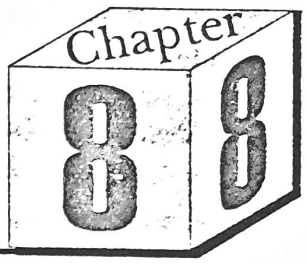
Fundamentals of Genetic Algorithms



Decision-making features occur in all fields of human activities such as scientific and technological and affect every sphere of our life. Engineering design, which entails sizing, dimensioning, and detailed element planning is also not exempt from its influence.

For example an aircraft wing can be made from aluminium or steel and once material and shape are chosen, there are many methods of devising the required internal structure. In civil engineering also, designing a roof to cover large area devoid of intermediate columns requires optimal designing.

The aim is to make objective function a maximum or minimum, that is, it is required to find an element X_0 in A if it exists such that

$$F(X_0) \leq F(X) \text{ for minimization}$$
$$F(X) \leq F(X_0) \text{ for maximization} \qquad (8.1)$$

The following major questions arise in this process

- Does an optimal solution exist?
- Is it unique?
- What is the procedure?
- How sensitive the optimal solution is?
- How the solution behaves for small changes in parameters?

Since 1940, several optimization problems have not been tackled by classical procedures including:

1. Linear programming
2. Transportation
3. Assignment
4. Nonlinear programming
5. Dynamic programming
6. Inventory
7. Queuing
8. Replacement
9. Scheduling

The classification of optimization techniques is shown in Fig. 8.1. Basically, we have been following traditional search technique for solving nonlinear equations. Figure 8.2 shows the classes of both traditional and nontraditional search techniques. Normally, any engineering problem will have a large number of solutions out of which some are feasible and some are infeasible. The designer's task is to get the best solution out of the feasible solutions. The complete set of feasible solutions constitutes feasible design space and the progress towards the optimal design involves some kind of search within the space (combinatorial optimization). The search is of two kinds, namely deterministic and stochastic.

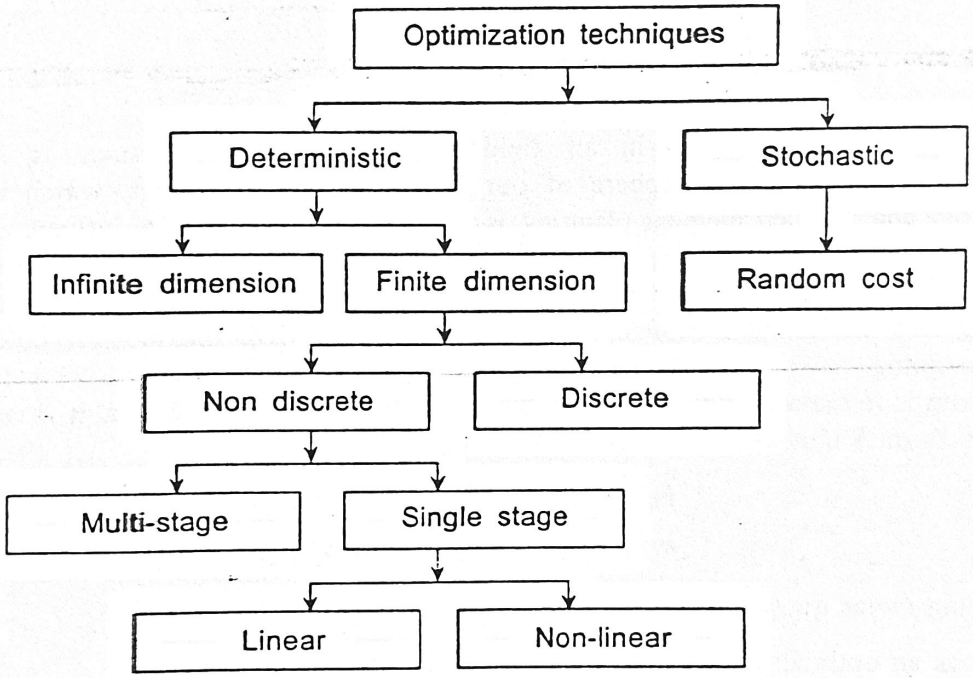


Fig. 8.1 Classification of optimization techniques.

In the case of *deterministic search,* algorithm methods such as steepest gradient methods are employed (using gradient concept), whereas in *stochastic approach*, random variables are introduced. Whether the search is deterministic or stochastic, it is possible to improve the reliability of the results where reliability means getting the result near optimum. A transition rule must be used to improve the reliability. Algorithms vary according to the transition rule used to improve the result.

Nontraditional search and optimization methods have become popular in engineering optimization problems in recent past. These algorithms include:

1. Simulated annealing (Kirkpatrik, et al. 1983)
2. Ant colony optimization (Dorigo and Caro, 1999)
3. Random cost (Kost and Baumann, 1999)
4. Evolution strategy (Kost, 1995)
5. Genetic algorithms (Holland, 1975)
6. Cellular automata (Wolfram, 1994)

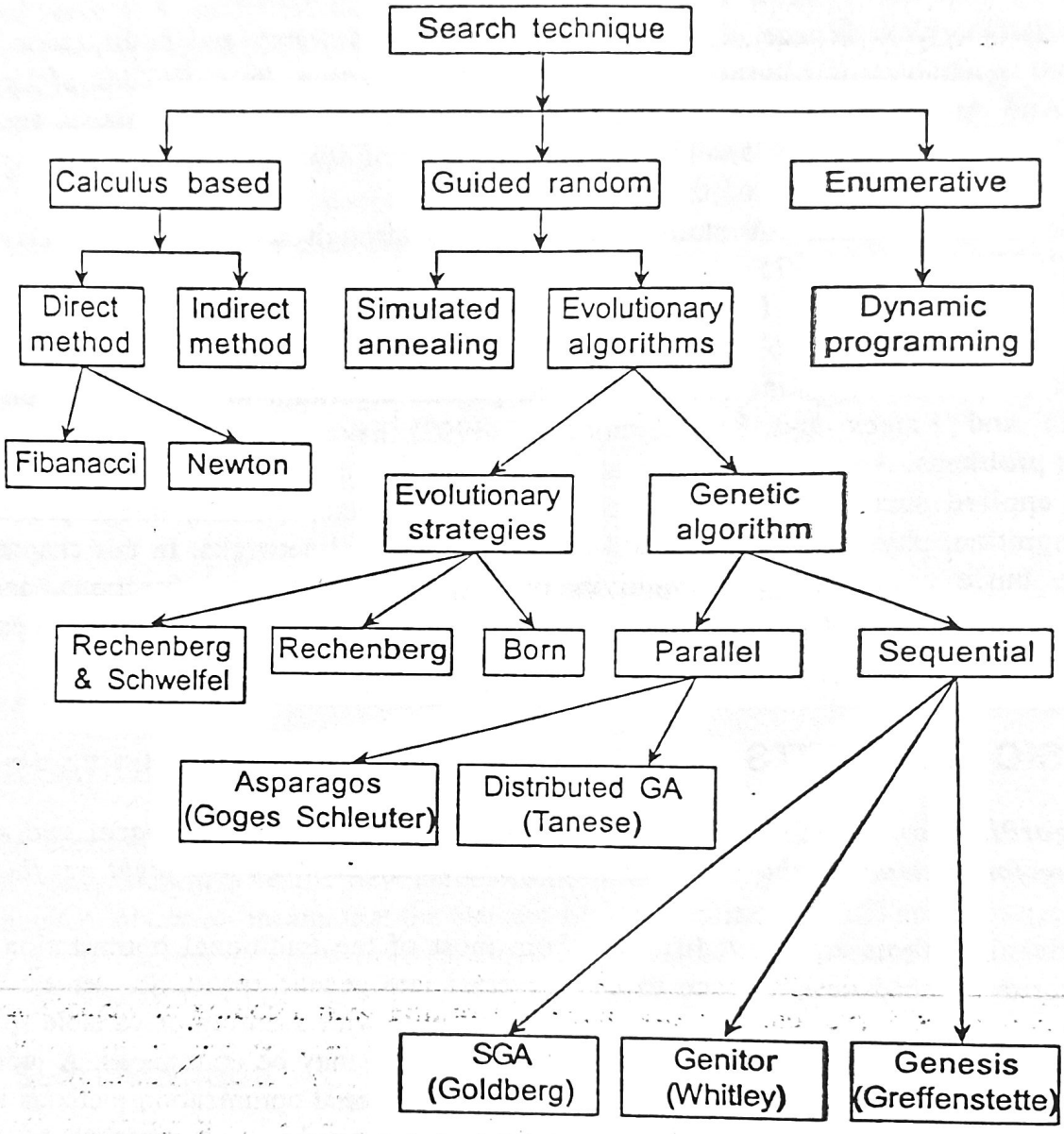


Fig. 8.2 Classes of search techniques.

