
UNIT 11 SIMPLE GENETIC ALGORITHM

Structure	Page No
11.1 Introduction	5
Objectives	
11.2 Basic Concepts of Simple Genetic Algorithm	6
11.3 Genetic Operators	11
11.4 Traditional Search Methods	19
11.5 Simple Genetic Algorithm	21
11.6 Advantages and Disadvantages of GA	24
11.7 Summary	25
11.8 Solutions/Answers	25
11.9 Practical Assignment	27
11.10 References	27

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 are 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.

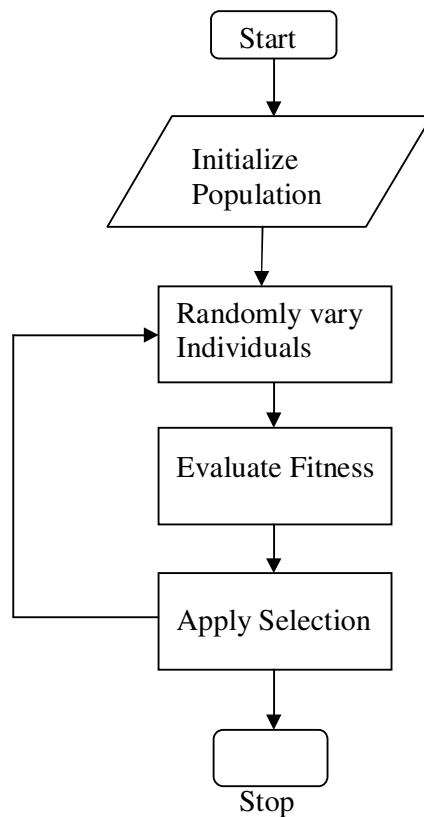


Fig. 1: Flow chart of genetic algorithm

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 $[0, 31]$.

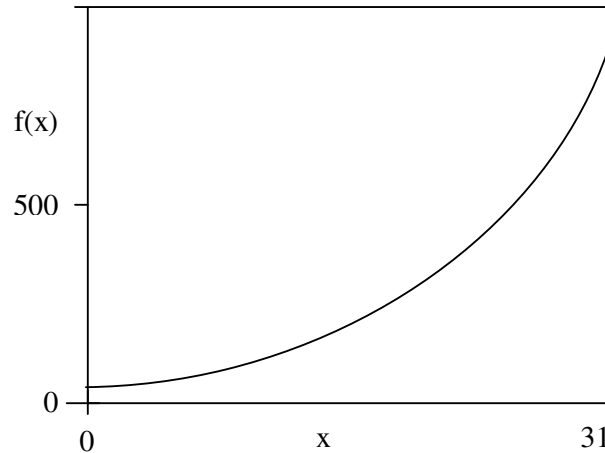


Fig. 2: Function $f(x) = x^2$ on the integer interval $[0, 31]$

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	0	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 .

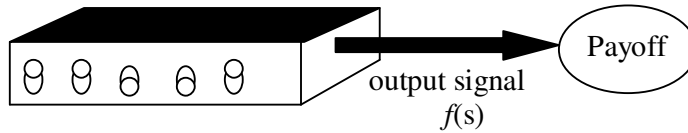


Fig. 3: Black box switching problem

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

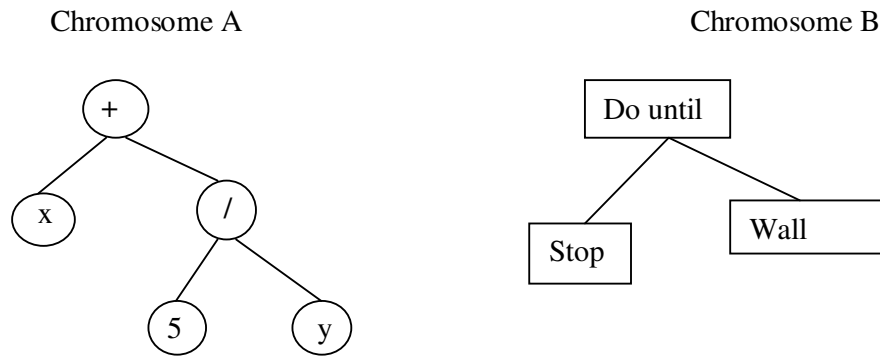


Fig. 4: Tree coding

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 $\text{Max}(F(x)) = -\text{Min}(-f(x))$. In general, fitness function $F(X)$ 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,

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

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

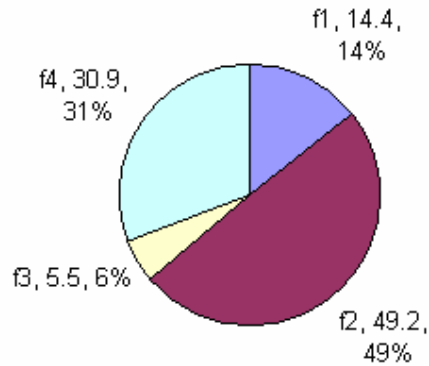


Fig. 5: A Roulette-wheel selection

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%

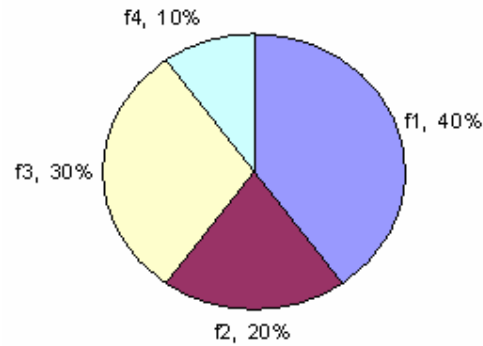


Fig. 6: According to rank selection

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

- Tournament selection
- 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.

- Using elitism, we get

String No.	p_i	$p_i / \text{average } p_i$	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.

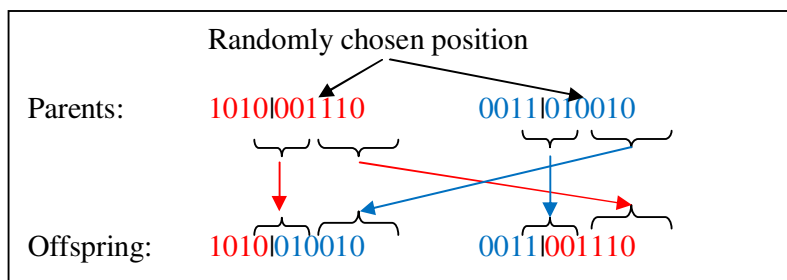


Fig. 7: Single point crossover

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.

