UNIT 11 SIMPLE GENETIC ALGORITHM

Structure **Page No**

11.1 INTRODUCTION

Inspired by processes and models in nature in the last quarter of the 20th century there was growing interest towards evolving mathematical and computational models. In strict terms Genetic Algorithm (GA) is a mathematical search technique based on the principles of natural selection and natural genetics. It is also viewed as a numerical optimization technique capable of being applied to an extremely wide range of problems.

A possible solution to a problem is referred to as an individual. An individual is represented by a computational data structure called a chromosome, which is encoded using a fixed length bit string. Unlike some computational and mathematical approaches, the potential and promise has not been overemphasised in the literature. In fact GAs have been used for a very wide range of application domains to help solve practical problems on a daily basis. One of the basic reasons is that the approach is very simple and appealing. The algorithms are simple to understand and the required computer code is also easy to write in any procedural programming language. Although there is a growing number of disciples of GAs, the technique has never attracted as much attention as some other artificial intelligence techniques such as artificial neural networks have. More recently there has been an increasing attempt to look for models and approaches that are inspired by nature and natural systems. Therefore, the thought of extending the concept of natural selection and natural genetics to problem solving is such an obvious one that one might wonder why it was not tried earlier. In fact it was. From the very beginning, computer scientists have had visions of systems that mimicked one or more of the attributes of life. The idea of using a population of solutions to solve practical engineering optimisation problems was considered several times during the 1950s and 1960s. However, GAs was in essence invented by John Holland in the 1960s. Holland's double aim in his publication *Adaptation in Natural and Artificial Systems* was to have a better understanding of the natural adaptation processes, and to design artificial systems that had characteristic similar to the natural systems. The basic idea of GA is that a genetic pool of a given population potentially contains the solution, or at least a better solution, to a given adaptive problem. More recently others, for example De Jong, in a paper entitled *Genetic Algorithms are NOT Function Optimizers*, have been keen to remind us that GAs have far more potential than just being a robust method for estimating a series of unknown parameters within a model of a physical system.

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, shop job scheduling, games playing. We shall start the unit by basic concepts of simple genetic algorithm in Sec. 11.2. In Sec. 11.3, the three genetic operator reproduction, crossover and mutation are discussed in detail. In Sec. 11.4, traditional search methods are discussed. We shall discuss simple Genetic Algorithm in Sec. 11.5. We shall list the advantages and disadvantages in Sec 11.6.

Objectives

After studying this unit you will be able to:

- appreciate the genetic or evolutionary approaches as to why and how they have influenced the computing methods for problem solving;
- describe Simple Genetic Algorithm (SGA), a limited exposure of the genetic operators;
- define the genetic operators;
- apply reproduction, crossover and mutation operators;
- describe the traditional search methods;
- define a simple genetic algorithm;
- apply genetic algorithm to find better solution;
- list advantages and disadvantages.

11.2 BASIC CONCEPTS OF SIMPLE GENETIC ALGORITHM

In 1960, Rechenberg introduced the concept of computing. In computing, the non-traditional computerized search and the algorithm for optimization arc Genetic algorithm, which are based on the mechanics of natural selection and natural genetics. Genetic Algorithms (GA) was envisaged by Prof. Holland of University of Michigan. The GA are applied in structural engineering, biology, image processing and pattern recognition, computer science, neural networks, physical sciences and social sciences.

In GA, design space is converted into genetic space. For this the coding of variables is done, which discretizes the space. The function may be continuous. The traditional optimization techniques like linear programming, transportation, assignment, etc., use a single point approach, while GA uses a population of points at one time. GA uses randomized operators while traditional optimization techniques use the deterministic transition rules. GA can be applied to define the objective function, implement genetic representation and genetic operators.

Genetic algorithms use a vocabulary borrowed from natural genetics. In a population comprising of individuals (or genotypes, structures); these individuals are also known as *strings* or *chromosomes*. While in the other organisms each cell carries a certain number of chromosomes (man, for example, has 46 of them); however, in GA each individual is composed of one chromosome. Chromosomes are made of units better known as *genes* (also features, characters, or decoders) arranged in linear succession, array. Each gene is responsible for the inheritance of one or several characters. Genes of certain characters are located at certain positions of the chromosome, in other words a position may associate to a character of the organism. Such locations are called loci (string positions). Further, the value of the gene at a locus can be one from finite possibilities. Thus a gene is said to be in several states, called **alleles** (feature values). Each chromosome would represent a potential solution to a problem. The meaning of a particular chromosome is defined externally by the user. An evolution process runs on a population of chromosomes corresponding to a search through a space of potential solutions called a search space. Such a search requires balancing two

(apparently conflicting) objectives: exploiting the best solutions and exploring the search space. Hill climbing is an example of a strategy which exploits the best solution for possible improvement; on the other hand, it neglects exploration of the search space. Random search is a typical example of a strategy which explores the search space ignoring the exploitations of the promising regions of the space. Genetic algorithms pose more like a class of general purpose (domain independent) search methods which strike a remarkable balance between exploration and exploitation of the search space.

The mechanisms of a simple genetic algorithm are very simple. A genetic algorithm involves the simple tasks of coding a set of parameters, search from a population of points, use payoff information, i.e. objective function not auxiliary knowledge, and probabilistic transition rules and not deterministic rules. Thus effectively a simple genetic algorithm involves simple activities such as copying strings and swapping partial strings. Therefore, following are the essential elements of a typical genetic algorithm as presented by flow chart in the Fig. 1 and subsequently the stepwise procedure.