Simulated annealing mimics the cooling phenomenon of molten metals to constitute a search procedure. *Genetic algorithm* and *evolutionary strategies* mimic the principle of natural genetics and natural selection to construct search and optimization procedures. The collective behaviour that emerges from a group of social insects such as ants, bees, wasps, and termites has been dubbed as *Swarm intelligence*. The foraging of ants has led to a novel algorithm called *Ant colony optimization* for rerouting network traffic in busy telecommunication systems. This method was originally developed by Deneubourg and extended by Dorigo (1999) of Brussels. Random cost method is a stochastic algorithm which moves as enthusiastically uphill as down-hill. The method has no severe problems in escaping from a dead end and is able to find the optima. In this chapter, we discuss the fundamentals of genetic algorithms.

8.1 GENETIC ALGORITHMS: HISTORY

The idea of evolutionary computing was introduced in 1960 by I. Rechenberg in his work

Evolutionary strategies. Genetic algorithms are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. Prof. Holland of University of Michigan, Ann Arbor, envisaged the concept of these algorithms in the mid-sixties and published his seminal work (Holland, 1975). Thereafter, a number of students and other researchers have contributed to the development of this field.

To date, most of the GA studies are available through some books by Davis (1991), Goldberg (1989), Holland (1975), Michalewicz (1992) and Deb (1995) and through a number of conference proceedings. The first application towards structural engineering was carried by Goldberg and Samtani (1986). They applied genetic algorithm to the optimization of a ten-member plane truss. Jenkins (1991) applied genetic algorithm to a trussed beam structure. Deb (1991) and Rajeev and Krishnamoorthy (1992) have also applied GA to structural engineering problems. Apart from structural engineering there are many other fields in which GAs have been applied successfully. It includes biology, computer science, image processing and pattern recognition, physical science, social sciences and neural networks. In this chapter, we will discuss the basic concepts, representatives of chromosomes, fitness functions, and genetic inheritance operators with example. In Chapter 9, genetic modelling for real life problems will be discussed.

8.2 BASIC CONCEPTS

Genetic algorithms are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solutions which we might not find in a life time.

Genetic algorithms are very different from most of the traditional optimization methods. Genetic algorithms need design space to be converted into genetic space. So, genetic algorithms work with a coding of variables. The advantage of working with a coding of variable space is that coding discretizes the search space even though the function may be continuous. A more striking difference between genetic algorithms and most of the traditional optimization methods is that GA uses a population of points at one time in contrast to the single point approach by traditional optimization methods. This means that GA processes a number of designs at the same time. As we have seen earlier, to improve the search direction in traditional optimization methods, transition rules are used and they are deterministic in nature but GA uses randomized operators. Random operators improve the search space in an adaptive manner.

Three most important aspects of using GA are:

1. definition of objective function
2. definition and implementation of genetic representation
3. definition and implementation of genetic operators.

Once these three have been defined, the GA should work fairy well beyond doubt. We can, by different variations, improve the performance, find multiple optima (species if they exist) or parallelize the algorithms.

8.2.1 Biological Background

All living organisms consist of cells. In each cell, there is a set of chromosomes which are strings

of DNA and serve as a model for the whole organism. A chromosome consists of genes on blocks of DNA as shown in Fig. 8.3. Each gene encodes a particular pattern. Basically, it can be said that each gene encodes a trait, e.g. colour of eyes. Possible settings of traits (bluish brown eyes) are called *alleles*. Each gene has its own position in the chromosome search space. This position is called *locus*. Complete set of genetic material is called *genome* and a particular set of genes in genome is called *genotype*. The genotype is based on organism's phenotype (development after birth), its physical and mental characteristics such as eye colour, intelligence and so on.

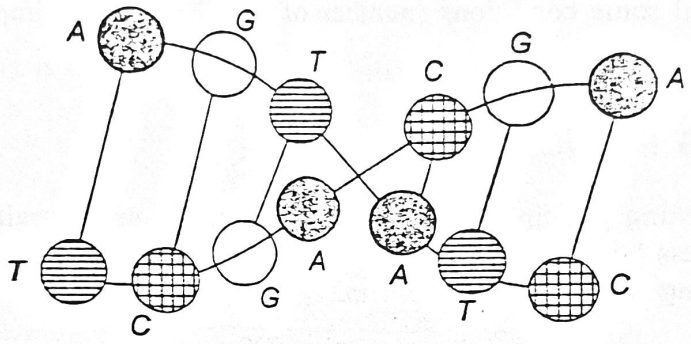


Fig. 8.3 Genome consisting of chromosomes.

8.3 CREATION OF OFFSPRINGS

During the creation of offspring, recombination occurs (due to cross over) and in that process genes from parents form a whole new chromosome in some way. The new created offspring can then be mutated. *Mutation* means that the element of DNA is modified. These changes are mainly caused by errors in copying genes from parents. The *fitness* of an organism is measured by means of success of organism in life.

8.3.1 Search Space

If we are solving some problems, we work towards some solution which is the best among others. The space for all possible feasible solutions is called *search space*. Each solution can be marked by its value of the fitness of the problem. 'Looking for a solution' means looking for extrema (either maximum or minimum) in search space. The search space can be known by the time of solving a problem and we generate other points as the process of finding the solution continues (shown in Fig. 8.4).

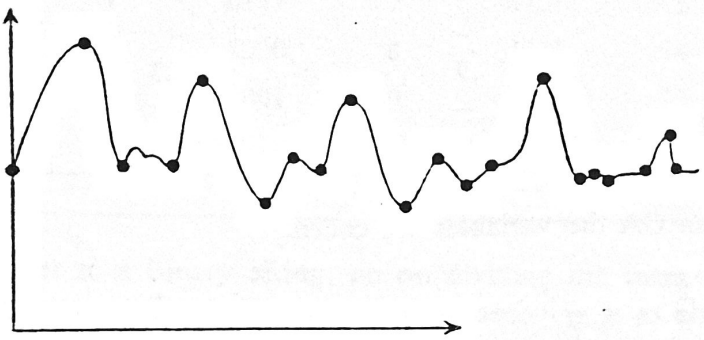


Fig. 8.4 Examples of search space.

The problem is that, search space is complicated and one does not know where to look for the solution or where to start from and this is where genetic algorithm is useful. GAs are inspired by *Darwinian theory of the survival of the fittest.* Algorithm is started with a set of solutions (represented by chromosomes) called *populations.* Solutions for one population are taken and used to form a new population. This is motivated by a hope that new population will be better than the old one. Solutions, which are selected to form new population (offspring), are selected according to their fitness. The more suitable they are, the more chances they have to reproduce. This is repeated until some conditions (number of populations) for improvement of best solution are satisfied.

8.4 WORKING PRINCIPLE

To illustrate the working principle of GA, we first consider unconstrained optimization problem. Later, we shall discuss how GA can be used to solve a constrained optimization problem. Let us consider the following maximization problem.

$$\text{maximize } f(X) \tag{8.2}$$

$$X_i^{(L)} \le X_i \le X_i^{(U)} \text{ for } i = 1, 2, ..., N$$

If we want to minimize $f(X)$, for $f(X) > 0$, then we can write the objective function as

$$\text{maximize } \frac{1}{1 + f(X)} \tag{8.3}$$

If $f(X) < 0$ instead of minimizing $f(X)$, maximize $\{-f(X)\}$. Hence, both maximization and minimization problems can be handled by GA.