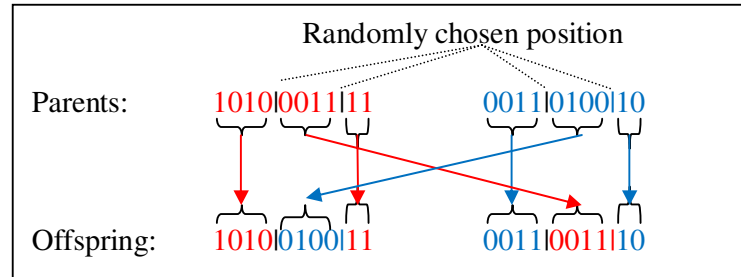


Fig. 8: Two point (multi-point) crossover

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 $[1/\text{population_size}, 1/\text{chromosome_length}]$. 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,

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

where,

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

For the above function, f can be maintained by decreasing ϵ or by making the interval of $[x^l, x^u]$ 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,

$$f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$$

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 than 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 it 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:

$$p = e^{\frac{-E}{kT}}$$

where E stands for the energy, k is the Boltzmann constant and T is the temperature. In the mid 1970s, Kirkpatrick 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 for 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 operators while conventional methods for continuous optimization apply deterministic transition operators i.e., GAs do 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

$$1 \leq x \leq 16$$

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 $= x^L + \frac{x^U - x^L}{2^L} D$
1	100101	$1 \times 2^5 + 1 \times 2^2 + 1 = 37$	9.81
2	011010	$1 \times 2^4 + 1 \times 2^3 + 1 \times 2 = 26$	7.19
3	010110	$1 \times 2^4 + 1 \times 2^2 + 1 \times 2 = 22$	6.24
4	111010	$1 \times 2^4 + 1 \times 2^2 + 1 \times 2 = 22$	14.81

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

Where X^l 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 $\frac{f_i}{f}$
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	01 1010	011100	011100
2	111010	5	101100	10 1100	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 $1 \leq x \leq 25$, 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

$$f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1x_2, \text{ in the range of } 0.0 \leq x_1, x_2 \leq 5.0.$$

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	$P_i/\text{average } P_i$	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	1
Offspring 2:	0	1	0	1	1	0	1	0	1	1	0	1	1	1	0	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	0	0
Offspring 2:	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0

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 $= 1 + \frac{24}{32} D.$	$f(x) = \sqrt{x}$	$P = \frac{f_i}{\sum f_i}$	$\frac{f_i}{f}$
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 $1 \leq x \leq 100$ by considering the string length 8, using GA.

11.10 REFERENCES

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

UNIT 12 APPLICATIONS OF GENETIC ALGORITHM

Structure	Page No
12.1 Introduction	29
Objectives	
12.2 Revisiting Simple Genetic Algorithm	29
12.3 Travelling Salesman Problem (TSP)	31
12.4 Summary	40
12.5 Solutions/Answers	41
12.6 Practical Assignment	43
12.7 References	44

12.1 INTRODUCTION

In this Unit we present an example of genetic algorithms tailored to a benchmark problem for a graph based application for optimization in Sec. 12.2. The travelling salesman problem (TSP) is a special problem which we shall discuss in Sec. 12.3. There are many reasons why it is considered as the mother of all problems. First of all, the TSP is conceptually very simple: the travelling salesman must visit every city in his territory exactly once and then return to the starting point. The path the salesman takes is called a tour. Given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour? The search space for the TSP is a set of permutations of given n cities. Any single permutation of n cities is a potential solution (which is a complete tour of n cities). The optimal solution is a permutation which yields the minimum cost of the tour. The size of the search space is $n!$. The TSP is a relatively old problem: it was documented as early as 1759 by Euler (though not by that name), whose interest was in solving the knights' tour problem. A correct solution would have a knight visit each of the 64 squares of a chessboard exactly once in its tour.

Objectives

After studying this unit you will be able to:

- recall the simple genetic algorithm;
- apply genetic algorithm for travelling salesman problem;
- apply crossover for TSP to find the solution.

Let us begin this section by revisiting the simple genetic algorithm.

12.2 REVISITING SIMPLE GENETIC ALGORITHM (SGA)

ALGORITHM

BEGIN

```
Generate initial population;  
Compute fitness of each individual;  
REPEAT /* New generation */  
  FOR population_size / 2 DO  
    Select two parents from old generation;  
    /* biased to the fitter ones */
```

```

    Recombine parents for two offspring;
    Compute fitness of offspring;
    Insert offspring in new generation
  END FOR
UNTIL population has converged
END

```

GA COMPONENTS

- **Generate initial population;**
- **Fitness Function;**
- **Production (Recombine) operators**
 - **Selection**
 - **Crossover**
 - **Mutation**
 - **Inversion**
- **Insert offspring in new generation**

Let us first discuss the following example.

Example 1: Maximize $f(x) = x^2$ subject to $0 \leq x \leq 31$

Solution: GA APPROACH

- Representation: binary code, e.g. 01101 = 13
- Population size: 4
- Random initialisation
- 1-point crossover, bitwise mutation
- Roulette wheel selection

PARAMETER SETTING

- Initialization:
 - Population size
 - Random
 - Dedicated greedy algorithm
- Reproduction:
 - Generational: as described before (insects)
 - Generational with elitism: fixed number of most fit individuals are copied unmodified into new generation
 - Steady state: two parents are selected to reproduce and two parents are selected to die; two offspring are immediately inserted in the pool (mammals)
- Stop criterion:
 - Number of new chromosomes
 - Number of new and unique chromosomes
 - Number of generations
- Measure:
 - Best of population
 - Average of population
- Duplicates
 - Accept all duplicates
 - Avoid too many duplicates, because that degenerates the population (intelt)
 - No duplicates at all

We show one generational cycle done by hand

Table 1

String no.	Initial population	x Value	Fitness $f(x) = x^2$	Prob _i	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	058	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Now try the following exercises

E1 Maximize $f(x, y) = 4x + 3y$

Subject to

$$2x + 3y \leq 6$$

$$-3x + 2y \leq 3$$

$$2x + y \leq 4$$

$$0 \leq x \leq 2$$

using genetic algorithm.

E2) Minimize $f(x) = x^2$

Subject to $1 \leq x \leq 51$

So far we discussed the simple genetic algorithm with its application. In the following section we shall discuss another important application of genetic algorithm, known as traveling salesman problem (TSP). TSP is a special type of scheduling problem.

