Genetic Algorithms

1.Introduction

Charles Darwin stated the theory of natural evolution in the origin of species. Over several generations, biological organisms evolve based on the principle of natural selection "survival of the fittest" to reach certain remarkable tasks.

In nature, an individual in population competes with each other for virtual resources like food, shelter and so on. Also in the same species, individuals compete to attract mates for reproduction.

Due to this selection, poorly performing individuals have less chance to survive, and the most adapted or "fit" individuals produce a relatively large number of offspring's. It can also be noted that during reproduction, a recombination of the good characteristics of each ancestor can produce "best fit" offspring whose fitness is greater than that of a parent. After a few generations, species evolve spontaneously to become more and more adapted to their environment.

In 1975, Holland developed this idea in his book "Adaptation in natural and artificial systems". He described how to apply the principles of natural evolution to optimization problems and built the first Genetic Algorithms. Holland's theory has been further developed and now Genetic Algorithms (GAs) stand up as a powerful tool for solving search and optimization problems. Genetic algorithms are based on the principle of genetics and evolution.

The power of mathematics lies in technology transfer: there exist certain models and methods, which describe many different phenomena and solve wide variety of problems. GAs are an example of mathematical technology transfer: by simulating evolution one can solve optimization problems from a variety of sources. Today, GAs are used to resolve complicated optimization problems, like, timetabling, job shop scheduling, games playing.

2. A Simple Genetic Algorithm

An algorithm is a series of steps for solving a problem. A genetic algorithm is a problem solving method that uses genetics as its model of problem solving. It's a search technique to find approximate solutions to optimization and search problems.

Basically, an optimization problem looks really simple.

GA handles a population of possible solutions. Each solution is represented through a chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of a Genetic Algorithm. A set of reproduction operators has to be determined, too. Reproduction operators are applied directly on the chromosomes, and are used to perform mutations and recombination over solutions of the problem. Appropriate representation and reproduction operators are really something determinant, as the behavior of the GA is extremely dependant on it. Frequently, it can be extremely difficult to find a representation, which respects the structure of the search space and reproduction operators, which are coherent and relevant according to the properties of the problems. Selection is supposed to be able to compare each individual in the population. Selection is done by using a fitness function. Each chromosome has an associated value corresponding to the fitness of the solution it represents. The fitness should correspond to an evaluation of how good the candidate solution is. The optimal solution is the one, which maximizes the fitness function. Genetic Algorithms deal with the problems that maximize the fitness function. But, if the problem consists in minimizing a cost function, the adaptation is quite easy. Either the cost function can be transformed into a fitness function, for example by inverting it; or the selection can be adapted in such way that they consider individuals with low evaluation
functions as better. Once the reproduction and the fitness function have been properly defined, a Genetic Algorithm is evolved according to the same basic structure. It starts by generating an initial population of chromosomes. This first population must offer
a wide diversity of genetic materials. The gene pool should be as large as possible so that any solution of the search space can be engendered. Generally, the initial population is generated randomly. Then, the genetic algorithm loops over an iteration process to make the population evolve.

Each iteration consists of the following steps:

- *SELECTION: The first step consists in selecting individuals for reproduction. This selection is done randomly with a probability depending on the relative fitness of the individuals so that best ones are often chosen for reproduction than poor ones.*
- *REPRODUCTION: In the second step, offspring are bred by the selected individuals. For generating new chromosomes, the algorithm can use both recombination and mutation.*
- *EVALUATION: Then the fitness of the new chromosomes is evaluated.*
- *REPLACEMENT: During the last step, individuals from the old population are killed and replaced by the new ones.*

The algorithm is stopped when the population converges toward the optimal solution.

The basic genetic algorithm is as follows:

- *[start] Genetic random population of n chromosomes (suitable solutions for the problem)*
- *[Fitness] Evaluate the fitness f(x) of each chromosome x in the population*
- *New population] Create a new population by repeating following steps until the New population is complete*

- [selection] select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to get selected).