If the same problem is solved by multiple regression analysis, given k independent variables, for regressing the dependent variable $2^{(k+1)} - 1$ including the intercept which are given in Table 8.1.

Table 8.1 Subsets for regression analysis

Variable	Subsets
2	7
3	15
–	–
–	–
9	1023
–	–
19	10,48,578

On the other hand, in GA the variables are coded.

8.5 ENCODING

There are many ways of representing individual genes. Holland (1975) worked mainly with string bits but we can use arrays, trees, lists or any other object. Here, we consider only bit strings.

8.5.1 Binary Encoding

Example Problem (Knapsack Problem)

There are things with given values and size. The knapsack has a given capacity. Select things to minimize their value in knapsack not exceeding the capacity of the knapsack.

Encoding

Each bit says if the thing is in knapsack or not. Binary coding is the most commonly used in GA as shown in Table 8.2.

Table 8.2 Chromosomes

Chromosome *A*	101101100011
Chromosome *B*	010011001100

Binary encoding gives many possible chromosomes even with small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after genetic operator corrections.

In order to use GA to solve the maximization or minimization problem, unknown variables X_i are first coded in some string structures. It is important to mention that coding of the variable is not absolutely necessary. There exist some studies where GAs are directly used on the variables themselves, but here we shall ignore the exceptions and discuss the encoding for simple genetic algorithm. Binary-coded strings having 1s and 0s are mostly used. The length of the string is usually determined according to the desired solution accuracy. For example, 4-bit binary string can be used to represent 16 numbers as shown in Table 8.3.

Table 8.3 Four-bit string

4-bit string	Numeric value	4-bit string	Numeric value	4-bit string	Numeric value
0000	0	0110	6	1100	12
0001	1	0111	7	1101	13
0010	2	1000	8	1110	14
0011	3	1001	9	1111	15
0100	4	1010	10		
0101	5	1011	11		

To convert any integer to a binary string, go on dividing the integer by 2 as shown in Fig. 8.5. We get equivalent integer for the binary code by decoding it as shown in Fig. 8.6.

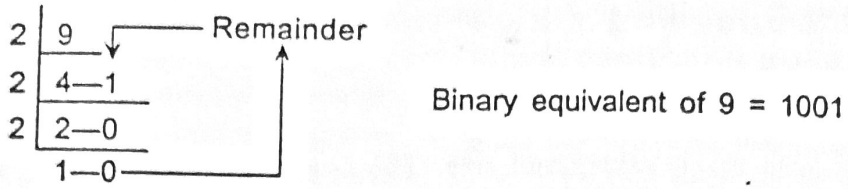


Fig. 8.5 Binary coding.

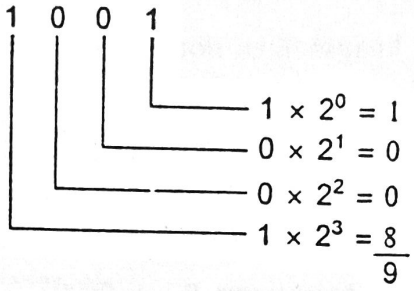


Fig. 8.6 Equivalent integer for a binary code.

For example, if we want to code a two variable function assuming four bits are used for each variable, we represent the two variables X_1, X_2 as (1011 0110). As given in Eq. (8.2), every variable will have both upper and lower limits as

$$X_i^L \le X_i \le X_i^U \tag{8.4}$$

As shown in Table 8.3 a four-bit string can represent the integers from 0 to 15 (16 elements) and hence, (0000 0000) and (1111 1111) represent the points for X_1, X_2 as (X_1^L, X_2^L); (X_1^U, X_2^U) respectively because the substrings (0000) and (1111) have the minimum and the maximum decoded values. Hence, an n-bit string can represent integers from 0 to $2^n - 1$, i.e. 2^n integers. Assume that X_i is coded as a substring S_i of length n_i. The decoded value of a binary substring S_i is calculated as shown in Fig. 8.6 as

$$\sum_{k=0}^{k=n_i-1} 2^k s_k \tag{8.5}$$

where s_i can be either zero or 1 and the string S is represented as

$$s_{n-1} \cdots s_3 s_2 s_1 s_0 \tag{8.6}$$

For example, a four-bit string (0111) has a decoded value equal to

$$2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = 7$$

Knowing X_i^L and X_i^U corresponding to (0000) and (1111), the equivalent value for any 4-bit string can be obtained as

$$X_i = X_i^L + \frac{(X_i^U - X_i^L)}{(2^{n_i} - 1)} \times (\text{decoded value of string}) \tag{8.7}$$

Assume for a variable X_i, $X_i^L = 2$, and $X_i^U = 17$, to find what value of 4-bit string of $X_i = (1010)$ would represent. First we get the decoded value for S_i as

$$S_i = 1010 = 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 = 10 \tag{8.8a}$$

$$X_i = 2 + \frac{(17-2)}{(2^4-1)} \times 10 = 12 \tag{8.8b}$$

Hence, the accuracy that can be obtained with a four-bit code is 1/16th of search space. But as the string length is increased by one, the obtained accuracy increases exponentially to 1/32th of the search space. It is not necessary to code all variables in equal substring length. The length of substring representing a variable depends on the desired accuracy in that variable. Generalizing the concept, we may say that with n_i bit-length coding for a variable, the obtainable accuracy in that variable approximation is $(X_i^U - X_i^L)/2^{n_i}$. Once the coding of the variables is done, the corresponding point $(X_1 \ldots X_n)^T$ can be found out using Eq. (8.7). For continuous design variable, if ε is the precision representation required then string length 'S' should be equal to

$$S = \log_2\left(\frac{X^U - X^L}{\varepsilon}\right) \tag{8.9}$$

In some cases, X_i need not be equally distributed so as to apply the linear mapping rule. Hence, X_i can be given in the form of a table as shown in Table 8.4.

Table 8.4 Binary representation of fibre angles

S.No.	Binary coding	Decoded value	Fibre angle
1	0000	0	0
2	0001	1	10
3	0010	2	20
4	0011	3	30
5	0100	4	45
6	0101	5	60
7	0110	6	70
8	0111	7	80
9	1000	8	90
10	1001	9	−10
11	1010	10	−20
12	1011	11	−30
13	1100	12	−45
14	1101	13	−60
15	1110	14	−70
16	1111	15	−80

Hence, when the values are not uniformly distributed, tabulated values can be used to find the corresponding point $X = (X_1, X_2, \ldots, X_n)^T$. Thereafter, the function value at that point X can also be calculated by substituting X in the given objective function.

8.5.2 Octal Encoding (0 to 7)

To convert any integer to an octal string, go on diving the integer by 8 as shown in Fig. 8.7. For example, 542 is given in octal form as 1036.

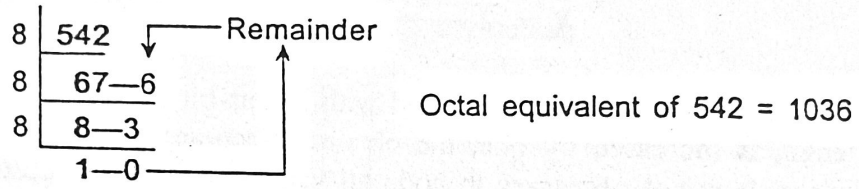


Fig. 8.7 Octal encoding.

For the octal code, we can get the equivalent integer by decoding it as shown in Fig. 8.8. The integer value for the octal code 1036 is 542.

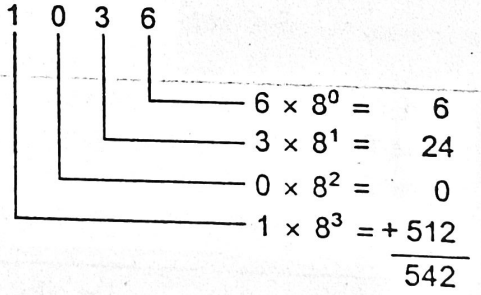


Fig. 8.8 Equivalent integer for an octal code.