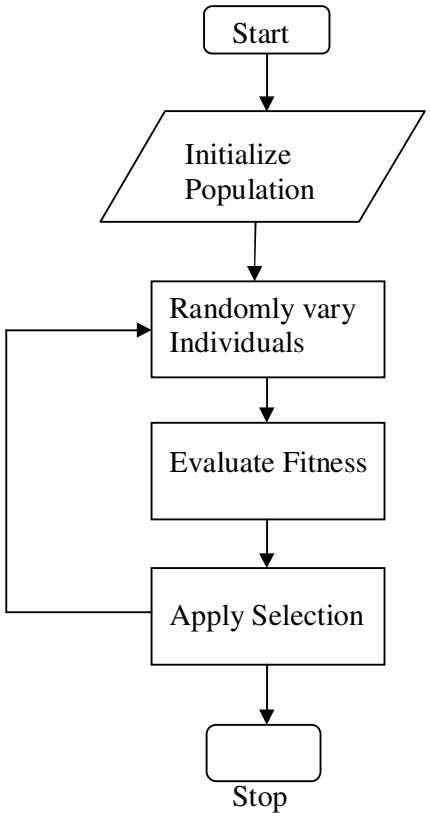

Fig. 1: Flow chart of genetic algorithm

A simple genetic algorithm procedure is given in the following steps.

1. Generation of a population of solution: In this, an **encoding** of solutions to the problem as a chromosome is done and a **number**, or **population**, of guesses of the solution to the problem are found.

2. Fitness evaluation: a way of **calculating how good or bad** (fitness) the individual solutions within the population are listed.

3. Reproduction: a method for **mixing fragments of the better solutions** (reproduction) to form new, on average even better solutions; and

4. A mutation operator to avoid permanent loss of diversity within the solutions.

Let us discuss these steps in details.

ENCODING

Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet.

For this, consider an optimization problem of maximizing the function $f(x) = x^2$ on the integer interval [0, 31].

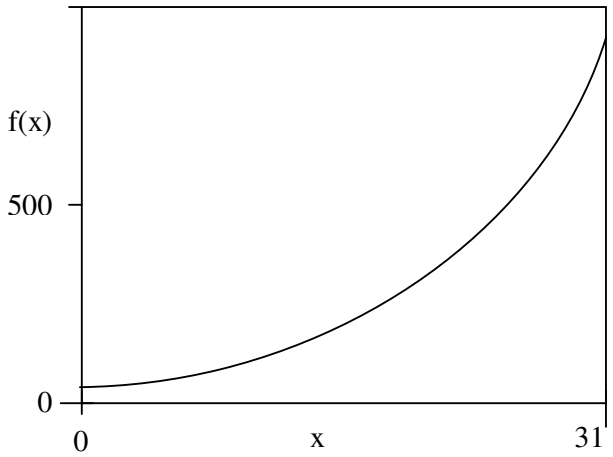

Fig. 2: Function $f(x) = x^2$ on the integer interval [0, 31]

The parameter in this problem is x. In order to maximize the value of a traditional method would determine the values of x such that the objective function value is highest. However, resorting to a GA the first requirement is to code the parameter x as a bit-string of finite length.

There are many ways to code the parameter x.

METHODS FOR CODING

i) Simple Binary Coding

The most common way of encoding is a binary string, which would be represented as in Table 1. 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 can represent a number. The way bit strings can code differs from problem to problem.

Table 1

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

Binary encoding gives many possible chromosomes with a smaller number of alleles. On the other hand this encoding is not natural for many problems and sometimes corrections must be made after genetic operation is completed. Binary coded strings with 1s and 0s are mostly used. The length of the string depends on the accuracy.

In this,

- Integers are represented exactly
- Finite number of real numbers can be represented
- Number of real numbers represented increases with string length

To understand the method of simple binary coding, let us consider the following illustration. Consider a black box device with five input switches. For every setting of switches an output signal f, $f = f(s)$, where s denotes a setting of the five switches. The objective of the problem is to find the setting of the five switches for which value of f is possibly maximum. The optimization methods, based on some deterministic transition rules work with the switch setting or the parameter set to find out the setting that would yield the maximum value of f.

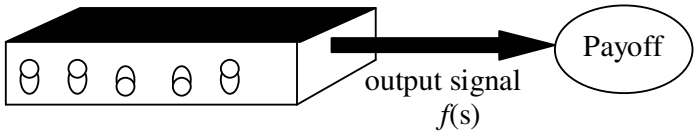

Fig. 3: Black box switching problem

In order to use genetic algorithms the switches need to be coded as string of finite length. A string of length five contains 1s to represent the switch being on and 0s to denote a switch which is off. The simple binary code 11001 denotes the first, second and fifth switches are on and the third and fourth switches are off, this is shown in Fig. 3. Thus with a random start for an initial population of size n = 3 following may be the bit-string of length five,

00101
10001
01100

Binary coding is the most used and popular coding method.

ii) Octal Coding

Octal codes are a string of finite length of digits 0 -7. Such as 1036 is a four-bit octal string. It is equivalent to the integer value 542. Therefore the octal representation of any integer value can be one possible octal code where integer value itself may indicate a parameter. A four bit octal string can represent integers from 0 to 4095. Hence (0000 0000) and (7777 7777) could represent points the points X_1, X_2 as (X_1^L, X_2^L); (X_1^U, X_2^U) where X_1^L and X_2^L are the lower limits of X_1 and X_2, and X_1^U and X_2^U are the upper limits of X_1 and X_2, respectively.

iii) Hexadecimal coding

Hexadecimal code may represent a chromosome by string of hexadecimal values. In this coding system 0-9, A-F is used.

iv) Permutation Coding

In Permutation Coding every chromosome is a string of numbers which represent the numbers in the parameter set. The example is given in Table 2.

v) Value Coding

Value Coding is very useful for some special problems as a chromosome is string of some values such as in Table 3.

vi) Tree Coding

Tree Coding is mainly useful for evolving program expressions for genetic programming. In a tree encoding, every chromosome is a tree of some objects such as functions and commands in a programming language. LISP is often used because

programs in it are represented in this form and can be easily be parsed as a tree so that functions and genetic operators can be applied easily. The same is shown in Fig. 4.