- [crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

- [Mutation] With a mutation probability, mutate new offspring at each locus(position in chromosome)

- *[Accepting] Place new offspring in the new population.*

- *[Replace] Use new generated population for a further sum of the algorithm.*
- *[Test] If the end condition is satisfied, stop, and return the best solution in current population.*

- *[Loop] Go to step2 for fitness evaluation.*

Based on the foregoing discussion, the important criteria for GA approach can be formulated as given below:

- *Completeness: Any solution should have its encoding*
- *Non redundancy: Codes and solutions should correspond one to one*
- *Soundness: Any code (produced by genetic operators) should have its corresponding solution*
-*Characteristic perseverance: Offspring should inherit useful characteristics from parents.*

3.Comparison of Genetic Algorithm with Other Optimization Techniques

Genetic algorithm differs from conventional optimization techniques in following ways:
1. GAs operate with coded versions of the problem parameters rather than parameters themselves i.e., GA works with the coding of solution set and not with the solution itself.
2. Almost all conventional optimization techniques search from a single point but GAs always operate on a whole population of points(strings) i.e., GA uses population of solutions rather than a single solution fro searching. This plays a major role to the robustness of genetic algorithms. It improves the chance of reaching the global optimum and also helps in avoiding local stationary point.
3. GA uses fitness function for evaluation rather than derivatives. As a result, they can be applied to any kind of continuous or discrete optimization problem. The key point to be performed here is to identify and specify a meaningful decoding function.
4. GAs use probabilistic transition operates while conventional methods for continuous optimization apply deterministic transition operates i.e., GAs does not use deterministic rules.

4.Applications of Genetic Algorithm

A few applications of GA are as follows:
• Nonlinear dynamical systems–predicting, data analysis
• Robot trajectory planning

- Evolving LISP programs (genetic programming)
- Strategy planning
- Finding shape of protein molecules
- TSP and sequence scheduling
- Functions for creating images
- Control–gas pipeline, pole balancing, missile evasion, pursuit
- Design–semiconductor layout, aircraft design, keyboard configuration, communication networks
- Scheduling–manufacturing, facility scheduling, resource allocation
- Machine Learning–Designing neural networks, both architecture and weights ,improving classification algorithms, classifier systems
- Signal Processing–filter design
- Combinatorial Optimization–set covering, traveling salesman (TSP), Sequence scheduling, routing, bin packing, graph coloring and partitioning

Individuals

An individual is a single solution. A chromosome is subdivided into genes. A gene is the GA's representation of a single factor for a control factor. Each factor in the solution set corresponds to gene in the chromosome. Chromosomes are encoded by bit strings are given below in Fig.1 Genes are the basic "instructions" for building a Generic Algorithms. A chromosome is a sequence of genes. Genes may describe a possible solution to a problem.

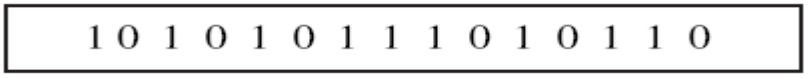

Fig.1 chromosome

Fitness

The fitness of an individual in a genetic algorithm is the value of an objective function . For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.

5.1 Populations

A population is a collection of individuals. A population consists of a number of individuals being tested.

The two important aspects of population used in Genetic Algorithms are:

1. The initial population generation.
2. The population size.

Population	Chromosome 1	1 1 1 0 0 0 1 0
	Chromosome 2	0 1 1 1 1 0 1 1
	Chromosome 3	1 0 1 0 1 0 1 0
	Chromosome 4	1 1 0 0 1 1 0 0

Fig.2 Population

5.2.Encoding

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers.

5.2.1 Binary Encoding

The most common way of encoding is a binary string, which would be represented as in Fig. 3 Each chromosome encodes a binary (bit) string. Each bit in the string can represent some characteristics of the solution. Every bit string therefore is a solution but not necessarily the best solution. Another possibility is that the whole string.

Chromosome 1	1 1 0 1 0 0 0 1 1 0 1 0
Chromosome 2	0 1 1 1 1 1 1 1 1 1 0 0

Fig.3 binary encoding

5..2.2 *Octal Encoding*

This encoding uses string made up of octal numbers (0–7).