12.3 TRAVELLING SALESMAN PROBLEM

The term 'travelling salesman' was first used in a 1932 German book, *The travelling salesman, how and what he should do to get commissions and be successful in his business*, written by a veteran travelling salesman. The TSP was introduced by the RAND Corporation in 1948. The Corporation's reputation helped to make the TSP a well known and popular problem. The TSP also became popular at that time due to the new subject of linear programming and attempts to solve combinatorial problems. The Travelling Salesman Problem was proved to be NP-hard. It arises in numerous real life applications as found in the literature where the number of nodes (cities) to be connected are very large such as, Circuit board drilling applications with up to 17,000 cities, X-ray crystallography instances with up to 14,000 cities, VLSI fabrication have been reported with as many as 1.2 million cities. Therefore, hours on a multi-million dollar super computer (due to the large number of cities) for the best solution may not be cost-effective if it be possible to obtain a good (near-optimal) solution within a few seconds on a PC. In the TSP, the goal is to find the shortest distance between n different cities. Testing every possibility for an n city tour would be $n!$ math additions. A 30 city tour would have to measure the total distance of be 2.65×10^{32} different tours. Assuming a comforting speed of trillion additions per second, this would take 252,333,390,232,297 years. Adding one more city would cause the time to increase by a factor of 31. Obviously, this is an impossible solution. This indicates the need for a heuristic approach/method to yield the acceptable solution in good time. A genetic algorithm can be used to find a solution in much less time. Although it might not find the best solution, it can find a near perfect solution for a 100 city tour in less than a minute.

During the last decades, several algorithms emerged to approximate the optimal solution: nearest neighbour, greedy algorithm, nearest insertion, farthest insertion, double minimum spanning tree, strip, space-filling curve, algorithms by Karp, Litke, Christofides, etc., some of these algorithms assume that the cities correspond to points in the plane under some standard metric. Another group of algorithms B-opt, 3-opt, etc., aim at a local optimization: an improvement of a tour by local perturbations. The TSP also became a target for the GA community: several genetic-based algorithms were reported for optimization problems. These algorithms aim at producing near-optimal solutions by maintaining a population of potential solutions which undergoes some unary and binary transformations ('mutations' and 'crossovers') under a selection scheme biased towards fit individuals. It is interesting to compare these approaches, paying particular attention to the representation and genetic operators used – this is what we intend to do in this Unit.

Evolution of genetic algorithms for TSP has been traced. The other objective of this Unit is to help you in coding (representing as chromosome) the solution. This involves determining the length of the vector/chromosome, determining significance of each bit position and thus representation of the solution itself. The Unit would also enable you to write programmes for various components of a genetic algorithm and thus to help you in writing genetic algorithms to solve problems.

To underline some important characteristics of the TSP, it is useful to consider the CNF-satisfiability problem first. A logical expression in conjunctive normal form (CNF) is a sequence of clauses separated by the Boolean operator $\wedge$ (and/conjunction); a clause is a sequence of literals separated by the Boolean operator $\vee$ (or/disjunction); a literal is a logical variable or its negation; a logical variable is a variable that may be assigned values TRUE or FALSE (1 or 0). For example, the following logical expression is in CNF:

$$(a \vee \sim b \vee c) \wedge (b \vee c \vee d \vee \sim e) \wedge (\sim a \vee c) \wedge (a \vee \sim c \vee \sim e),$$

where, a , b , c , d , and e are logical variables; $\sim a$ denotes the negation of variable a ($\sim a$ has the value TRUE if and only if a has the value FALSE). The problem is to determine whether there exists a truth value assignment for the variables in the expression, so that the whole expression evaluates to TRUE. For the above CNF logical expression there exist several truth assignments, for which the whole expression evaluates to TRUE, e.g., any assignment with $a = \text{TRUE}$ and $c = \text{TRUE}$ irrespective of the truth values of b , d and e .

If we try to apply a genetic algorithm to the CNF-satisfiability problem, we notice that it is hard to imagine a problem with better suited representation: a binary vector of fixed length (the length of the vector corresponds to the number of variables) should do the job. Moreover, there are no dependencies between bits: any change would result in a meaningful vector representing a potential solution of the problem.

Thus we can apply mutations and crossovers without any need for decoders or repair algorithms. However, for a problem such as this one designing the evaluation function is the hardest task. Note that all logical expressions evaluate to TRUE or FALSE, and if a specific truth assignment evaluates the whole expression to TRUE, then the solution to the problem is found. The point is that during the search for a solution, all chromosomes (vectors) in a population would evaluate to FALSE (unless a solution is found), so it is impossible to distinguish between *good* and *bad* chromosomes. In short, the CNF-satisfiability problem has natural representation and operators, without any natural evaluation function.

On the other hand, the TSP has an extremely easy (and natural) evaluation function: for any potential solution (a permutation of cities), we can refer to the table providing distances between cities and (after $n-1$ addition operations) it is possible to obtain the total length of the tour. Thus, in a population of tours, it is easily possible to compare

any two of them. However, there is no rule of thumb for the choice of the representation of a tour, choice of operators and for setting the parameters.

The basic steps to solving the traveling salesman problem using a GA are as below,

- First, create a group of many random tours in what is called a **population**. The algorithm uses a greedy initial population that gives preference to linking cities that are close to each other.
- Second, **select** 2 of the better (shorter) tours as **parents** in the population and combine (**crossover**) them to make 2 new **child** tours. Hopefully, these children tour will be better than either of the parents.
- A small percentage of the times, the child tours are **mutated**. This is done to prevent all tours in the population from looking identical.
- The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same.
- New children tours are repeatedly created until the desired goal is reached.

As the name implies, Genetic Algorithms mimic nature and evolution using the principles of **Survival of the Fittest**. The two complex issues with using a Genetic Algorithm to solve the Traveling Salesman Problem are the **encoding** of the tour and the **crossover** algorithm that is used to combine the two parent tours to make the child tours.

Ordering or sequencing problems form a special type of problems whose solutions cannot be coded by a binary representation. TSP involves ordering of the cities so as to meet the objective of finding a route, in fact a shortest route. Here the task is to arrange the cities in a certain order where important thing is which elements occur next to each other (adjacency). Therefore a random list of cities need not be a valid tour. The list of connected cities may only provide a valid option for a solution. Therefore in such cases the solution is generally expressed as a permutation. If there are n cities then the representation is as a list of n cities, each of which occurs exactly once.

- Given n cities
- Find a complete tour with minimal distance or cost (as provided in a table)

Encoding:

- Label the cities $1, 2, \dots, n$
- One complete tour is one permutation
- For $n = 7$

Tours: [F, A, B, E, C, G, D], [D, E, A, C, G, B, F], etc.

The search space for this problem is BIG for a significant number of cities. For 30 cities there are $30! \approx 10^{32}$ possible tours.

Crossover as described earlier is performed by identifying a random point in the two parent sequences and switching every number in the sequence after that point.

Example 2: Consider a TSP involving 7 cities. The crossover point is between the 3rd and 4th cities in the list. To create the children, every item in the parent's sequence after the crossover point is swapped.

Parent 1 F A B | E C G D

Parent 2 D E A | C G B F

Child 1 F A B | C G B F

Child 1 D E A | E C G D

Another difficulty with the Traveling Salesman Problem is that every city can only be used once in a tour. If the letters in the above example represented cities, this child tours created by this crossover operation would be invalid. Child 1 goes to city F & B twice and never goes to cities D or E.