Table 2: Permutation Coding

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

Table 3: Value Coding

Chromosome -A	4.3251	7.345	4.1234	6.6543	8.4563
Chromosome -B	Red	Black	White	Yellow	Pink
Chromosome -A	Right	Right	Left	Forward	Backward

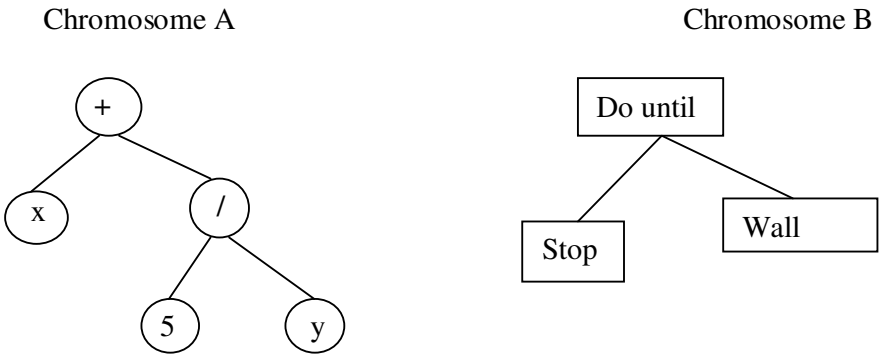

Fig. 4: Tree coding

Example 1: Compare the natural evolution and genetic algorithm in terms of components used in it.

Solution

Table 4: Comparison of natural evolution and genetic algorithm terminology

Natural evolution	Genetic algorithm
Chromosome	String
Gene	Feature or character
Allele	Feature value
Locus	String position
Genotype	Structure or coded string
Phenotype	Parameter set, a decoded structure

FITNESS FUNCTION

As pointed out that GAs mimic the Darwinian theory of survival of the fittest and the principle of nature to make a search process. Therefore, of the various types of optimization problems, GAs is usually suitable for solving maximizing problems. Minimization problems are usually transformed into maximization problems through a suitable transformation such as $\mathrm{Max}\left(\mathrm{F}(\mathrm{x})\right) = -\mathrm{Min}\left(-\mathrm{f}\left(\mathrm{x}\right)\right)$. In general, fitness function $\mathrm{F}(\mathrm{X})$ is first derived from the objective function and used in successive genetic operations. Some genetic operators require that the fitness function be non-negative, although certain operators do not have this requirement. The fitness functions are also known as the string's (chromosome's) fitness.

Most often one is in search of the best solution in a specific set of solutions. The set of solutions among which the desired solution resides also better known as the space of all feasible solutions is called search space (also state space). Each and every point in the search space represents one possible solution. Therefore each possible solution can be "marked" by its fitness value, depending on the problem definition. With Genetic Algorithm one looks for the best solution among a number of possible solutions represented by one point in the search space i.e.; GAs are used to search the search space for the best solution e.g., minimum. The difficulties in this case are the local minima and the starting point of the search.

SEARCH

Hill climbing methods use the iterative improvement technique. The technique is applied to a single point (a chromosome in case of GA) in the search space. During a single iteration, a new point is selected from the neighbourhood of the current point (this is why this technique is known also as neighbourhood search or local search). If the new point provides a better value of the objective function formulated as the fitness function, the new chromosome becomes the current chromosome. Otherwise, some other neighbour is selected and tested against the current chromosome. The method terminates if no further improvement (with respect to the fitness function) is possible. It is clear that the hill climbing methods provide local optimum values only and these values depend on the selection of the starting point. Moreover, there is no information available on the relative error (with respect to the global optimum) of the solution found. To increase the chances to succeed, hill climbing methods are usually executed for a (large) number of different starting points (the chromosomes in GA or the points need not be selected randomly). At times a selection of a starting point for a single execution may depend on the result of the previous runs.

A few important features of Genetic Algorithm are of significance. First it is a stochastic algorithm. Therefore, randomness has an essential role in genetic algorithms. Both selection and reproduction needs random procedures. A second very important point is that genetic algorithms always consider a population of solutions. Keeping in memory more than a single solution, at each iteration, offers a lot of advantages. The algorithm can recombine different solutions to get better ones and so, it can use the benefits of assortment. A population base algorithm is also very amenable for parallelization. The robustness of the algorithm should also be mentioned as something essential for the algorithm success. Robustness refers to the ability to perform consistently.

Now, in the following section, we shall discuss the genetic operators.

11.3 GENETIC OPERATORS

A simple GA is composed of three genetic operators:

i) Reproduction

ii) Crossover

iii) Mutation

Reproduction

Reproduction operation is also referred as selection operator. This operator is basically used to select the good ones from the population of strings based on their fitness information. Reproduction is an artificial version of natural selection, a Darwinian survival of fittest among string creatures competing with each other. It is a

process in which individual strings are copied according to their objective function values, f (fitness function). Intuitively a fitness function may be a measure of profit, utility or goodness that is desired to be maximized. Copying strings according to their fitness values means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation. Considering the principle of natural selection, the fitness is determined by the creature's ability to survive the predator, pestilence, and other obstacles at adulthood and subsequent reproduction. There are a number of reproduction operators in GA literature but the essential idea in all of them is that the above average chromosomes are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. The various schemes for selecting chromosomes for the mating pool for reproduction are:

- Roulette-wheel selection or Proportionate selection
- Rank selection
- Tournament selection
- Elitism

Besides the above Boltzmann selection and steady-state selection methods are also found in the literature but detailed description of all these methods is beyond the scope of this course.

i) Roulette-wheel Selection

The reproduction operator is implemented in the algorithmic form in a number of ways. The easiest method is the use of a biased roulette wheel. In the Roulette-wheel selection a chromosome is selected from the mating pool with a probability proportional to its fitness. Let us select i^{th} chromosome in the population with a probability proportional to F_i the fitness value of the string. In this a wheel called Roulette-wheel is considered and the top surface area of the wheel is divided into N parts (N is the population size). Each part is proportional to the functional value F_i. The wheel is rotated either clockwise or anti-clockwise. After it stops, a fixed pointer is used to indicate the winning area. Since the population size is usually fixed in a simple GA, the sum of the probabilities of each of the selected strings for the mating pool must be one. Therefore, the probability of the i^{th} selected string is,

$$p_i = \frac{F_i}{\sum_{i=1}^{N} F_i} \qquad (1)$$

In this way, the wheel is rotated N times and each time only one area is identified by the pointed as winner.