A four-bit octal string can represent the integers from 0 to 4095 and hence, (0000 0000) and (7777 7777) would represent the points for X_1 and X_2 as (X_1^L, X_2^L); (X_1^U, X_2^U) respectively. The decoded value of a binary substring S_i is calculated as

$$\sum_{k=0}^{k=n_i-1} 8^k s_k \tag{8.10}$$

and hence, the obtainable accuracy in that variable approximation is $(X_i^U - X_i^L)/8^{n_i}$.

5.3 Hexadecimal Encoding (0123456789ABCDEF)

To convert any number to hexadecimal form, we go on dividing the number by 16 as shown in Fig. 8.9. The hexadecimal code for 67897 is shown to be 10939. We get equivalent integer for the hexadecimal code by decoding it as shown in Fig. 8.10. The decoded value for the hexadecimal number BO79E6 is 11565542.

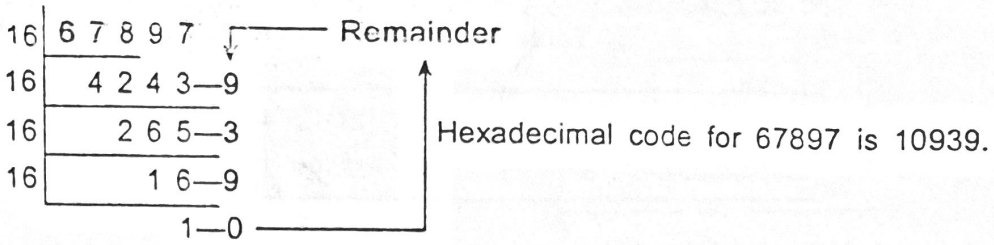


Fig. 8.9 Hexadecimal coding.

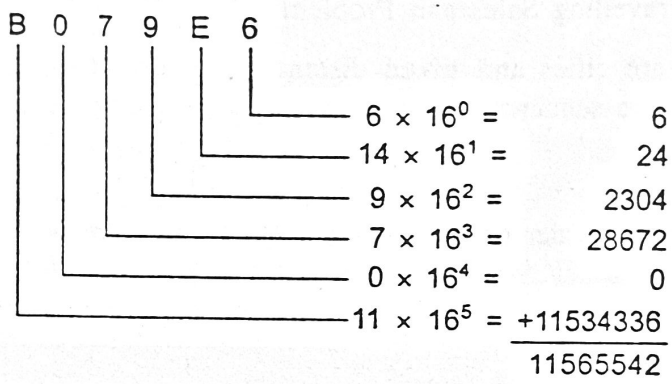


Fig. 8.10 Equivalent integer for hexadecimal code.

A four-bit hexadecimal can represent the integers from 0 to 65535 and hence, (0000 0000) and (FFFF FFFF) would represent the points for X_1 and X_2 as (X_1^L, X_2^L); (X_1^U, X_2^U) respectively. The decoded value of a hexadecimal string S_i is calculated as

$$\sum_{k=0}^{k=n_i-1} 16^k s_k \tag{8.11}$$

And hence, the obtainable accuracy in that variable approximation is $(X_i^U - X_i^L)/16^{n_i}$. From the above discussion it is clear that encoding can be given to any base 'b', bits of n_i length can represent the integers from 0 to $(b^{ni} - 1)$ and hence (0000 0000), and $((b-1)(b-1)(b-1)(b-1)$, and $(b-1)(b-1)(b-1)(b-1))$ would represent the points X_1 and X_2 as (X_1^L, X_2^L); (X_1^U, X_2^U) respectively. The decoded value of 'b' bit-string S_i is calculated as

$$\sum_{k=0}^{k=n_i-1} b^k s_k \tag{8.12a}$$

And hence, obtainable accuracy in that variable approximation is

$$(X_i^U - X_i^L)/b^{n_i} \tag{8.12b}$$

8.5.4 Permutation Encoding

This can be used in ordering problems such as travelling salesman or task ordering. In a *permutation encoding*, every chromosome is a string of numbers which represents the number in the sequence as shown in Table 8.5.

Table 8.5 Permuation encoding

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

Even for ordering problems after applying for sometimes, the genetic operators corrections must be made to leave the chromosome consistent.

Example Problem Travelling Salesman Problem

The problem: There are cities and given distances between them. Travelling salesman has to visit all of them. Find the sequence of cities to minimize the travelling distance.

Encoding

Chromosome illustrates the order of cities in which the salesman would visit them.

8.5.5 Value Encoding

In this, every chromosome is a string of some values and the values can be any thing connected to the problem. From numbers, real numbers characterize some complicated objects as shown in Table 8.6.

Table 8.6 Value encoding

Chromosome–*A*	1.234	5.3243	0.4556	2.0253
Chromosome–*B*	abdjetijdhj...			
Chromosome–*C*	(Back),	(Right),	(Forward),	(Left)

Value encoding is very good for some special problems. On the other hand, this encoding is often necessary to develop new genetic operators specific to the problem.

Example Find the weights of neural network.

The problem: To find the weights of synapses connecting input to hidden layer and hidden layer to output layer.

Encoding

Each value in chromosome represents the corresponding weights.

8.5.6 Tree Encoding

This is mainly used 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 as shown in Fig. 8.11. Tree encoding is good for evolving programs in a programming language. LISP is often used because programs in it are represented in this form and can easily be parsed as a tree so that functions and genetic operators can be applied rather easily.

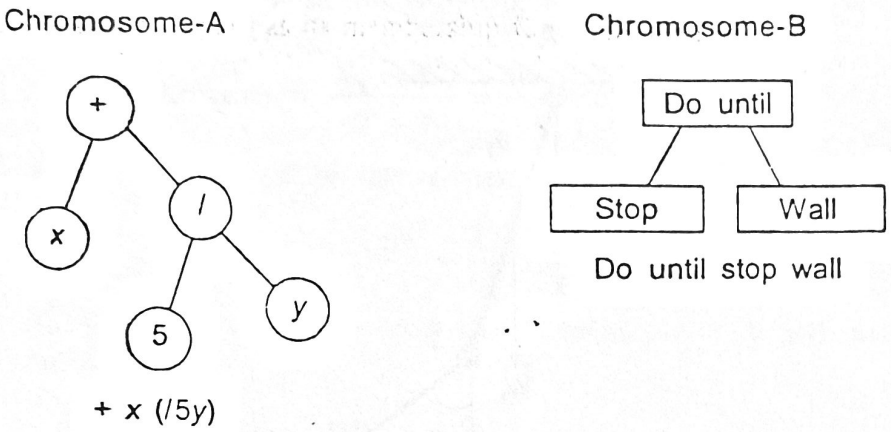


Fig. 8.11 Tree encoding.

Example Find the function for a given value.

Problem: Some input and output values are given. The task is to find the function which will give the best relation to satisfy all values.

Encoding

Chromosomes are functions represented in a tree.

8.6 FITNESS FUNCTION

As pointed out earlier GAs mimic the Darwinian theory of survival of the fittest and principle of nature to make a search process. Therefore, GAs are usually suitable for solving maximization problems. Minimization problems are usually transformed into maximization problems by some suitable transformation. In general, fitness function $F(X)$ is first derived from the objective function and used in successive genetic operations.