The encoding cannot simply be the list of cities in the order they are traveled. Other encoding methods have been created that solve the crossover problem. Although these methods will not create invalid tours, they do not take into account the fact that the tour "A B C D E F G" is the same as "G F E D C B A". To solve the problem properly the crossover algorithm will have to get much more complicated.

One way to deal with this issue could be to store the **links** in both directions for each tour. In the above tour example, Parent 1 would be stored as given in Table 2.

Table 2

City	First Connection	Second Connection
A	F	B
B	A	E
C	E	G
D	G	F
E	B	C
F	D	A
G	C	D

The crossover operation for TSP is more complicated than combining 2 strings. The crossover will take every link that exists in both parents and place those links in both children. Then, for Child 1 it alternates between taking links that appear in Parent 2 and then Parent 1. For Child 2, it alternates between Parent 2 and Parent 1 taking a different set of links. For either child, there is a chance that a link could create an invalid tour where instead of a single path in the tour there are several disconnected paths. These links must be rejected. To fill in the remaining missing links, cities are chosen at random. Since the crossover is not completely random, this is considered a greedy crossover.

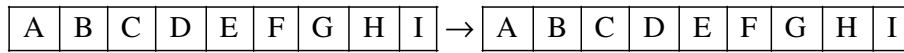
Eventually, this GA would make every solution look identical. Once every tour in the population is identical, the GA will not be able to find a better solution. There are two ways to address this problem. The first is to use a very large initial population so that it takes the GA longer to make all of the solutions the same. The second method is **mutation**, where some child tours are randomly altered to produce a new unique tour.

Normal mutation operator for permutation may lead to inadmissible solutions. Consider a bit-wise mutation. Let in a permutation gene i have value j . Then let the value of gene i be changed to some other value k would mean that k occurred twice and j no longer occurred. Therefore mutation must change at least two values. Mutation parameter then reflects the probability that some operator is applied once to the whole string rather than individually to the genes in each position. Therefore mutation for permutation is of various types:

- Insert mutation
- Swap mutation
- Inversion mutation, and
- Scramble mutation.

Insertion mutation:

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate



- Note that this preserves most of the order and the adjacency information

This Genetic Algorithm also uses a greedy initial population. The city links in the initial tours are not completely random. The GA will prefer to make links between cities that are close to each other. This is not done 100% of the time, because that would cause every tour in the initial population to be very similar.

There are 6 parameters to control the operation of the Genetic Algorithm:

- **Population Size:** The population size is the initial number of random tours that are created when the algorithm starts. A large population takes longer to find a result. A smaller population increases the chance that every tour in the population will eventually look the same. This increases the chance that the best solution will not be found.
- **Neighborhood / Group Size:** Each generation, this number of tours is randomly chosen from the population. The best 2 tours are the parents. The worst 2 tours get replaced by the children. For group size, a high number will increase the likelihood that the really good tours will be selected as parents, but it will also cause many tours to never be used as parents. A large group size will cause the algorithm to run faster, but it might not find the best solution.
- **Mutation %:** The percentage that each child after crossover will undergo **mutation**. When a tour is mutated, one of the cities is randomly moved from one point in the tour to another.
- **# Nearby Cities:** As part of a greedy initial population, the GA will prefer to link cities that are close to each other to make the initial tours. When creating the initial population this is the number of cities that are considered to be close.
- **Nearby City Odds %:** This is the percent chance that any one link in a random tour in the initial population will prefer to use a nearby city instead of a completely random city. If the GA chooses to use a nearby city, then there is an equally random chance that it will be any one of the cities from the previous parameter.
- **Maximum Generations:** How many crossovers are run before the algorithm is terminated.

Example 3: The parameter setting is given as below.

Parameter	Initial Value
Population Size	10,000
Group Size	5
Mutation	3 %
# Nearby Cities	5

Example 4: Let us suppose that order crossover #1 is to be carried out on the following two parents to create two children:

Parent 1	A	B	C		D	E	F	G		H	I
Parent 2	C	D	E		A	B	I	H		G	F
					*			*			

The steps to be followed to get two children solutions from the above two parents are:

- Step 1:** Select two crossover sites at random, as shown above.
- Step 2:** To determine child 1, the elements – A , B, I, H (lying between the two crossover sites) are directly copied from parent 2, keeping their locations and order intact.
- Step 3:** Child 1 tour then begins from parent 1, starting from the first position after the second crossover site and searching towards the initial elements of parent 1.
- Step 4:** Cities (elements) H, I, A, B are already present in child 1 solution, which should not be considered again.

The H, I, A, B and C of child 1 are selected as follows:

Child 1[H]	=	Parent 1[C]	=	C
Child 1[I]	=	Parent 1[D]	=	D
Child 1[A]	=	Parent 1[E]	=	E
Child 1[B]	=	Parent 1[F]	=	F
Child 1[C]	=	Parent 1[G]	=	G

Thus, the obtained child 1 will look as follows:

Child 1: E F G (A B I H) C D

Similarly, child 2 can be determined and it will look like the following:

Child 2: B I H (D E F G) C A

Example 5: Consider the following two parents participate in order crossover (#2).

Parent 1	A	B	C	D	E	F	G	H	I
		*		*			*	*	
Parent 2	C	D	E	A	B	I	H	G	F

Let us first explain the principle of this operator below in steps.

- Step 1:** Selected the positions – 2, 4, 7, and 8 as the key positions, at random.
- Step 2:** The elements of parent 2 at the above mentioned key positions are D, A, H, G. To determine child 1, the ordering of these elements is 4, 1, 8, 7 in parent 2 is imposed on parent 1.
- Step 3:** In parent 1, the elements – D, A, H and G found in positions – 4th, 1st, 8th and 7th, respectively.

Step 4: In child 1, the elements in these positions (that is, 1st, 4th, 7th and 8th) are selected by matching the order of the elements D, A, H, G as found in parent 2, that is,

Child 1[A] = D;

Child 1[D] = A;

Child 1[G] = H;

Child 1[H] = G.

Step 5: The remaining elements of child 1 are directly copied from parent 1.

Thus, child 1 will look like the following:

Child 1: D B C A E F H G I

Similarly, child 2 can be determined, which will look as follows:

Child 2: C B E D A I G H F

Now try the following exercise.

E3) Consider the following travelling salesman problem involving 9 cities.

Parent 1	F	I	G	E	D	C	A	H	B
Parent 2	C	B	G	I	H	F	D	E	A

Determine the children solution using:

- order crossover #1, assuming 4th and 7th sites as the crossover sites.
- order crossover #2, assuming 3rd, 5th and 7th as the key positions.

Now let us solve the TSP using cycle crossover, position-based crossover and partial mapped crossovers in the following examples.

Example 6: Consider two parents, given below, which will participate in a cycle crossover to create two children solutions.

