ACO.
8. Write down the steps/flowchart of PSO.
9. Write down the advatages and disadvantages of ACO.
10. Write down the advatages and disadvantages of PSO.
11. Write down the application of ACO.
12. Write down the application of PSO.