Chromosome 1	03467216
Chromosome 2	15723314

Fig.4 octal encoding

5.2.3 Hexadecimal Encoding

This encoding uses string made up of hexadecimal numbers (0–9, A–F).

Chromosome 1	9CE7
Chromosome 2	3DBA

Fig.5 hexa encoding

5.2.4 Permutation Encoding (Real Number Coding)

permutation encoding, every chromosome is a string of integer/real values, which represents number in a sequence.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Fig.6 permutation encoding

5.2.5 Value Encoding

Every chromosome is a string of values and the values can be anything connected to the problem. This encoding produces best results for some special problems. On the other hand, it is often necessary to develop new genetic operator's specific to the
problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult.
In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.
Value encoding is very good for some special problems. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Fig.7 value encoding

Breeding

The breeding process is the heart of the genetic algorithm. It is in this process, the search process creates new and hopefully fitter individuals. The breeding cycle consists of three steps:

a. Selecting parents.
b. Crossing the parents to create new individuals (offspring or children).
c. Replacing old individuals in the population with the new ones.

5.3 Selection

Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection i.e., how to choose individuals in the population that will create offspring for the

next generation and how many offspring each will create. The purpose of selection is to emphasize fitter individuals in the population in hopes that their off springs have higher fitness. Chromosomes are selected from the initial population to be parents for reproduction. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offspring.

5.3 .1 Roulette Wheel Selection

Roulette selection is one of the traditional GA selection techniques. The commonly used reproduction operator is the proportionate reproductive operator where a string is selected from the mating pool with a probability proportional to the fitness. The principle of roulette selection is a linear search through a roulette wheel with the slots in the wheel weighted in proportion to the individual's fitness values. A target value is set, which is a random proportion of the sum of the fit nesses in the population. The population is stepped through until the target value is reached. This is only
a moderately strong selection technique, since fit individuals are not guaranteed to be selected for, but somewhat have a greater chance. A fit individual will contribute more to the target value, but if it does not exceed it, the next chromosome in line has a chance, and it may be weak. It is essential that the population not be sorted by fitness, since this would dramatically bias the selection.
The above described Roulette process can also be explained as follows: The expected value of an individual is that fitness divided by the actual fitness of the population. Each individual is assigned a slice of the roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is
the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation.

This method is implemented as follows:

1. Sum the total expected value of the individuals in the population.
   Let it be T.
2. Repeat N times:
   i. Choose a random integer 'r' between o and T.
   ii. Loop through the individuals in the population, summing the

expected values, until the sum is greater than or equal to 'r'. The individual whose expected value puts the sum over this limit is the one selected. Roulette wheel selection is easier to implement but is noisy. The rate of evolution depends on the variance of fitness's in the population.

5.3 .2 Random Selection

This technique randomly selects a parent from the population. In terms of disruption of genetic codes, random selection is a little more disruptive, on average, than roulette wheel selection.

5.3 .3 Rank Selection

The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population and every chromosome receives fitness from the ranking. The worst has fitness 1 and the best has fitness N. It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the
fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent. There are many ways this can be achieved and
two suggestions are,

1. Select a pair of individuals at random. Generate a random number, *R*, between 0 and 1. If $R < r$ use the first individual as a parent. If the $R >= r$ then use the second individual as the parent. This is repeated to select the second parent. The value of *r* is a parameter to this method.
2. Select two individuals at random. The individual with the highest

evaluation becomes the parent. Repeat to find a second parent.

5.3 .4 Tournament Selection

An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-tune GA search performance. Unlike, the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among Nu individuals. The best individual from the tournament is the one with the highest fitness, which is the winner of Nu. Tournament competitions and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes.This method is more efficient and leads to an optimal solution.

5.4 Crossover (Recombination)

Crossover is the process of taking two parent solutions and producing from them a child. After the selection (reproduction) process, the population is enriched with better individuals. Reproduction makes clones of good strings but does not create
new ones. Crossover operator is applied to the mating pool with the hope that it creates a better offspring. Crossover is a recombination operator that proceeds in three steps:

i. The reproduction operator selects at random a pair of two individual strings for the mating.
ii. A cross site is selected at random along the string length.
iii. Finally, the position values are swapped between the two strings following the cross site.