Example 2: Consider the bit-strings of length 5 like in case of the black box example. Suppose the sample population has values of the fitness function as presented in Table 4. Let the percentage of population total fitness be also as given in the same table. The corresponding weighted roulette wheel for the given sample is as in Fig. 5. To reproduce, the weighted Roulette-wheel defined in this manner is spun four times. Consider the string 1 with fitness value 169 representing 14.4% of the total fitness. As a result, string 1 is given 14.4% of the biased roulette wheel, and each spin turns up string 1 with probability of 0.144. Each time a new string is required the spinning of the weighted roulette wheel produces a candidate. Thus higher the fitness of the string higher is the number of offspring in the successive generation. The strings selected by reproduction are primarily the replica of the strings in the previous population. The

reproduced set of strings is put in the mating pool, which is a tentative new population, for it to be put through the next genetic operation.

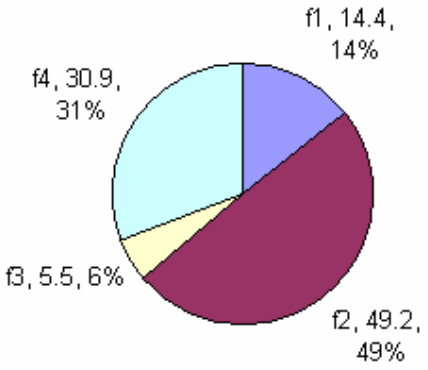

Fig. 5: A Roulette-wheel selection

Table 5: Sample problem strings and fitness values

No.	String	Fitness	% of Total	p_i	Expected count $= Np_i$	String no. in rotation	The count in the mating pool
1	01101	169	14.4	0.144	0.516	2	2
2	11000	576	49.2	0.492	1.968	1	2
3	01000	64	5.5	0.055	0.220	1	0
4	10011	361	30.9	0.309	1.236	2	0

ii) Rank Selection

Due to the chance of occurrence of the premature convergence in Roulette-wheel selection, ranking selection is applied. In this the strings are arranged in the ascending order of their fitness value and are ranked from 1 to N respectively for minimum fitness value to maximum fitness value.

Then a proportionate selection scheme based on the assigned rank is adopted.

Example 3: Consider the fitness values given in Example 2. The ascending order of these fitness value is F_3, F_1, F_4, F_2.

Table 6

String No.	%	Rank	Rank based proportional
1	14.4%	2	20%
2	49.2%	4	40%
3	5.5%	1	10%
4	30.9%	3	30%

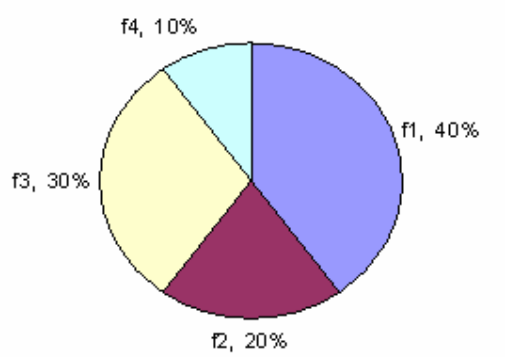

Fig. 6: According to rank selection

iii) Tournament Selection

This selection was studied first in Brindle's dissertation. In this scheme, a tournament size n is selected, which is smaller comparative to the population size N. Now n strings are picked from the population, and the best string in terms of fitness value is considered. Therefore, only one string is selected per tournament and N tournaments are considered to make the size of making pool equals to N.

iv) Elitism

In this scheme, an elite string again in terms of fitness value is selected first from a population of string, which is then copied into next generation to ensure its presence.

Example 4: Consider the population given in Example 2 and find the expected number of copies of the best string using
i) Tournament selection
ii) Elitism

Solution: i) Let us consider a tournament of 2 individuals.

Random selection of Tournament	Chosen string from the Tournament
2 and 4	2
2 and 3	2
1 and 4	4
3 and 4	4

The strings 2 and 4 are chosen twice and 1 and 3 are not chosen at all, therefore the new mating pool is 11000, 10011, 11000 and 10011.

ii) Using elitism, we get

String No.	p_i	p_i /average p_i	Count	Mating pool
1	0.144	0.576	1	01101
2	0.492	1.968	2	11000
3	0.055	0.220	0	11000
4	0.309	1.236	1	10011

Now try the following exercise.

E1) Consider the following population of binary strings for a maximization problem reproduce it using:

i) Tournament selection
ii) Roulette-wheel selection
iii) Rank selection
iv) Elitism

String	000101	010101	110000	101010	011011	100001
Fitness	2	5	1	10	8	20

Crossover

Crossover is the process of taking two solutions as parent and producing from the child(ren). 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. While two parents produce two offspring there is a chance that

- the chromosomes of the two parents are copied unmodified as offspring
- the chromosomes of the two parents are randomly recombined (crossover) to form offspring

Generally the chance of crossover is observed to be between 0.6 and 1.0

Thus 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 Crossover operator is popularly of the following three types:
- Single point crossover
- Two point crossover (Multipoint crossover)
- Uniform crossover

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.