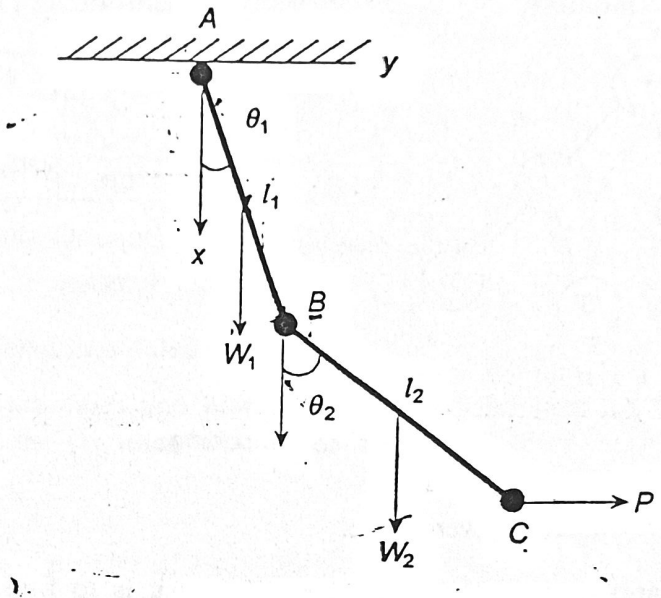
Certain genetic operators require that fitness function be non-negative, although certain operators do not have this requirement. Consider the following transformations

$$\begin{aligned} F(X) &= f(X) \text{ for maximization problem} \\ F(X) &= 1/f(X) \text{ for minimization problem, if } f(X) \neq 0 \\ F(X) &= 1/(1 + f(X)), \quad \text{if } f(X) = 0 \end{aligned} \tag{8.13}$$

A number of such transformations are possible. The fitness function value of the string is known as *string's fitness.*

Example 8.1

Two uniform bars are connected by pins at A and B and supported at A. A horizontal force P acts at C. Knowing the force, length of bars and its weight determine the equilibrium configuration of the system if friction at all joints are neglected (see Fig. 8.12).



P = 2
W_1 = 2
W_2 = 2
l_1 = 2
l_2 = 2

Fig. 8.12 Two bar pendulum.

The total potential for the two bar pendulum is written as

$$\Pi = -P[(l_1\sin\theta_1 + l_2\sin\theta_2)] - \frac{W_1 l_1}{2}\cos\theta_1 - W_2\left[\frac{l_2}{2}\cos\theta_2 + l_1\cos\theta_1\right] \tag{8.14}$$

Substituting the values for P, W_1, W_2, and for the lengths as 2 we get,

$$\Pi(\theta_1, \theta_2) = -4\sin\theta_1 - 6\cos\theta_1 - 4\sin\theta_2 - 2\cos\theta_2 \tag{8.15a}$$

$$0 \le \theta_1, \theta_2 \le 90 \tag{8.15b}$$

Equilibrium configuration is the one which makes Π a minimum.

Theoretical solution

$\delta\Pi = 0$, for Π to be maximum or minimum

$$\delta\Pi = \frac{\partial\Pi}{\partial\theta_1}\delta\theta_1 + \frac{\partial\Pi}{\partial\theta_2}\delta\theta_2 = 0 \tag{8.16}$$

$\delta\theta_1$, $\delta\theta_2$ are arbitrary. Therefore we get,

$$\frac{\partial\Pi}{\partial\theta_1} = 4\cos\theta_1 - 6\sin\theta_1 = 0 \tag{8.17a}$$

$$\frac{\partial\Pi}{\partial\theta_2} = 4\cos\theta_2 - 2\sin\theta_2 = 0 \tag{8.17b}$$

From Eq. (8.17(a)) and (b) we get,

$$\tan\theta_1 = \frac{2}{3},\ \theta_1 = 33.7^\circ\ (0.558 \text{ radians})$$

$$\tan\theta_2 = 2,\ \theta_2 = 63.43^\circ\ (1.107 \text{ radians}) \tag{8.18}$$

For which $\Pi = -11.68$

Since there are two unknowns θ_1 and θ_2 in this problem, we will use 4-bit binary string for each unknown.

$$\text{Accuracy} = \frac{X^U - X^L}{2^4 - 1} = \frac{90}{15} = 6^\circ \tag{8.19}$$

Hence, the binary coding and the corresponding angles are given as

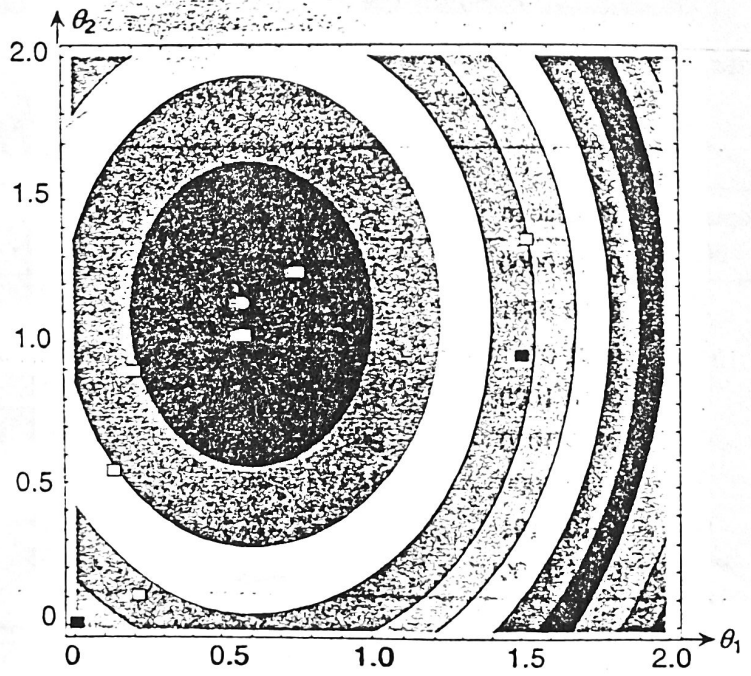
$$X_i = X_i^L + \frac{X_i^U - X_i^L}{2^4 - 1} S_i \tag{8.20}$$

where S_i is the decoded value of the ith chromosome. The binary coding and the corresponding angles are given in Table 8.7.

Table 8.7 Binary coding and the corresponding angles

S. no.	Binary coding	Angle	S. no.	Binary coding	Angle
1	0000	0	9	1000	48
2	0001	6	10	1001	54
3	0010	12	11	1010	60
4	0011	18	12	1011	66
5	0100	24	13	1100	72
6	0101	30	14	1101	78
7	0110	36	15	1110	84
8	0111	42	16	1111	90

The objective function of the problem is given in Eq. (8.15). The contours of the objective function as well as the 3D plot are shown in Figs. 8.13(a) and (b) respectively.



■ Left out population from initial population ○ Minimum point (0.588, 1.107) □ Population chosen for mating pool

Fig. 8.13(a) Contours of equal objective functions.

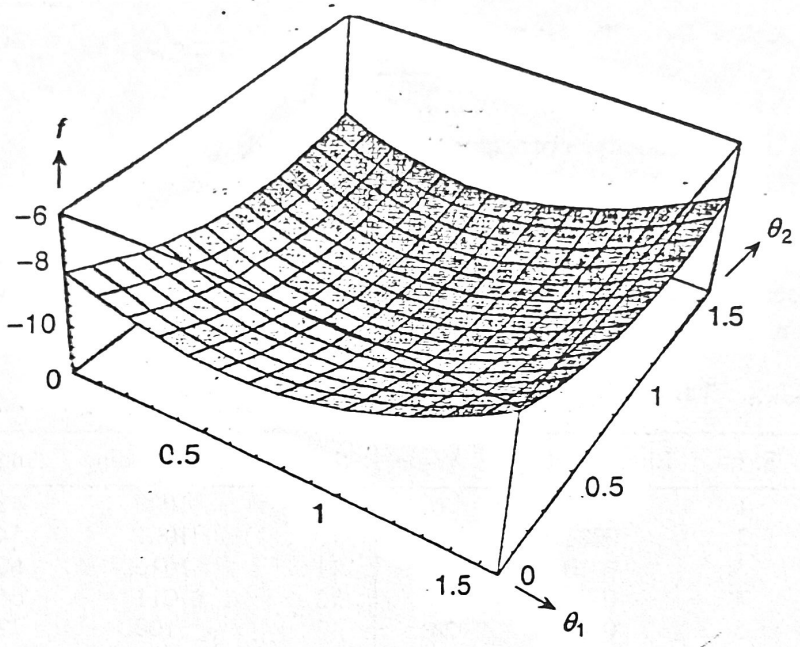


Fig. 8.13(b) Three-dimensional plot of the objective function.

Since the objective function is negative, instead of minimizing the function 'f' let us maximize $-f = f'$. The maximum value of $f = 8$ when θ_1, θ_2 are zero. Hence, the fitness function F is given as

$$F = f' - 7 = -f - 7 \tag{8.21}$$

First randomly generate eight populations with 8-bit strings as shown in Table 8.8.