Parent 1 A B C D E F G H I

*

Parent 2 C D E A B I H G F

The operator cycle crossover is explained in the following steps:

Step 1: To determine child 1, select a parent (say parent 1) and the starting position of the cycle (say Parent 1[C] = C), at random. Thus, the 3rd element of child 1 is nothing but element C, that is, child 1[C] = C.

Step 2: Parent 2 is then searched to check the presence of element C and it has been found in the 1st position. The first element of child 1 is selected from the first element of parent 1, that is, Child 1[A] = Parent 1[A] = A.

Step 3: Parent 2 is again searched for a presence of element A and it has occurred at the 4th position. Thus, the 4th element of parent 1 has been copied as the 4th

element of child 1, that is, Child 1[D] = Parent 1[D] = D. Similarly, we determine

$$\text{Child 1[B]} = \text{Parent 1[B]} = 2,$$

$$\text{Child 1[E]} = \text{Parent 1[E]} = \text{E}.$$

This completes one cycle because element E is seen to be present at the 3rd position of parent 2 and the corresponding 3rd position element of parent 1 is element C, which has already been selected as the starting element of the cycle.

Step 4: The remaining elements of child 1 are selected directly from parent 2, as follows:

$$\text{Child 1[F]} = \text{Parent 2[F]} = \text{I},$$

$$\text{Child 1[G]} = \text{Parent 2[G]} = \text{H},$$

$$\text{Child 1[H]} = \text{Parent 2[H]} = \text{G},$$

$$\text{Child 1[I]} = \text{Parent 2[I]} = \text{F}.$$

Thus, child 1 is found to be like the following:

Child 1 A B C D E I H G F

Using the same procedure, child 2 can be determined as follows:

Child 2 C D E A B F G H I

Example 7: Consider the following two parents, which will participate in the position-based crossover:

Parent 1	A	B	C	D	E	F	G	H	I
	*			*			*		*
Parent 2	C	D	E	A	B	I	H	G	F

The principle of position-based crossover is discussed in the following steps:

Step 1: To determine child 1, choose a number of crossover points (say 1st, 4th, 7th, 9th) on a parent, say parent 1. The elements – A, D, G, I are directly copied from parent 1 (by keeping their position information intact) to child 1. Thus, we get

$$\text{Child 1[A]} = \text{A},$$

$$\text{Child 1[D]} = \text{D},$$

$$\text{Child 1[G]} = \text{G},$$

$$\text{Child 1[I]} = \text{I}.$$

Step 2: The remaining elements of child 1 are selected from parent 2 as follows:

$$\text{Child 1[B]} = \text{Parent 2[A]} = \text{C};$$

Child 1[C] = Parent 2[C] = E, as Parent 2[B] = D has already been included in child 1;

Child 1[E] = Parent 2[E] = B, as Parent 2[D] = A has already been selected as child 1[A] element;

Child 1[F] = Parent 2[G] = H, as Parent 2[F] = I has already been considered as child 1[I];

Child 1[H] = Parent 2[I] = F, as Parent 2[H] = G has already been occurred as child 1[G].

Thus, the resulting child 1 solution will look as follows:

Child 1 A C E D B H G F I

Similarly, child 2 can be determined, which will look like the following:

Child 2 C B D A E G H I F

Example 8: Consider the following two parents, which will participate in a Partially Mapped Crossover (PMX):

Parent 1:	A	B	C		D	E	F	G		H	I
Parent 2:	C	D	E		A	B	I	H		G	F
					*					*	

The steps involved in the Partially Mapped Crossover (PMX) are:

Step 1: Select two crossover sites at random, as shown above.

Step 2: The elements – D, E, F, G of parent 1 (lying within the two crossover sites) are directly copied into child 1. Thus, we get

Child 1[D] = D,
 Child 1[E] = E,
 Child 1[F] = F,
 and Child 1[G] = G.

Step 3: The remaining elements of child 1 are determined as follows: The search starts with the elements of parent 1 residing in between the two crossover sites. The element Parent 1[D] = D is located at the 2nd position of parent 2, that is, Parent 2 [B]. Thus, 2nd position of child 1 will be filled up by an element of Parent 2 located at 4th position, that is, city 1.

Child 1[B] = Parent 2[D] = A.

Similarly, the element Parent 1[E] = E is seen to be present at the 3rd position of parent 2, that is, Parent 2[C]. Thus, the 3rd position of child 1 will be occupied by an element of parent 2 located at the 5th position, that is, city 2.

Child 1[C] = Parent 2[E] = B.

Similarly, we can determine the following elements of child 1:

Child 1[I] = Parent 2[F] = I,
 Child 1[H] = Parent 2[G] = G.

Step 4: The remaining element of child 1 is directly copied from parent 2, that is, Child 1[A] = Parent 2[A] = C.

Thus, child 1 is obtained as follows:

Child 1: C A B D E F G H I

Child 2 can be determined by following the similar procedure and is given by:

It may be noted that partially mapped crossover may sometimes give rise to a child solution, in which a particular city may occur more than once and some other cities might also be missing from it.

Now, try the following exercises.

-
- E4) Let us consider a TSP problem involving eight cities A, B, C, D, E, F, G, H. A scheduling GA with various types of crossover operator is to be used to solve the said optimization problem. Determine children solutions of the following two parents:

Parent 1: A B C D E F G H

Parent 2: C A D B F H E G

- i) Order crossover #1, assuming 2nd and 5th sites as the crossover sites.
 - ii) Order crossover #2, considering 3rd, 4th, and 7th as the key positions.
 - iii) Cycle crossover assuming 4th as the starting position.
 - iv) Position-Based Crossover considering 2nd, 4th and 6th as the crossover points.
- E5) Consider the two parents which are participating in the Partially Mapped Crossover (PMX), as shown below:

Parent 1: A B | C D E F | G H I

Parent 2: C D | E A B I | H G F

*

*

Using Partially Mapped Crossover (PMX) assuming 2nd and 6th sites as the crossover sites, find the children solution.

Now, let us summarize the unit.

12.4 SUMMARY

The content of this unit has been summarized as follows:

1. The mechanisms of different crossover and mutation operators used in the real-coded Gas by various investigators, have been explained with the help of suitable numerical examples. The working principle of a micro-GA has been discussed, in which elitism has been used.
2. In a scheduling problem, not only the position of different elements but also their adjacency and order are to be considered. A number of crossover operators have been proposed by various researchers to solve the said problem. The principles of some of these operators have been explained with the help of some appropriate examples.

12.5 SOLUTIONS/ANSWERS

- E1) i) Generate initial population by using random number generation.
ii) Select any two parents using tournament selection.
iii) The off springs are generated using the following crossover operator:

$$x = a * x + (1 - a) * y$$

$$y = a * y + (1 - a) * x$$

where a is the random number.

- iv) Apply this operation separately for each iteration to calculate the maximum fitness value.