The various crossover techniques are discussed as follows:

5.4 .1 Single Point Crossover

The traditional genetic algorithm uses single point crossover, where the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged. Here, a cross-site or crossover point is selected randomly along the length

of the mated strings and bits next to the cross-sites are exchanged. If appropriate site is chosen, better children can be obtained by combining good parents else it severely hampers string quality. The following Fig.8 illustrates single point crossover and it can be observed that the bits next to the crossover point are exchanged to produce children. The crossover point can be chosen randomly.

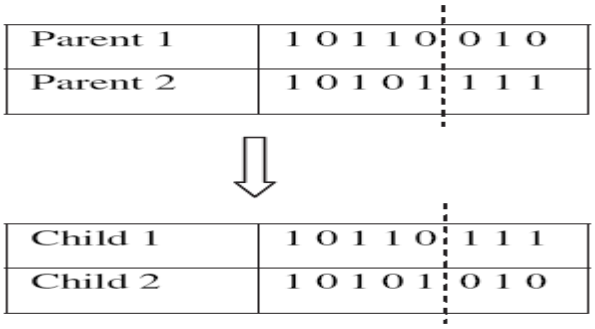

Fig.8 *Single Point Crossover*

5.4.2 Two Point Crossover

Apart from single point crossover, many different crossover algorithms have been devised, often involving more than one cut point. It should be noted that adding further crossover points reduces the performance of the GA. The problem with adding
additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly. In two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents.
In the following Fig. 9 the dotted lines indicate the crossover points. Thus the contents between these points are exchanged between the parents to produce new children for mating in the next generation.

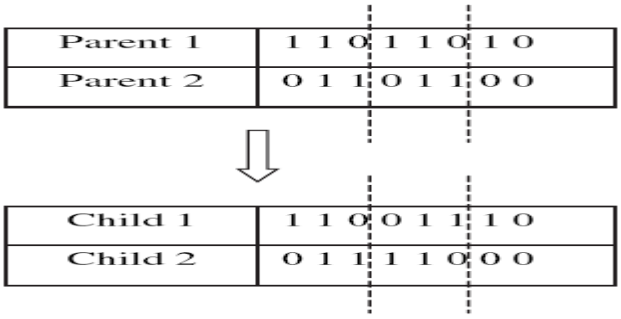

Fig.9 two *Point Crossover*

5.4.3 Multi-Point Crossover (N-Point crossover)

There are two ways in this crossover. One is even number of cross-sites and the other odd number of cross-sites. In the case of even number of cross-sites, cross-sites are selected randomly around a circle and information is exchanged. In the case of odd number of cross-sites, a different cross-point is always assumed at the string beginning.

5.4.4 Uniform Crossover

Uniform crossover is quite different from the N-point crossover. Each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offsprings, therefore contain a mixture of genes from each parent. The number of effective crossing point is not fixed, but will average L/2 (where L is the chromosome length).In Fig. 10, new children are produced using uniform crossover approach. It can be noticed, that while producing child 1, when there is a 1 in the mask, the gene is copied from the parent 1 else from the parent 2. On producing child 2, when there is a 1 in the mask, the gene is copied from parent 2, when there is a 0 in the mask; the gene is copied from the parent 1.

Parent 1	1 0 1 1 0 0 1 1
Parent 2	0 0 0 1 1 0 1 0
Mask	1 1 0 1 0 1 1 0
Child 1	1 0 0 1 1 0 1 0
Child 2	0 0 1 1 0 0 1 1

Fig.10 uniform *Crossover*

5.4.5 Three Parent Crossover

In this crossover technique, three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent. If both are the same, the bit is taken for the offspring otherwise; the bit from the third parent is taken for the
offspring. This concept is illustrated in Fig.11

Parent 1	1 1 0 1 0 0 0 1
Parent 2	0 1 1 0 1 0 0 1
Parent 3	0 1 1 0 1 1 0 0
Child	0 1 1 0 1 0 0 1

Fig.11 three parent *Crossover*

5.4.6 Precedence Preservative Crossover (PPX)

PPX was independently developed for vehicle routing problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedence relations of operations given in two parental permutations
to one offspring at the same rate, while no new precedence relations are introduced. PPX is illustrated in below, for a problem consisting of six operations A–F.