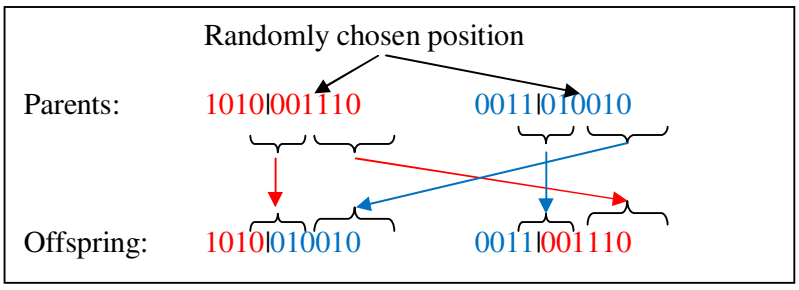

Fig. 7: Single point crossover

Fig. 7 illustrates the single point crossover and it can be observed that the bits next to the crossover point are swapped to produce the offspring while the crossover point is usually chosen randomly.

Two Point Crossover (Multi-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. Two point crossover is illustrated in Fig. 8.

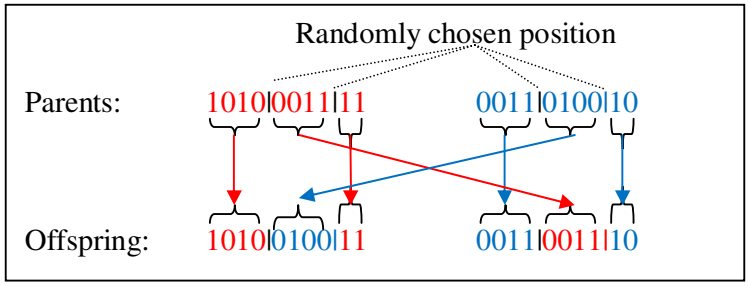

Fig. 8: Two point (multi-point) crossover

Two point crossover can easily be extended for more than two cut points, then it is multi-point crossover.

Uniform Crossover

Uniform Crossover is an extreme of the multi-point crossover. In a uniform crossover each bit from either parent is selected with a probability of 0.5 and then interchanged with the bit of the other parent as illustrated in Fig. 9. Uniform crossover is radically different from one-point crossover. Sometimes a gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to the randomly generated crossover mask.

A mask is a bit strings of length same as the length of the chromosomes. When there is 1 in the mask, the gene is copied from the first parent and when there is 0, then the gene is copied from the second parent. The process is repeated with the parents exchanging their status to produce the second child. A new crossover mask is randomly generated for each pair of parents. Offspring therefore contain a mixture of genes from each parent. The number of effective crossing points is not fixed but an average of L/2 is observed for L = length of the chromosomes.

Parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Crossover mask	1	0	1	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0

Children	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0

Fig. 9: Uniform crossover

Example 5: Consider two solution chromosomes of 11 bits randomly selected from a mating pool (population of solutions) on which uniform crossover is to be applied.

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

Crossover mask	1 0 0 1 0 1 1 1 0 0 1
Parent-1	1 0 1 0 0 0 1 1 1 0 1
Parent-2	0 1 0 1 0 1 0 0 1 1 0
Offspring-1	1 1 0 0 0 0 1 1 1 1 1

Crossover mask	1 0 0 1 0 1 1 1 0 0 1
Parent-2	0 1 0 1 0 1 0 0 1 1 0
Parent-1	1 0 1 0 0 0 1 1 1 0 1
Offspring-2	0 0 1 1 0 1 0 0 1 0 0

Now, try the following exercise.

E2) Applying single point at 9th cross site, multi-point at 5th and 12th cross site and uniform crossover with
Cross mask: 110011001100110011

Parent 1: 110001010101110000
Parent 2: 010110101010001111
Find the solution chromosomes of 18 bit from the given parents.

So far, we discussed reproduction and crossover. In the following section, we shall discuss mutation operation.

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 for improved results. Mutation has been traditionally considered as a simple search operator. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploration of the whole search space. Mutation is viewed as a background operator to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. Mutation helps escape from the trap of local minima and maintains diversity in the population. It also keeps the gene pool well stocked, and thus ensuring ergodicity.

A search space is said to be ergodic if there is a non-zero probability of generating any solution from any population state. Typically a population of genes and specifically a mating pool used by a GA is ergodic in nature as each chromosome is already a potential solution with some fitness.

There are many different forms of mutation for the different kinds of representation. For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability. Such an operator can accelerate the search. But care should be taken, because it might also reduce the diversity in the population and makes the algorithm converge toward some local optima. Mutation involves flipping each gene independently, changing 0 to 1 and vice-versa with a probability p_m . p_m is known as the mutation rate typically within the [1/population_size, 1/ chromosome_length]. The bit-wise mutation is performed bit-by-bit by flipping a coin with a probability of p_m .

Flipping a coin with a probability p_m is simulated by randomly choosing a number between 0 and 1 and compares it with p_m . If random number is less than p_m then the outcome of the coin flipping is true, otherwise outcome is false. If at any bit the outcome is true then the bit is altered independently otherwise the bit remains unchanged i.e. the mutation, of a gene does not affect the probability of mutation of another gene.

Example 6: Consider the following population of eight-bit strings.

Population:

0110 1011
0011 1101
0001 0110
0111 1100

Observe that all the four chromosomes have a zero in the leftmost position but if the optimal solution requires a one in the leftmost position then neither reproduction nor crossover operators would succeed in obtaining one in that position. Let mutation with a certain mutation probability p_{mi} produces the following:

Population:

0110 1011
0011 1101
0001 0110
1111 1100

Now, try the following exercise.

E3) Consider the following population having six, 8-bit strings
00100110
00101001
10010110
10011001
Apply mutation operators on this set of population.

Now, let us discuss the traditional methods of search in the following section.