Table 8.8 Computation of fitness function

Population No.	Population	Angles θ_1	Angles θ_2	$F = -f - 7$
1	0000 0000	0	0	1
2	0010 0001	12	6	2.1
3	0001 0101	6	30	3.11
4	0010 1000	12	48	4.01
5	0110 1010	36	60	4.66
6	1110 1000	84	48	1.91
7	1110 1101	84	78	1.93
8	0111 1100	42	72	4.55

As shown in Table 8.8 and Fig. 8.13(c), GA begins with a population of random strings representing design or decision variables. Thereafter, each string is evaluated to find the fitness value. The population is then operated by three main operators, namely reproduction, cross over, and mutation, to create a new population of points. The new population is further evaluated and

tested for termination. If the termination criteria are not met, the population is iteratively operated by the three operators and evaluated until the termination criteria are met. One cycle of these operations and the subsequent evaluation procedure is known as a *generation* in GA terminology.

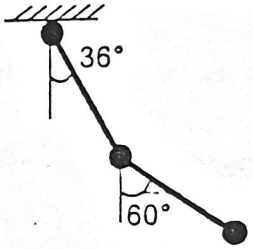
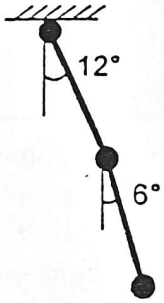
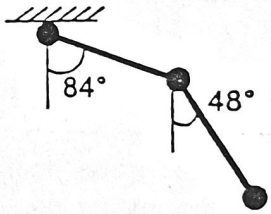
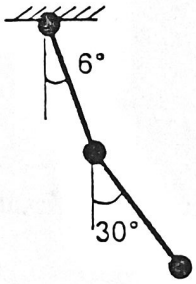
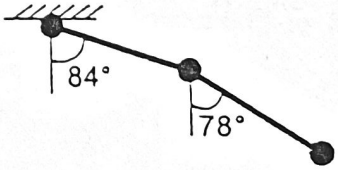
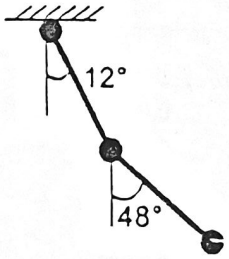
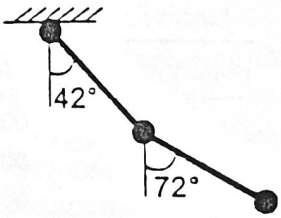
Population	*F*	Population	*F*
	1		4.60
	2.10		1.91
	3.11		1.93
	4.01		4.55

Fig. 8.13(c) '*F*' for various population.

8.7 REPRODUCTION

Reproduction is usually the first operator applied on population. Chromosomes are selected from the population to be parents to cross over and produce offspring. According to Darwin's evolution theory of survival of the fittest, the best ones should survive and create new offspring. That is why reproduction operator is sometimes known as the *selection operator*. There exists a number of reproduction operators in GA literature but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. The various methods of selecting chromosomes for parents to cross over are:

1. Roulette-wheel selection
2. Boltzmann selection
3. Tournament selection
4. Rank selection
5. Steady-state selection

8.7.1 Roulette-wheel Selection

The commonly used reproduction operator is the proportionate reproductive operator where a string is selected from the mating pool with a probability proportional to the fitness. Thus, *i*th string in the population is selected with a probability proportional to F_i where F_i is the fitness value for that string. Since the population size is usually kept fixed in a simple GA, the sum of the probabilities of each string being selected for the mating pool must be one. The probability of the *i*th selected string is

$$p_i = \frac{F_i}{\sum_{j=1}^{n} F_j} \tag{8.22}$$

where 'n' is the population size. For the example problem discussed in Example 8.1 the probability values of each string are given in Table 8.9.

Table 8.9 Probability of an individual string

Population No.	Population	$F = -f - 7$	β_i
1	0000 0000	1	0.0429
2	0010 0001	2.1	0.090
3	0001 0101	3.11	0.1336
4	0010 1000	4.01	0.1723
5	0110 1010	4.66	0.200
6	1110 1000	1.91	0.082
7	1110 1101	1.93	0.0829
8	0111 1100	4.55	0.1955
		$\bar{F} = 2.908$	

One way to implement this selection scheme is to imagine a Roulette-wheel with its circumference for each string marked proportionate to string's fitness (see Fig. 8.14). The fitness of the population is calculated as Roulette-wheel is spun 'n' times (in this example eight times), each time selecting an instance of the string chosen by the Roulette-wheel pointer. Since the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $F_i/\bar{F}$ copies of the ith string of the mating pool.

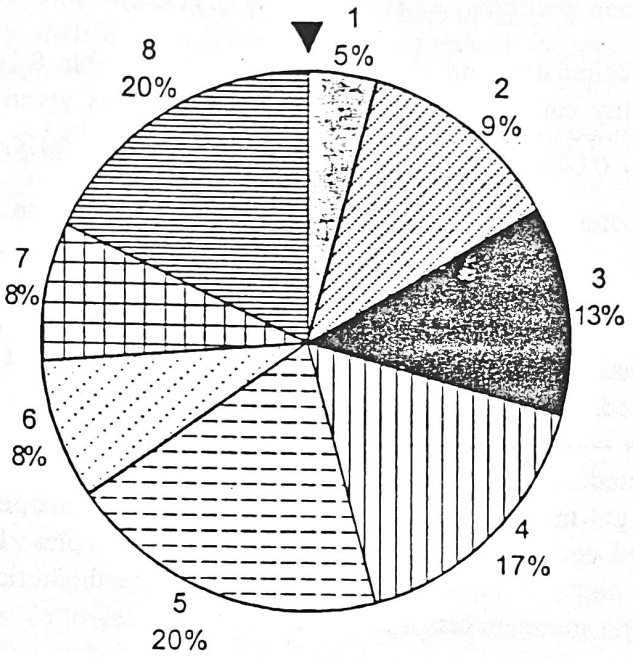


Fig. 8.14 Roulette-wheel marked for eight individuals according to fitness.

The average fitness

$$\bar{F} = \sum_{j=1}^{n} F_j / n \tag{8.23}$$

Figure 8.14 shows a Roulette-wheel for eight individuals having different fitness values. Since the fifth individual has a higher fitness than any other, it is expected that the Roulette-wheel selection will choose the fifth individual more than any other individual.

This Roulette-wheel selection scheme can be simulated easily. Using the fitness value F_i of all strings, the probability of selecting a string p_i can be calculated. Thereafter, cumulative probability P_i of each string being copied, can be calculated by adding the individual probabilities from the top of the list. Thus, the bottom most string in the population should have a cumulative probability of $P_8 = 1$. The Roulette-wheel concept can be simulated by realizing that the ith string in the population represents the cumulative probability from P_{i-1} to P_i. Thus, the first string represents the cumulative values from 0 to P_1.

Hence, cumulative probability of any string lies between 0–1. In order to choose n strings, n random numbers between zero and one are created at random. Thus, the string that represents the chosen random number in the cumulative probability range (calculated from fitness value) for the string, is copied to the matting pool. This way, the string with a higher fitness value will

represent a larger range in the cumulative probability values and therefore, has a higher probability of being copied into the mating pool. On the other hand, a string with a smaller fitness value represents a smaller range in cumulative probability values and has a smaller probability of being copied into the mating pool. Now, we illustrate the working of Roulette-wheel simulation for an example.