- E2) i) Generate the initial population.
ii) Find x value and fitness function $f(x) = x^2$. The iterations are given
Iteration 1 is:

S.No.	Population	X	F(X)
1	10101	21	441
2	11011	27	729
3	01011	11	121
4	10010	18	324
5	11110	30	900
	Sum = 2517	Average = 503	Minimum = 121

Iteration 2 is:

S.No.	Population	X	F(X)
6	10010	18	324
7	10110	22	484
8	11011	27	729
9	01111	15	225
10	01011	11	121
	Sum = 1886	Average = 377	Minimum = 121

Iteration 3 is:

S.No.	Population	X	F(X)
11	10111	23	529
12	10101	21	441
13	11001	25	625
14	10101	21	441
15	10100	20	400
	Sum = 2440	Average = 488	Minimum = 400

Iteration 4 is:

S.No.	Population	X	F(X)
16	10110	22	484
17	11111	31	961
18	01111	15	225
19	10111	23	529
20	10110	22	484
	Sum = 2685	Average = 537	Minimum = 225

Iteration 5 is:

S.No.	Population	X	F(X)
21	01001	9	81
22	01111	15	225
23	01011	11	121
24	01111	15	225
25	11100	28	784
	Sum = 1436	Average = 287	Minimum = 81

Iteration 6 is:

S.No.	Population	X	F(X)
26	00000	0	0
27	00100	4	16
28	01111	15	225
29	11101	29	841
30	11010	26	676
	Sum = 1760	Average = 352	Minimum = 0

Iteration 7 is:

S.No.	Population	X	F(X)
31	10000	16	256
32	00000	0	0
33	01011	11	121
34	01010	10	100
35	00001	1	1
	Sum = 481	Average = 96	Minimum = 0

After iteration 7, minimum value is 0.

E3) i) Using order crossover #1,

Parent 1: F I G E | D C A | H B
 Parent 2: C B G I | H F D | E A

* *

Child 1: G E C A (H F D) B I
 Child 2: G I H F (D C A) E B

ii) Using order crossover #2

Parent 1: F I G E D C A H B
 * * *

Parent 2: C B G I H F D E A

The key position is III, V and VII selected at random.

Child 1: F I G E H C A D B
 Child 2: C B G I H F D E A

E4) i) Child 1: G H D B F A C E
 Child 2: H G C D E A B F

- ii) Child 1: A D C B E F G H
Child 2: A B C D F H G E
- iii) Child 1: A B C D F H G E
Child 2: C A D B E F G H
- iv) Child 1: C B A D H F E G
Child 2: C A D B E H F G

E5) The following children solutions are obtained using the PMX:

Child 1: E A (C D E F) H G I
Child 2: D E (E A B I) G H F

In child 1, city 5 has occurred twice and city 2 is missing from it. The cities lying inside the first brackets are generally not disturbed. Thus, the first element of child 1, that is, city 5 is to be replaced by the missing city, that is, 2. The modified child 1 may be written as follows:

Child 2: B A C D E F H G I

Similarly, the modified child 2 is found to be like the following:

Child 2: D C E A B I G H F

It is important to mention that the performance of different crossover operators is problem-dependent [50].

12.6 PRACTICAL ASSIGNMENT

Session 11

- Write a program in 'C' language to write the children solution of a TSP consisting of n cities using
 - Crossover order (#1);
 - Crossover order (#2).
 Test your program for E4).
- Write a program in 'C' language to write the children solution of a TSP consisting of n cities using
 - Cyclic crossover;
 - Position-based crossover.
 Test your program for E4).

12.7 REFERENCES

1. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Networks: 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*.

UNIT 13 THE SCHEMA THEORAM

Structure	Page No
13.1 Introduction	45
Objectives	
13.2 Introduction to Schema	45
13.3 Schema Theorem	49
13.4 Convergence of GA	50
13.5 Summary	51
13.6 Solutions/Answers	52
13.7 References	52

13.1 INTRODUCTION

Current research on evolutionary algorithms aims at developing sound theories that describe and predict the behavior of genetic algorithms. Traditional genetic algorithm theory is based on the schema theorem and the building block hypothesis. The schema theorem describes the behavior of a genetic algorithm by the way a genetic algorithm processes schemata. A schema S is a generalized form of a bit string consisting of 0s, 1s, and don't cares '*'. In a particular schema, e.g., "*011***", each '*' can be substituted by a 0 or 1. The schema is introduced in Sec. 13.2. The building block hypothesis states that a genetic algorithm combines those schemata that have above average fitness as well as a short defining length, i.e., the maximal distance between 0's and 1's, which would be 3 in the above example. It is believed that by combination building blocks, a genetic algorithm constructs a solution as a combination of sub solutions (low order schemata).

In Section 13.3, schema theorem is discussed. For the case of relative fitness proportional selection and one point crossover alone, the schema theorem states that the expected number is applied to find the expected number of instances of schema S . The Convergence of elitism model of GA is discussed in Sec. 13.4.

Objectives

After studying this unit you will be able to:

- define a schema;
- find the length and order of a schema;
- apply single point crossover for a schema;
- apply notation;
- describe the schema theorem;
- describe the convergence of GA.

13.2 INTRODUCTION TO SCHEMA

We have seen that Genetic Algorithms are one of the most successful engineering tools ever developed. They are easy to implement. A simple GA and GAs are surprisingly robust against lack of skill. GAs tend to be used *when nothing else works*. But this does raise interesting questions:

- Why do genetic algorithms work?
- When do they work?

We need to reason rigorously about the operation of GAs. We will look at the most important theoretical work in GAs: the **Schema Theorem**. Reasoning about complicated GAs is very hard. Fundamental analysis is largely based on examination of simple GAs i.e. Bit string representation, Proportional selection and single-point cross-over.

First we try to understand what a schema is?

A schema is a pattern comprising 0s, 1s, and *s could be either 0 or 1. Schemas capture particular subsets of strings. * stands for “I don’t care” or, if you like, “a 0 or a 1”. Some examples of schemas are

1. $S_1 = 000000$ (every bit is defined, only one string)
2. $S_2 = 111111$ (every bit is defined, only one string)
3. $S_3 = 101010$ (every bit is defined, only one string)
4. $S_4 = 0****0$ (only beginning and final bits are defined, the rest middle 4 positions may be filled by $2^4 = 16$ possible strings)

We talk about the bit string x being an *instance of a schema* S if x is in the set defined by s . Thus 010101 is an instance of 010101. 010101 is an instance of 01****. 010101 is an instance of 01**01. 010101 is an instance of *****. 010101 is **not** an instance of 11****. A string x can be an instance of many schemata.

Of how many schemas is a particular bit string an instance? 0 is an instance of $S = 0$ and also an instance of $s = *$. Similarly 1 is an instance of $s = 1$ and also an instance of $s = *$. So for each bit value in a string there are two possible element values for a containing schema. Therefore the number of schemas containing bit string x is: 2^L where L is the length of the string. Schema can be defined population of binary strings created at random. Also a schema is a subset of the space of all possible individuals for which all the genes match the template for schema S .