11.4 TRADITIONAL SEARCH METHODS

The basic principle of optimization is the efficient allocation of the limited resources. Optimization can be applied to any scientific or engineering discipline. The aim of optimization is to find an algorithm, which solves a given problems. There exists no specific method, which solves all optimization problems. Consider a function,

$$f(x) : [x^1,\ x^u] \rightarrow [0,\ 1]$$

where,

$$f(X) = \begin{cases} 1, \text{ if } |x - a| < \in, \in > 0 \\ -1, \text{ otherwise} \end{cases}$$

For the above function, f can be maintained by decreasing $\in$ or by making the interval of $[x^1,\ x^u]$ large. Thus a difficult task can be made easier. Therefore, one can solve optimization problems by combining human creativity and the raw processing power of the computers.

The various conventional optimization and search techniques available are discussed as follows:

i) Gradient-Based Local Optimization Method

When the objective function is smooth and one need efficient local optimization, it is better to use gradient based or Hessian based optimization methods. The performance and reliability of the different gradient methods varies considerably. To discuss gradient-based local optimization, a smooth objective function (i.e., continuous first and second derivatives) is considered below.

The objective function is denoted by,

$$f(x) : R^n \rightarrow R$$

The first derivatives are contained in the gradient vector $\nabla f(x)$.

$$\nabla f(X) = \begin{bmatrix} \partial f(x)/\partial x_1 \\ \vdots \\ \partial f(x)/\partial x_n \end{bmatrix}$$

Similarly second derivative is obtained and search direction as well as the next iterations point is determined in each of the iterations. There are a number of methods such as Newton's method, conjugate gradient method, secant method, etc. to search the direction. However all the gradient methods search for the minimum and not the maximum.

ii) Random Search

Random search is an extremely basic method. It only explores the search space by randomly selecting solutions and evaluates their fitness. This is quite an unintelligent strategy, and is rarely used by itself. Nevertheless, this method is sometimes worth testing. It is very simple to implement and an important number of evaluations can be done fairly quickly. For new unresolved problems, it can be useful to compare the results of a more advanced algorithm to those obtained just with a random search for the same number of evaluations. The efficiency of GA is extremely dependant on consistent coding and relevant reproduction operators. Building a genetic algorithm,

which performs no more than a random search, happens more often than we can expect. If the reproduction operators are just producing new random solutions without any concrete links to the ones selected from the last generation, the genetic algorithm is just doing nothing else that a random search. However good the obtained solution may be, if it's not optimal one, it can be always improved by continuing the run of the random search algorithm for long enough. A random search never gets stuck in any point such as a local optimum. Furthermore, theoretically, if the search space is finite, random search is guaranteed to reach the optimal solution.

iii) Stochastic Hill Climbing

Efficient methods exist for problems with well-behaved continuous fitness functions. These methods use a kind of gradient to guide the direction of search. Stochastic Hill Climbing is the simplest method of these kinds. In each of the iterations a solution in the neighbourhood of the current solution is randomly selected, and this new solution is retained only if has improved fitness function. Stochastic Hill Climbing converges towards the optimal solution if the fitness function of the problem is continuous and has only one peak (unimodal function). On functions with many peaks (multimodal functions), the algorithm is likely to stop on the first peak it finds even if it is not the highest one. Once a peak is reached, hill climbing cannot progress anymore, and that is problematic when this point is a local optimum. Stochastic hill climbing usually starts from a random select point.

A simple idea to avoid getting stuck on the first local optimal consists in repeating several hill climbs each time starting from different randomly chosen points. This method is sometimes known as iterated hill climbing. By discovering different local optimal points, it gives more chance to reach the global optimum. It works well if there are not too many local optima in the search space. But if the fitness function is very "noisy" with many small peaks, stochastic hill climbing is definitely not a good method to use. Nevertheless such methods have the great advantage to be really easy to implement and to give fairly good solutions very quickly.

iv) Simulated Annealing

Simulated Annealing was originally inspired by formation of crystal in solids during cooling i.e., the physical cooling phenomenon. As discovered a long time ago by iron age blacksmiths, the slower the cooling, the more perfect is the crystal formed. By cooling, complex physical systems naturally converge towards a state of minimal energy. The system moves randomly, but the probability to stay in a particular configuration depends directly on the energy of the system and on its temperature. Gibbs law gives this probability formally:

$$p = e^{\frac{E}{k^1}}$$

where E stands for the energy, k is the Boltzmann constant and T is the temperature. In the mid 1970s, Kirlpatrick by analogy of these physical phenomena laid out the first description of simulated annealing. As in the stochastic hill climbing, the iteration of the simulated annealing consists of randomly choosing a new solution in the neighbourhood of the actual solution. If the fitness function of the new solution is better than the fitness function of the current one, the new solution is accepted as the new current solution.

The simulated annealing behaves like a hill climbing method but with the possibility of going downhill to avoid being trapped at local optima. Simulated Annealing by mixing exploration features such as the random search and exploitation features like hill climbing usually gives quite good results. Simulated Annealing is a serious competitor to Genetic Algorithms.

Now, we shall discuss the model of a simple Genetic Algorithm.

11.5 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 is a search technique to find approximate solutions to optimization and search problems. Basically, an optimization problem looks really simple. The set of all the solutions that meet this form constitute the search space. The problem consists in finding out the solution that fits the best, i.e. the one with the most payoffs, from all the possible solutions. In this it is needed to use a specific technique to find the optimal solution.

Genetic Algorithms provides one of these methods. GA handles a population of possible abstract representation of the solutions, namely chromosome. Coding all the possible solutions into a chromosome is the first part. Subsequently a set of reproduction operators to be applied to the chromosomes needs to be determined. Reproduction operators are used to perform mutations and recombination over solutions of the problem. The behaviour of GA is dependant on representation and reproduction operators. Selection using a fitness function is enabling to compare each individual in the population. 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 Algorithm starts by randomly 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. 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 terminates when the population converges toward the optimal solution.