The operator works as follows:

- A vector of length Sigma, sub i=1to mi, representing the number of operations involved in the problem, is randomly filled with elements of the set {1, 2}.
- This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.
- We can also consider the parent and offspring permutations as lists, for which the operations 'append' and 'delete' are defined.
- First we start by initializing an empty offspring.
- The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
- After an operation is selected it is deleted in both parents.
- Finally the selected operation is appended to the offspring.
- This step is repeated until both parents are empty and the offspring contains all operations involved.

• Note that PPX does not work in a uniform-crossover manner due to the 'deletion append' scheme used.
Example is shown in Fig.12

Parent permutation 1	A	B	C	D	E	F
Parent permutation 2	C	A	B	F	D	E
Select parent no. (1/2)	1	2	1	1	2	2
Offspring permutation	A	C	B	D	F	E

Fig.12 PPX *Crossover*

5.4.7 Ordered Crossover

Ordered two-point crossover is used when the problem is of order based, for example in U-shaped assembly line balancing etc. Given two parent chromosomes, two random crossover points are selected partitioning them into a left, middle and right portion. The ordered two-point crossover behaves in the following way: child 1
inherits its left and right section from parent 1, and its middle section is determined

by the genes in the middle section of parent 1 in the order in which the values appear in parent 2. A similar process is applied to determine child 2. This is shown in Fig.13

Parent 1 : 4 2 | 1 3 | 6 5	Child 1 : 4 2 | 3 1 | 6 5
Parent 2 : 2 3 | 1 4 | 5 6	Child 2 : 2 3 | 4 1 | 5 6

Fig.13 ordered *Crossover*

5.4.8 PartiallyMatched Crossover (PMX)

PMX can be applied usefully in the TSP. Indeed, TSP chromosomes are simply sequences of integers, where each integer represents a different city and the order represents the time at which a city is visited. Under this representation, known as
permutation encoding, we are only interested in labels . It may be viewed as a crossover of permutations that guarantees that all positions are found exactly once in each offspring, i.e. both offspring receive a full complement of genes, followed by the corresponding filling in of alleles from their parents.

PMX proceeds as follows:

1. The two chromosomes are aligned.
2. Two crossing sites are selected uniformly at random along the strings, defining a matching section
• The matching section is used to effect a cross through position-by-position exchange operation
• Alleles are moved to their new positions in the offspring
• The following illustrates how PMX works.
• Consider the two strings shown in Fig. 3.14
• Where, the dots mark the selected cross points.
• The matching section defines the position-wise exchanges that must take place in both parents to produce the offspring.
• The exchanges are read from the matching section of one chromosome to that of the other.
• In the example, the numbers that exchange places are 5 and 2, 6 and 3, and 7 and 10.
• The resulting offspring are as shown in Fig. 3.14

Name 9 8 4 . 5 6 7 . 1 3 2 1 0 Allele 1 0 1 . 0 0 1 . 1 1 0 0
Name 8 7 1 . 2 3 1 0 . 9 5 4 6 Allele 1 1 1 . 0 1 1 . 1 1 0 1

Name 9 8 4 . 2 3 1 0 . 1 6 5 7 **Allele** 1 0 1 . 0 1 0 . 1 0 0 1
Name 8 1 0 1 . 5 6 7 . 9 2 4 3 **Allele** 1 1 1 . 1 1 1 . 1 0 0 1

Fig.14 PMX *Crossover*

5.6.Mutation

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly disturbing genetic information. Mutation of a bit involves flipping a bit, changing 0 to 1 and vice-versa.

Flipping

Flipping of a bit involves changing 0 to 1 and 1 to 0 based on a mutation chromosome generated.