Referring to Table 8.10, once probability of the individual strings are known we can find the *expected count* of each string as

$$\text{Expected count} = (n = 8) \times p_i \tag{8.24}$$

These values are calculated and shown in column A of Table 8.10. From the probability p_i, the cumulative probability can be computed. For example, P_5 is given by

$$P_5 = 0.0429 + 0.090 + 0.1336 + 0.1723 + 0.2 = 0.6388 \tag{8.25}$$

These distributions are shown in column B of Table 8.10. In order to form the mating pool, we create random numbers between zero and one (given in column C) and identify the particular string which is specified by each of these random numbers. For example, if a random number of 0.428 is created, the fourth string gets a copy in the mating pool because the string occupies the interval 0.266–0.438, as shown in column B. Column D refers to the selected string. Similarly, other strings are selected according to random numbers shown in column C. After this selection procedure is repeated $n = 8$ times, where 'n' is the population size, the number of selected copies for each string is counted. This number is shown in column E. For example, the strings 4 and 5 get 2 copies, 6 and 7 get no copies, and the remaining strings get one copy each. Comparing to column A, the expected counts are that strings 5 and 8 get 2 copies, 1 and 6 get no copies, and the remaining get one copy. Column A and E reveal that the theoretical expected count and the true count of each string more or less agree with each other.

Table 8.10 Roulette-wheel selection

Population No.	Population θ_1	Population θ_2	$\beta_i = p_i$	A	B	C	D	E	Population θ_1	Population θ_2
1	0000	0000	0.0429	0.33	0.0429	0.259	3	1	0000	0000
2	0010	0001	0.090	0.72	0.1329	0.038	1	1	0010	0001
3	0001	0101	0.1336	1.064	0.266	0.486	5	1	0001	0101
4	0010	1000	0.1723	1.368	0.438	0.428	4	2	0010	1000
5	0110	1010	0.200	1.6	0.638	0.095	2	2	0010	1000
6	1110	1000	0.082	0.656	0.720	0.3	4	0	0110	1010
7	1110	1101	0.0829	0.664	0.809	0.616	5	0	0110	1010
8	0111	1100	0.1955	1.56	1.0	0.897	8	1	0111	1100

P_I = Probability
A = Expected count
B = Cumulative probability
C = Random number between 0–1
D = String number
E = The count in the mating pool

Figure 8.13(a) shows the initial random population and the mating pool after reproduction. The points marked with enclosed box are the points in the mating pool and the points marked with a

filled box show the population left out in the pool. The action of the reproduction operator is clear from this point. The inferior points have been probabilistically eliminated from further consideration. It should also be noted that not all selected points are better than rejected points. For example, first individual is selected whereas the sixth individual is not selected. Although the above Roulette-wheel selection is easier to implement, it is noisy. A better stable version of the selection operator is sometimes used. After the expected count for each individual string is calculated, the strings are first assigned value exactly equal to the mantissa of the expected count. Thereafter, the regular Roulette-wheel selection is implemented using decimal part of the expected count of the probability distribution. This selection method is less noisy and is known as *stochastic remainder selection.*

8.7.2 Boltzmann Selection

Simulated annealing is a method of functional minimization or maximization. This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. The cooling phenomenon is simulated by controlling a temperature like parameter introduced with the concept of Boltzmann probability distribution so that a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to

$$P(E) = \exp\left(-\frac{E}{kT}\right) \tag{8.26}$$

where 'k' is Boltzmann constant. This expression suggests that a system at a high temperature has almost uniform probability of being at any energy state, but at a low temperature it has a small probability of being at a high energy state. Therefore, by controlling the temperature T and assuming search process follows Boltzmann probability distribution, the convergence of the algorithm is controlled. This is beyond the scope of this book and the reader is advised to refer to the book by Deb (1995)

8.7.3 Tournament Selection

GA uses a strategy to select the individuals from population and insert them into a mating pool. Individuals from the mating pool are used to generate new offspring, which are the basis for the next generation. As the individuals in the mating pool are the ones whose genes will be inherited by the next generation, it is desirable that the mating pool consists of good individuals. A selection strategy in GA is simply a process that favours the selection of better individuals in the population for the mating pool.

There are two important issues in the evolution process of genetic search, population diversity and selective pressure, as given by Whitley (1989).

Population diversity means that the genes from the already discovered good individuals are exploited while promising the new areas of the search space continue to be explored.

Selective pressure is the degree to which the better individuals are favoured.

The higher the selective pressure the more, the better individuals are favoured. The selective pressure drives GA to improve population fitness over succeeding generations. The convergence rate of GA is largely determined by the selective pressure and population diversity. In general, higher selective pressure results in higher convergence rates. However, if the selective pressure is too high, there is an increased chance of GA prematurely converging to local optimal solution because the population diversity of the search space to be exploited is lost.

If the selective pressure is too low, the convergence rate will be slow and the GA will take unnecessarily long time to find the optimal solution because more genes are explored in the search. An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-tune GA search performance.

Whitley (1989) pointed out that the fitness proportional selection (e.g. Roulette-wheel selection) is likely to lead to two problems, namely

1. Stagnation of search because it lacks selection pressure, and
2. Premature convergence of the search because it causes the search to narrow down too quickly.

Unlike the Roulette-wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among N_U individuals (Frequency of $N_U = 2$) (Goldberg and Deb, 1991).

The best individual (the winner) from the tournament is the one with highest fitness ϕ which is the winner of N_U. Tournament competitors and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of succeeding genes. The following steps illustrate the tournament selection strategy (see Table 8.11) and the fitness values are taken from Table 8.8.

Table 8.11 Fitness values for individuals

Individuals	1	2	3	4	5	6	7	8
Fitness	1	2.10	3.11	4.01	4.66	1.91	1.93	4.55

Step 1: First select individuals 2 and 4 at random.

ϕ_2	ϕ_4
2.10	4.01

4 is the winner and hence, select the string as 0010 1000.

Step 2: Select individuals 3 and 8 at random.

ϕ_3	ϕ_8
3.11	4.55

8 is the winner and hence, select the string as 0111 1100.

Step 3: Next select 1 and 3.

ϕ_1	ϕ_3
1.0	3.11

3 is the winner and thus, select the third string as 0001 0101.

Similarly, other populations are selected from the mating pool as

Individuals	Selected
4 and 5	5
1 and 6	6
1 and 2	2
4 and 2	4
8 and 3	8

From the above, it is clear that 2, 3, 5 and 6 are chosen only once 4, 8 are chosen twice, and 1 and 7 are not chosen at all. Table 8.12 gives the new mating pool.

Table 8.12 Population for mating pool

Population no.	Population
1	0010 1000
2	0111 1100
3	0001 0101
4	0110 1010
5	1110 1000
6	0010 0001
7	0010 1000
8	0111 1100

Roulette-wheel selection omitted populations 6 and 7, two copies of 4 and 5, and single copy for the others whereas tournament selection omitted 1 and 7, two copies for 4 and 8, and single copy for the others.

During the early genetic evolution process, there are a large number of individuals or chromosomes that almost satisfy all constraints except one or two. A change in one or two design variable (strings) may produce a solution with a higher fitness value. This means throwing out these solutions may result in a loss of some important information which might eventually lead to optimal solution.

8.7.4 Rank Selection

The Roulette-wheel will have problem when the fitness values differ very much. For example, if the best chromosome fitness is 90%, its circumference occupies 90% of Roulette-wheel, then other chromosomes will have very few chances to be selected. *Rank selection* first ranks the population and taken every chromosome, receives fitness from the ranking. The worst will have fitness 1, the next 2, …, and the best will have fitness N (N is the number of chromosomes in the population). The Roulette-wheel selection is applied to the modified wheel as shown in Figs. 8.15 and 8.16.

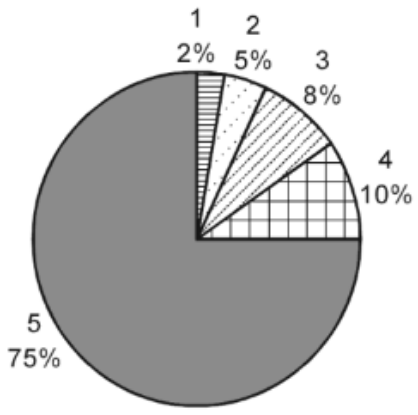


Fig. 8.15 Roulette-wheel according to fitness.

Figure 8.15 is according to fitness and Fig. 8.16 is according to rank. The method can lead to slow convergence because the best chromosome does not differ so much from the other.