The basic genetic algorithm is as follows:

1. [start] Genetic random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [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. If no crossover was performed, offspring is the exact copy of parents.
- [Mutation] With a mutation probability, mutate new offspring at each locus.
- [Accepting] Place new offspring in the new population.

4. [Replace] Use new generated population for a further sum of the algorithm.
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population.
6. [Loop] Go to step2 for fitness evaluation.

Comparison of Genetic Algorithms 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.

These are the major differences that exist between Genetic Algorithm and conventional optimization techniques.

Example 7: Let us try to examine whether the GA can improve the solution from one generation to the next generation for the following.

Maximize $f(x) = \sqrt{x}$

Subject to

$$1 \le x \le 16$$

Solution: Assuming that the length of the string is 6. Now, let us first find the fitness values and corresponding counts, which are given in Table 6.

Table 7

String No.	Initial population	Decoded value (D)	x value $= x^L + \frac{x^U - x^L}{2^\ell} D$
1	100101	$1\times2^5 + 1\times2^2 + 1 = 37$	9.81
2	011010	$1\times2^4 + 1\times2^3 + 1\times2 = 26$	7.19
3	010110	$1\times2^4 + 1\times2^2 + 1\times2 = 22$	6.24
4	111010		14.81

5 6	101100 001101	$1\times2^5+1\times2^4+1\times2^3+1\times2=58$ $1\times2^5+1\times2^3+1\times2^2=44$ $1\times2^3+1\times2^2+1=13$	11.48 4.09
		200	

Where X^2 the lower value of x is, X^u is the upper value of x, l is the length of the string.

Table 7 (continued)

String No.	**f(X) = $\sqrt{x}$**	**p selection $\frac{f_i}{\sum f_i}$**	**Expected count $\frac{f_i}{\bar{f}}$**
1	3.13	0.18	1.07
2	2.38	0.15	0.91
3	2.50	0.14	0.85
4	3.85	0.22	1.31
5	3.39	0.19	1.16
6	2.02	0.12	0.69
	Sum $\sum f_i = 17.57$ Average $\bar{f} = 2.93$ Maximum $f = 3.85$		

Table 8

Actual count Roulette wheel	**Mating pool**	**Mating pair**	**Parents**	**Crossover site**	**Children strings**	**Mutation**
1	100101	3	100101	10|0101	101010	101010
1	011010	6	111010	11|1010	110101	110101
0	111010	1	011010	011|010	011100	011100
2	111010	5	101100	101|100	101010	111010
2	101100	4	111010	11|1010	111100	111100
0	101100	2	101100	10|1100	101010	101010

Table 8 (continued)

Decoded value	**x value**	**f(x) = $\sqrt{x}$**
42	11.00	3.32
53	13.62	3.69
28	7.67	2.77
58	14.81	3.85
60	15.28	3.91
42	11.00	3.32
		Sum $\sum f = 20.86$ Average $f = 3.48$ Maximum $f = 3.91$

The initial population of binary-string (of size 6) are created at random, which are modified using the proportionate selection (Roulette-Wheel selection), a single-point crossover $(p_c = 1.0)$ and a bit-wise mutation $(p_m = 0.03)$. The decoded values of the binary strings are determined and depending on the range of the variable, its real values are calculated corresponding to different binary-strings. Knowing the values of

the variables, the function values have been determined. As it is a maximization problem, the fitness values of different strings, their probabilities of being selected in the mating pool are calculated. The mating pool has been generated, using the principle of proportionate selection scheme. The mating pairs are identified and all the pairs participate in the single-point crossover, as the probability of crossover p_c has been assumed to be equal to 1.0. The children solutions created due to this crossover are shown in Table 8. As there are $6 \times 6 = 36$ bits in the population and $p_m = 0.03$, there is a chance that only one bit will be mutated. Let us suppose that the second bit (from left) of fourth string of the population created due to crossover, has been mutated (that is, 0 is changed into 1). Tables 7 and 8 shows that the population average has increased from 2.93 to 3.48 and the maximum fitness value has improved from 3.85 to 3.91, in one generation of the GA-run. Thus, it indicates that the GA can improve the solutions.

Now, try the following exercises.

E4) Improve the solution of the following problem.

$$f(x) = \sqrt{x}$$

subject to $1 \le x \le 25$, by considering the length of the string 5. Show only one iteration by a hand calculation.

E5) Use a binary-coded GA to minimize the function $f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1 x_2$, in the range of $0.0 \le x_1, x_2 \le 5.0$. Use a random population of size N = 6, a single point crossover with probability $p_c = 1.0$ and neglect mutation. Assume 3 bits for each variable and thus, the GA-string will be 6-bits long. Show only one iteration by a hand calculation.

Now let us discuss the advantages and disadvantages of Genetic Algorithm.

11.6 ADVANTAGES AND DISADVANTAGES OF GENETIC ALGORITHM

Genetic Algorithm has a number of advantages over the traditional optimization tools but it has some disadvantages too, which are discussed below:

Advantages of GA

1. Genetic algorithm can handle the integer programming or mixed-integer programming problems effectively.
2. Genetic algorithm can optimize discontinuous objective functions also without the requirement of the gradient information of objective function.
3. It is suitable for parallel implementations.
4. A genetic algorithm starts with a population of initial solutions created at random. Out of all these initial solutions, if at least one solution falls in the global basin, there is a good chance for the GA to reach the global optimal solution. It is important to mention that initial population may not contain any solution lying in the global basin. In that case, if the GA-operators (namely crossover, mutation) can push at least one solution into the global basin, there is a chance that the GA will find the global optimal solution. Thus, the chance of its solutions for being trapped into the local minima is less.

5. The same GA with a little bit of modification in the string can solve a variety of problems. Thus, it is a versatile optimization tool.