Example 1: Let us consider the alphabet of gene alleles and denote it by A . Here AU^* is the schema alphabet, where $*$ is the ‘wild card’ symbol, which matches any allele value. For instance the binary alphabet $A \in \{0,1,*\}$ where $* \in \{0,1\}$.

Length and order of a Schema

The **defining length** $d(S)$ of a schema S is the distance between the first defined bit and the last defined bit i.e. it is the distance between first and last non * gene in schema S .

Example 2: $d(101010) = 5$;
 $d(**10**) = 1$;
 $d(**1*1*) = 2$.

Thus length of a schema is position (last defined) – position (first defined).

The **order** $o(s)$ of a schema s is the number of *defined* bits it contains, i.e. order is the number of non * genes in schema S .

Example 3: $o(101010) = 6$;
 $o(**10**) = 2$;
 $o(**1*1*) = 2$; $o(**1*1*) = 2$.

Here it can be noted that for cardinality, k , there are $(k+1)^\ell$ schema in string of length ℓ .

-
- E1) Write the schema for the gene sequence $\{0111000\}$ and $\{1110011\}$.
- E2) Write at least 4 chromosome set, which are identified by schema $S = (01*1*)$.
- E3) Find the length and order of the following schema.
- i) $S_1 = (0*11*0**)$
 - ii) $S_2 = (*11*0**)$
 - iii) $S_3 = (**0****)$
-

After discussing schema, we are in a position to incrementally introduce the effect of the various selection and search operators associated with the above definition.

Selection of operators-fitness proportional selection

In this let us first find the probability of an individual x_k , samples schema S i.e.

$p(x_k)$. For this we assume that

- i) The probability of x_k is proportional to the number of instances of schema S in the population, and
- ii) The probability of x_k is proportional to the average fitness of schema S relative to the average fitness of all individuals in the population.

Let us consider the number of instances of S at time t , which is denoted by $N(S,t)$ and the expected number of instances of S at time t is denoted by $E(N(S,t))$. The observed average fitness of S at time t (i.e. the average fitness of instances of S in the population) is denoted by $u(S,t)$. Then the probability of selecting x_k under proportional selection is

$$p_k = F(x_k) / \sum_{j=1}^N F(x_j)$$

where $f(x_k)$ is the fitness of string s_k .

Now, the expected number of instances of schema S at time t is given by

$$\begin{aligned} E(N(S,t)) &= N \times p_k = N \times \frac{F(x_k)}{\sum_{j=1}^N F(x_j)} \\ &= F(x_k) \times \frac{N}{\sum_{j=1}^N F(x_j)} \\ &= \frac{F(x_k)}{\bar{F}(t)} \end{aligned}$$

where $\bar{F}(t)$ is the fitness of all strings in population at time t . This $E(N(s,t))$ implies that the schemas with fitness greater (lower) than the average population fitness are likely to account for proportionally more (less) of the population at the next

generation. Also it can be concluded that for the accurate estimates of expectation and probability, the population size should be infinite.

Crossover

The fitness of individuals in a population cannot be improved by reproduction, therefore let us apply single point crossover to modify the distribution of schema in the population.

Now consider crossover. Let p_c be the probability that single point cross over will be applied to a string. Instance of schema S is picked as a parent. Schema S **survives** if one of the offspring is also an instance of schema S .

Lower bound on probability $S_c(s)$ of survival of schema S is given by

$$\begin{aligned} S_c(s) &= \text{pr}(S \text{ Survive}) \\ &= \text{pr}(\text{Survive}|\text{no cross}) \text{pr}(\text{no cross}) + \text{pr}(\text{Survive}|\text{cross})\text{pr}(\text{cross}) \\ &= 1 \times (1 - p_c) + \text{pr}(\text{Survive}|\text{cross}) \times (p_c) \end{aligned}$$

We need to determine the $\text{pr}(\text{Survive}|\text{cross})$

Let L = length of genome = 7

$L-1$ = number of possible cross over points = 6

And consider $S = (* 0 * * 0 * *)$

$D(s)$ = defining length = 3

Any of the (3) points between the first and last **defined** bits is **potentially** disruptive. So the probability of randomly choosing a **potentially** destructive crossing point is $d(s)/(L-1) = 3/6 = 1/2$. The probability of **actually** causing disruption must be **less than** this. And so the probability of **surviving** crossover **given that you do crossover** must be **greater than**

$$\left(1 - \frac{d(s)}{L-1}\right)$$

And so the $\text{pr}(\text{Survive}|\text{cross})\text{pr}(\text{cross})$ must be greater than or equal to

$$\left(1 - \frac{d(s)}{L-1}\right) p_c = p_c \left(1 - \frac{d(s)}{L-1}\right)$$

And so $\text{pr}(\text{Survive}) = S_c(s)$ must satisfy

$$S_c(s) \geq (1 - p_c) + p_c \left(1 - \frac{d(s)}{L-1}\right)$$

$$S_c(s) \geq \left(1 - p_c \left(\frac{d(s)}{L-1}\right)\right)$$

Mutation

In the special worst case lower bound $P_c = 1$

Now we consider the disruptive effects of **mutation**. Mutation is applied gene by gene. Let p_m be the probability of flipping any bit in a string. A schema s is disrupted when any **defined** bit is flipped. There are $o(s)$ bits defined, and so the probability they all non * genes given by survive is

$$S_m(s) = (1 - p_m)^{o(s)}$$

The probability of applying the mutation operator is very very less, therefore $(1 - pm)^{o(s)} \approx 1 - o(s)pm$.

The Schema Theorem

Note: If a bit is in a position corresponding to a * in the schema template it does matter whether or not it is flipped.

Now let us discuss schema theorem, which is also known as fundamental theorem of Genetic Algorithm.

13.3 THE SCHEMA THEOREM

The completed schema theorem can now be stated.

The Schema Theorem

The expected number of schema S at generation $t + 1$ with proportional selection, single point crossover and genewise mutation is given by

$$E(N(s, t + 1)) \geq \frac{\hat{u}(s, t)}{\bar{F}(t)} N(s, t) \left[1 - p_c \left(\frac{d(s)}{L - 1} \right) \right] (1 - p_m)^{o(s)}$$

Conservatism

Note that the schema theorem is conservative.

1. Only deals with *destructive* effects.
2. Evolution may also be *constructive* effects.
3. Non-s schema instances may combine to give s-schema instances.
4. Similar properties are valid for mutation.

Implications of Schema theorem

Schemas, like families, need nourishment and encouragement and careful protective management

- The more bits in your building block family the more likely one is to go off the rails (causing great frustration and heartache).
- Genes living far apart are prone to breaking up
- Conducting a constructive relationship at a distance is hard.
- Best results achieved by the family unit huddled together in consecutive positions.