5.7 Replacement

Replacement is the last stage of any breeding cycle. Two parents are drawn from a fixed size population, they breed two children, but not all four can return to the population, so two must be replaced i.e., once off springs are produced, a method must determine which of the current members of the population, if any, should be replaced by the new solutions. The technique used to decide which individual stay in a population and which are replaced in on a par with the selection in influencing convergence . Basically, there are two kinds of methods for maintaining the population; generational updates and steady state updates.

In a steady state update, new individuals are inserted in the population as soon as they are created, as opposed to the generational update where an entire new generation is produced at each time step. The insertion of a new individual usually necessitates the replacement of another population member.

5.7.1 Random Replacement

The children replace two randomly chosen individuals in the population. The parents are also candidates for selection. This can be useful for continuing the search in small populations, since weak individuals can be introduced into the population.

5.7.2 Weak Parent Replacement

In weak parent replacement, a weaker parent is replaced by a strong child. With the four individuals only the fittest two, parent or child, return to population. This process improves the overall fitness of the population when paired with a selection technique
that selects both fit and weak parents for crossing, but if weak individuals and discriminated against in selection the opportunity will never raise to replace them.

5.7.3 Both Parents

Both parents replacement is simple. The child replaces the parent. In this case, each individual only gets to breed once. As a result, the population and genetic material moves around but leads to a problem when combined with a selection technique that strongly favors fit parents: the fit breed and then are disposed of.

5.8 Search Termination (Convergence Criteria)

In short, the various stopping condition are listed as follows:

• **Maximum generations–**The genetic algorithm stops when the specified number of generation's have evolved.

• **Elapsed time–**The genetic process will end when a specified time has elapsed.

Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.

• **No change in fitness–**The genetic process will end if there is no change to the population's best fitness for a specified number of generations.

Note: If the maximum number of generation has been reached before the specified number of generation with no changes has been reached, the process will end.

• **Stall generations**–The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length **Stall generations**.

• **Stall time limit**–The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to **Stall time limit**.

Example:Maximizing a Function

Consider the problem of maximizing the function, f*(x)* = x2
where x is permitted to vary between 0 to 31. The steps involved in solving this problem are as follows:

Step 1: For using genetic algorithms approach, one must first code the decision variable 'x' into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained. The objective function here is f*(x)* = x2 which is to be maximized. A single generation of a genetic algorithm is performed here with encoding, selection, crossover
and mutation. To start with, select initial population at random. Here initial population of size 4 is chosen, but any number of populations can be elected based on the requirement and application. Table 1. shows an initial population randomly selected.

String No.	Initial population (randomly selected)	x value	Fitness value $f(x) = x^2$	$Prob_i$	Percentage probability	Expected count	Actual count
1	01100	12	144	0.1247	12.47%	0.4987	1
2	11001	25	625	0.5411	54.11%	2.1645	2
3	00101	5	25	0.0216	2.16%	0.0866	0
4	10011	19	361	0.3126	31.26%	1.2502	1
Sum			1155	1.0000	100%	4.0000	4
average			288.75	0.2500	25%	1.0000	1
maximum			625	0.5411	54.11%	2.1645	2

*Table 1.**Selection***

Step 2: Obtain the decoded x values for the initial population generated. Consider string 1,Thus for all the four strings the decoded values are obtained.

Step 3: Calculate the fitness or objective function. This is obtained by simply squaring the 'x' value, since the given function is
f(x) = x2.
When, x = 12, the fitness value is,
f(x) = 144
for x = 25, f(x) = 625 and so on, until the entire population is computed

Step 4: Compute the probability of selection,

$$\text{Probi} = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i}$$

where
n = no of populations
f(x)=fitness value corresponding to a particular individual in the population
Σf(x)- Summation of all the fitness value of the entire population.
Considering string 1,
Fitness f(x) = 144
Σf(x) = 1155

The probability that string 1 occurs is given by,

P1 = 144/1155 = 0.1247
The percentage probability is obtained as,

0.1247∗100 = 12.47%
The same operation is done for all the strings. It should be noted that, summation of probability select is 1.
Step 5: The next step is to calculate the expected count, which is calculated as,

$$\text{Expected count} = \frac{f(x)_i}{(\text{Avg}f(x))_i}$$

$$\text{where } (\text{Avg } f(x))_i = \left[\frac{\sum_{i=1}^{n} f(x)_i}{n} \right]$$