Disadvantages of GA

Although the GA has a number of advantages, it has the following disadvantages:

1. Genetic algorithm computationally expensive and consequently, has a slow convergence rate.
2. Genetic algorithm works like a black-box optimization tool.
3. There is no mathematical convergence proof of the GA, till today.
4. An user must have a proper knowledge of how to select an appropriate set of GA-parameters.

Now, let us summarize the unit.

11.7 SUMMARY

This unit has been summarized as follows:

1. The working cycle of a GA has been explained. In a GA, the solutions are modified using different operators, such as reproduction, crossover, mutation and others. Reproduction selects good strings from a population and copies them in the mating pool. In crossover, there is an exchange of properties between the parents and as a result of which, the children solutions are created. Mutation brings a local change in the current solution and thus, helps the solution to come out of the local minimum, if any. Thus, the chance of its solutions for getting stuck at the local minima is less.
2. To ensure an effective search of the GA, there must be a proper balance between its population diversity (exploration) and selection pressure (exploitation). An appropriate set of the parameters, such as crossover probability, mutation probability, population size, maximum number of generations, is to be determined through a careful study for the above purpose.
3. A binary-coded GA cannot obtain any arbitrary precision required. It is so, because a large number of bits are to be assigned to represent a variable for obtaining the better precision. As the number of bits in the GA-string increases, its computational complexity will be more and as a result of which, the GA will become slower.
4. A GA can overcome almost all drawbacks of the traditional methods of optimization but it is found to be computationally expensive.

11.8 SOLUTIONS/ANSWERS

E1) i) Tournament selection

Random selection of tournament	**Chosen string from the tournament**
3 and 6	6
3 and 5	5
2 and 5	5
2 and 4	4
1 and 2	2
6 and 2	6

The string 2 is selected once, 3 is once, 4 is once, 5 is twice and 6 is once. Therefore the mating pool is 010101, 110000, 101010, 011011, 100001, 011011.

ii)

No.	String	Fitness	% fitness	P_i	NP_i	String no. rotation	The cost in the mating pool
1.	000101	2	4.2	0.042	0.252	4	0
2.	010101	5	11.0	0.11	0.66	6	0
3.	110000	1	2.2	0.022	0.132	4	0
4.	101010	10	21	0.21	1.26	6	2
5.	011011	8	17.6	0.176	1.056	5	1
6.	100001	20	44	0.44	2.84	6	3

The mating pool is 101010, 101010, 011011, 100001, 100001, 100001.

iii) The rank of the string according to their fitness value is given as

String no.	%	Rank	Rank based proportions
1	4.27%	2	9.5%
2	11.0%	3	14.3%
3	2.2%	1	4.8%
4	21%	5	23.8%
5	17.6%	4	19.0%
6	44%	6	28.6%

iv) Using elitism, we get

String No.	P_i	P_i/average P_i	Count	Mating pool
1	0.042	0.252	0	010101
2	0.11	0.66	1	101010
3	0.022	0.132	0	011011
4	0.21	1.26	1	100001
5	0.176	0.956	1	100001
6	0.44	2.64	3	100001

E2) i) Using single point crossover
Suppose the cross site is selected randomly as 9th, then we get

										/									
Parent 1:	1	1	0	0	0	1	0	1	0	/	1	0	1	1	1	0	0	0	0
Parent 2:	0	1	0	1	1	0	1	0	1	/	0	1	0	0	0	1	1	1	1

Offspring 1:	1	1	0	0	0	1	0	1	0	0	1	0	0	0	1	1	1	1	
Offspring 2:	0	1	0	1	1	0	1	0	1	1	0	1	1	1	0	0	0	0	

ii) Using multipoint crossover selected at 5th and 12th positions, we get

						/							/							
Parent 1:	1	1	0	0	0	/	1	0	1	0	1	0	/	1	1	1	0	0	0	0
Parent 2:	0	1	0	1	1	/	0	1	0	1	0	1	/	0	0	0	1	1	1	1

Offspring 1:	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	0	
Offspring 2:	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	

iii) Using uniform crossover,

Crossover mask:	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Parent 1:	1	1	0	0	0	1	0	1	0	1	0	1	1	1	0	0	0	0
Parent 2:	0	1	0	1	1	0	1	0	1	0	1	0	0	0	1	1	1	1

Offspring 1:	1	1	0	1	0	1	1	0	0	1	1	0	1	1	1	1	1	1
Offspring 2:	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0

E3) At second position of every string, there is a zero, therefore mutation produces the following

0	0	1	0	0	1	1	0
0	0	1	0	1	0	0	1
1	0	0	1	0	1	1	0
1	1	0	1	1	0	0	1

E4)

String No.	**Initial population**	**Decoded value**	**X-value** $= 1 + \frac{24}{32}$ **D.**	$\mathbf{f(x) = \sqrt{x}}$	$\mathbf{P = \frac{f_i}{\sum f_i}}$	$\mathbf{\frac{f_i}{\bar{f}}}$
1	10010	18	14.5	3.81	0.22	1.10
2	01101	13	10.75	3.29	0.19	0.95
3	01011	11	9.25	3.04	0.18	0.90
4	11101	29	22.75	4.77	0.28	1.40
5	00110	6	5.5	2.35	0.13	0.65

Now, the mating pool, mating pair can be written. Then, crossover and mutation is applied to get the modified solution using GA technique.

11.9 PRACTICAL ASSIGNMENT

Session 10

Write a program in "C" language to max $f(x) = \sqrt{x}$, subject to $1 \le x \le 100$ by considering the string length 8, using GA.

11.10 REFERENCES

1. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Netwoks: Fuzzy Logic, and Genetic Algorithms*.
2. D.K. Pratihar, (2008), *Soft Computing*.
3. J. –S. R. Jang, C. –T. Sun and E. Mizutani, (2004), *Neuro-Fuzzy and Soft Computing.*