Applications of Schema Theorem

The schema theorem is more applicable at the early stages of a search rather than at the end. Schema theorem indicates that fitter than average schemas are rewarded. The fitter the schema the more it is rewarded.

- Reward is immediate: you see it in the next generation.
- Should alert us to one danger immediately.
- Premature converge of the population.

Now try the following exercise.

-
- E4) What implications does the schema theorem have for your choice of mutation rates?
-

So far we have discussed the schema theorem. In the following section, we shall discuss the convergence of the elitism model of genetic algorithms.

13.4 CONVERGENCE OF GA

Goldberg (1989) explained the convergence criteria of a GA from the viewpoint of schema. As we have discussed that a schema represents a subset of strings with similarities at certain positions. In general, any specific string is simultaneously an instance of 2^n schemata, where n is the length of the string. We can always associate a fitness value with a schema, i.e. the average fitness of the schema because schema is a representation of a robust of strings. Hence, a schema's average fitness value is different for each population from one generation to another.

GAs research can be visualized for the optimal strings as a simultaneous competition among schemata increases the number of their instances in the population.

The winners of individual schema competitions may form the optimal string, such schemata are called **building blocks**. A number of such building blocks in various parts along the string get emphasized by applying the genetic operators on a population of string. But there is every danger that the crossover operator may destroy good schemata. So, selection of good appropriate crossover operator plays a vital role here. If the cross site cuts the well defined position in the schema, this may not be preserved in the next generation unless otherwise. Second parent also will have the same schema in that position. In case of single-point crossover, falling of cross-sites within the defined positions has less probability while in the case of uniform crossover, disturbance of good schema takes place with a higher probability. As far as GA to engineering field is concerned single- and two-point crossover are common. The meaning of search in the genetic space is the development of building blocks. Building blocks are combined together due to combined action of genetic operators to form bigger and better building blocks and finally converge to the optimal solution.

If we see the processing of individual schema, it is found that the worst schema is not able to survive. So it depends totally on the selection of the genetic operators. For example if a cross site cuts a particular well defined position in a schema, it may be a reason for not preserving it in the next generation, unless the second parent has same schema at that position. In genetic spaces the search is only for the development of building blocks. To form bigger and better building blocks which finally converge to the optimal solution, the building blocks are combined together due to combined action of genetic operators. The corresponding GA cycle is shown in Fig 1.

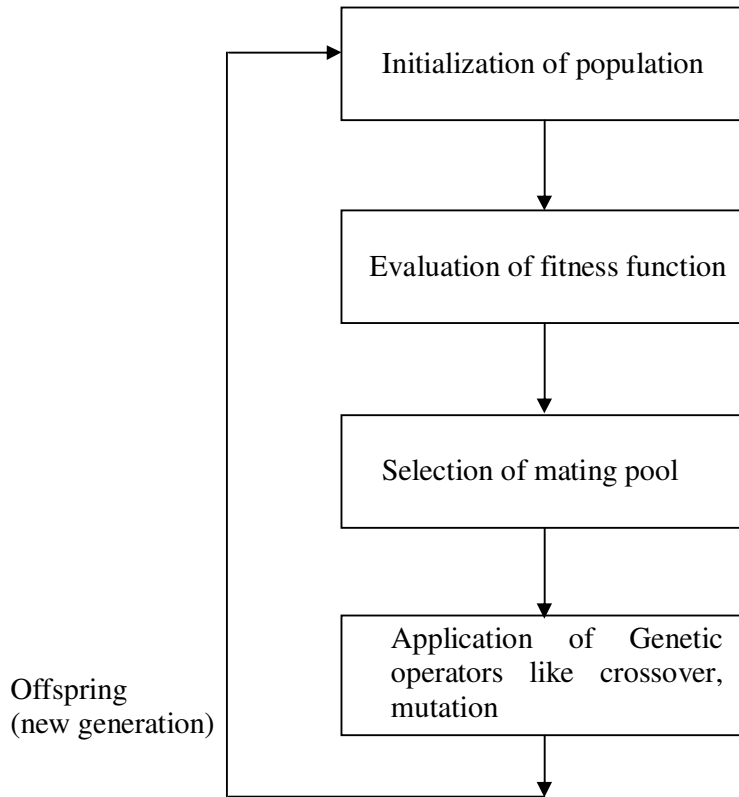


Fig. 1: The GA cycle

The situation of good strings in a population set and random information exchange among good strings are simple and straightforward. No mathematical proof has been derived yet for convergence of GA. Rajeev and Krishnamoorthy (1992), discussed one criterion for convergence. When a fixed percentage of columns and rows in population matrix becomes the same, it can be assumed that convergence is attained. The fixed percentage may be 80% or 85%.

In genetic algorithms, as we proceed with more generations, there may not be much improvement in the population fitness. As the generation progresses, the population gets filled with more fit individuals with only slight deviation from the fitness of best individuals so far found, and the average fitness comes very close to the fitness of the best individuals. We can specify some fixed number of generations after getting the optimum point to confirm that there is no change in the optimum in the subsequent generations.

Now let us summarise the unit.

13.5 SUMMARY

1. A schema is a subset of the space of all possible individuals for which all the genes match the template for schema.
2. Schema order, $o(S)$, is the number of non '*' genes in schema S .
3. Schema Defining Length, $d(S)$, is the distance between first and last non '*' gene in schema S .

4. Schemas with fitness greater (lower) than the average population fitness are likely to account for proportionally more (less) of the population at the next generation.
5. Reproduction does nothing to improve the fitness of individuals in a population. (Single point Crossover was the first of two search operators introduced to modify the distribution of schema in the population).
6. Mutation is applied gene by gene. In order for schema S to survive, all non * genes in the schema must remain unchanged.
7. The expected number of schema S at generation $t + 1$ when using a canonical GA with proportional selection, single point crossover and gene wise mutation (where the latter are applied at rates p_c and p_m) is defined in schema theorem.

13.6 SOLUTIONS/ANSWERS

E1) $S = (*11*0**)$

E2) The chromosome sets are (01010), (01011), (01110) and (01111).

E3) i) $d(S_1) = 7 - 1 = 6$

$o(S_1) = 4$

ii) $d(S_2) = 5 - 2 = 3$

$o(S_2) = 3$

iii) $d(S_3) = 4 - 4 = 0$

$o(S_3) = 1$

E4) Mutation is used to promote intensification (where local optimisation is required) and of course, to promote diversity.

13.7 REFERENCES

1. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Networks: Fuzzy Logic, and Genetic Algorithms*.
2. D.K. Pratihari, (2008), *Soft Computing*.
3. J. –S. R. Jang, C. –T. Sun and E. Mizutani, (2004), *Neuro-Fuzzy and Soft Computing*.
4. David Goldberg, (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*.