For string 1,
Expected count = Fitness/Average = 144/288.75 = 0.4987
Computing the expected count for the entire population. The expected count gives an idea of which population can be selected for further processing in the mating pool.
Step 6: Now the actual count is to be obtained to select the individuals, which would participate in the crossover cycle using Roulette wheel selection. The Roulette wheel is formed as shown in Fig. 3.15. Roulette wheel is of 100% and the probability of selection as calculated in step4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the no of occurrences of population is noted to get actual
count. String 1 occupies 12.47%, so there is a chance for it to occur at least once. Hence its actual count may be 1. With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of
being selected twice. Thus its actual count can be considered as 2.
On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, it actual count is 0. String 4 with 31.26% has at least one chance for

occurring while Roulette wheel is spun, thus its actual count is 1. The above values of actual count are tabulated as shown is Table 1

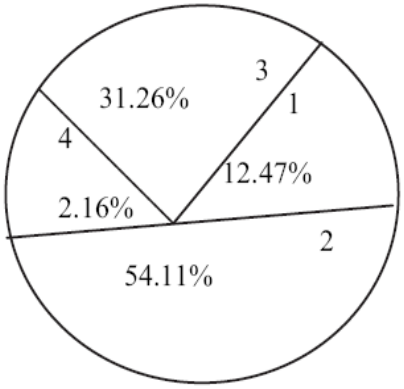

Fig.15 Roulette Wheel *Selection*

Step 7: Now, writing the mating pool based upon the actual count as shown in Table 2

The actual count of string no 1 is 1, hence it occurs once in the mating pool. The actual count of string no 2 is 2, hence it occurs twice in the mating pool. Since the actual count of string no 3 is 0, it does not occur in the mating pool. Similarly, the actual count of string no 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.

String No.	Mating pool	Crossover point	Offspring after crossover	x value	Fitness value $f(x) = x^2$
1	0 1 1 0 0	4	0 1 1 0 1	13	169
2	1 1 0 0 1	4	1 1 0 0 0	24	576
2	1 1 0 0 1	2	1 1 0 1 1	27	729
4	1 0 0 1 1	2	1 0 0 0 1	17	289
Sum					1763
average					440.75
maximum					729

*Table 2.**Crossover***

Step 8: Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single point crossover is performed and new offspring is produced. The parents are:

Parent 1 0 1 1 0 0
Parent 2 1 1 0 0 1

The offspring is produced as,
Offspring 1 0 1 1 0 1
Offspring 2 1 1 0 0 0

In a similar manner, crossover is performed for the next strings.
Step 9: After crossover operations, new off springs are produced and 'x' values are decodes and fitness is calculated.
Step 10: In this step, mutation operation is performed to produce new off springs after crossover operation. As discussed in mutation-flipping operation is performed and new off springs are produced.
Table 3. shows the new offspring after mutation. Once the off springs are obtained after mutation, they are decoded to x value and find fitness values are computed. This completes one generation. The mutation is performed on a bit-bit by basis .The crossover probability and mutation probability was assumed to 1.0 and 0.001 respectively. Once selection, crossover and mutation are performed, the new population is now ready to be tested.
This is performed by decoding the new strings created by the simple genetic algorithm after mutation and calculates the fitness function values from the x values thus decoded.

String No.	Offspring after crossover	Mutation chromosomes for flipping	Offspring after Mutation	X value	Fitness value $F(x) = x^2$
1	**0** 1 1 0 1	**1** 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
2	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 **0** 0 1	0 0 **1** 0 0	1 0 1 0 0	20	400
Sum					2546
average					636.5
maximum					841

*Table 3.**Mutation***

The results for successive cycles of simulation are shown in Tables 1–3.From the tables, it can be observed how genetic algorithms combine high performance notions to achieve better performance. In the tables, it can be noted how maximal and average performance has improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 841 during same period. Although random processes make this best solution, its improvement can also be seen successively.

The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance. When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 3.2), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed. Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice.
This example has shown one generation of a simple genetic algorithm.