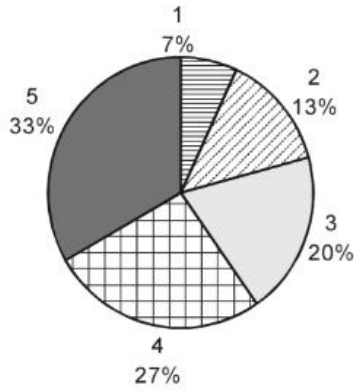


Fig. 8.16 Roulette-wheel according to rank.

8.7.5 Steady-state Selection

This is not a particular method of selecting the parents. The main idea of the selection is that bigger part of chromosome should survive to next generation. Here, GA works in the following way. In every generation are selected, a few (good individuals with high fitness for maximization problem) chromosomes, for creating new off springs. Then, some (bad with low fitness) chromosomes are removed and new offspring is placed in that place. The rest of population survives a new generation.

8.7.6 Elitism

In this method, first the best chromosome or few best chromosomes are copied to new population. The rest is done in a classical way. Elitism can very rapidly increase the performance of GA because it prevents loosing the best-found solutions. From practical consideration point of view, if F fitness functions are positive and for minimization problem, Goldberg (1989), suggest that the fitness of any ith individual must be subtracted from a large constant, so that all fitness values are non-negative and individuals get fitness values according to their actual merit.

Now, the new expression for fitness becomes

$$\phi_i = (F_{\max} - F_{\min}) - F_i(X) \tag{8.27}$$

for minimization problem.

If F_i are positive for maximization problem then $\phi_i = F_i$. For the example problem it is shown in Table. 8.13.

Table 8.13 Mating pool as per rank selection
($\bar{F} = 2.908$)

Population no.	Population	$F = \phi$	$F/\bar{F}$	Count	Mating pool
1	0000 0000	1	0.38	0	0010 0001
2	0010 0001	2.1	0.812	1	0001 0101
3	0001 0101	3.11	1.203	1	0010 1000
4	0010 1000	4.01	1.55	1	0110 1010
5	0110 1010	4.66	1.802	2	0110 1010
6	1110 1000	1.91	0.738	0	1110 1101
7	1110 1101	1.93	0.746	1	0111 1100
8	0111 1100	4.55	1.760	2	0111 1100

The reproduction operator selects fit individuals from the current population and places them in a mating pool. Highly fit individuals get more copies in the mating pool, whereas the less fit ones get fewer copies. As the number of individuals in the next generation is also same, the worst fit individuals die off. The reproduction operator can be implemented in the following manner.

The factor $\phi_i/\bar{\phi}$ for all individuals is calculated, where $\bar{\phi}$ is the average fitness. This factor is the expected count of individuals in the mating pool, and shown in column 4 of Table 8.13. It is then converted to an actual count by appropriately rounding off so that individuals get copies in the mating pool proportional to their fitness, as shown in Column 5 of Table 8.13. A mating pool is created where individuals 1 and 6 die off. This process of reproduction confirms the Darwinian principle of survival of the fittest. Figure 8.17 explains how the mating pool is created.

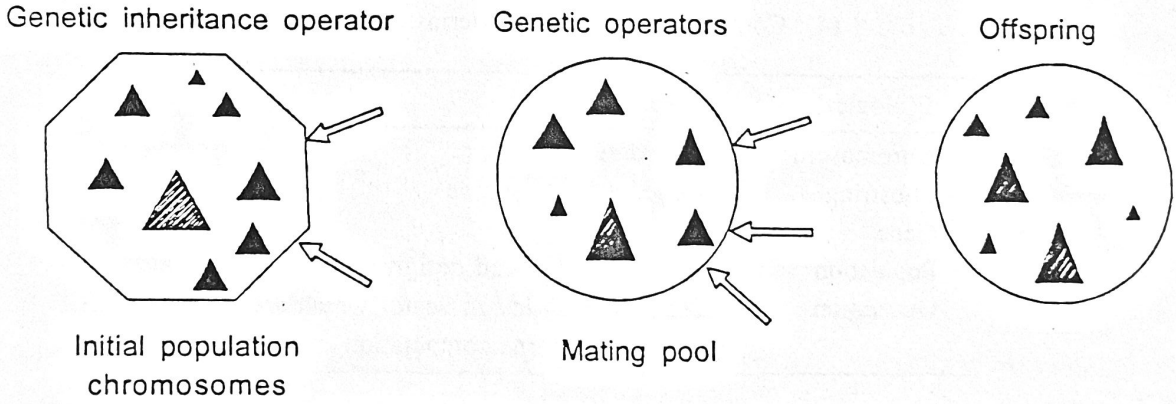


Fig. 8.17 Population for the mating pool.

8.7.7 Generation Gap and Steady-state Replacement

The *generation gap* is defined as the proportion of individuals in the population, which are replaced in each generation. So far, we have been doing reproduction with a generation gap of 1, i.e. population is replaced in each generation. However, a more recent trend has favoured steady-state replacement which is given by Whitley (1987, 1989). This operates at the other extreme and in each generation only a few (typically two) individuals are replaced. This may be a better model

of what happens in nature. In shortlived species including some insects, parents lay eggs and then die before their offsprings hatch. But in longer-lived species including mammal's, offspring and parents live concurrently. This allows parents to nurture and teach their offspring but also gives rise to competition among them. Generation gap can be classified as

$$G_p = \frac{p}{N_p} \tag{8.28}$$

where N_p is the population size and p is the number of individuals that will be replaced. Several schemes are possible. Some of which are:

1. Selection of parents according to fitness and selection of replacement at random.
2. Selection of parents at random and selection of replacement by inverse fitness.
3. Selection of both parents and replacements according to fitness/inverse fitness.

Generation gap can be gradually increased as the evolution takes place to widen exploration space and may lead to better results.

SUMMARY

In this chapter, we have seen that genetic algorithm comprises a set of individuals, elements (the populations) and a set of biologically inspired operators defining the population itself. According to evolutionary theory, only the most suited element in a population is likely to survive and generate offspring, thus transmitting the biological heredity to the new generation. In computing, GA maps problem on to a set of (typically binary) strings, each string representing a potential solution. Table 8.14 gives the comparison between biological terms and the corresponding terms in GA.

Table 8.14 Comparison of biological terms with GA terms

Biological term	GA term
Chromosome	Coded design vector
Substring	Coded design variable
Gene	Every bit
Population	A number of coded design variable
Generation	Population of design vector which are obtained after one computation

In the next chapter we will discuss inheritance operators, their performance, and the application of GA to real life problems.

- Various optimization techniques are illustrated.
- Non-traditional search and optimization methods are discussed.
- Encoding of variables in GA are given.
- Evaluation of fitness functions for an example of two bar pendulum bar is described.
- Various selection methods such as Roulette-wheel selection, Boltzmann selection, Tournament selection, Rank selection, Steady-state selection are discussed.

PROGRAMMING ASSIGNMENT

P8.1 In a three variable problem the following variable bounds are specified.

$$-6 < x < 12$$
$$0.002 \leq y \leq 0.004$$
$$10^4 \leq z \leq 10^5$$

What should be the minimum string length of any point (x, y, z) coded in binary string to achieve the following accuracy in the solution

1. two significant digits.
2. three significant digits.

P8.2 Repeat the above problem when ternary strings (with three alleles 0, 1, 2) are used instead of binary string.

P8.3 We want to use GA to solve the following nonlinear programming problem.

$$\text{minimize } (x_1 - 2.5)^2 + (x_2 - 5)^2$$

subject to

$$5.5x_1 + 2x_2^2 - 18 \leq 0$$
$$0 \leq x_1, x_2 \leq 5$$

We decide to give three and two decimal places of accuracy to variables x_1, x_2 respectively.

1. How many bits are required for coding the variables?
2. Write down the fitness function which you would be using in reproduction.

P8.4 Consider the following population of binary strings for a maximization problem.

String	Fitness
01101	5
11000	2
10110	1
00111	10
10101	3
00010	100

Find out the expected number of copies of the best string in the above population of the mating pool under

1. Roulette wheel selection.
2. Tournament selection.

If only the reproduction operator is used, how many generations are required before the best individual occupies the complete population under each selection operator.

P8.5 Write a program in "C" or in FORTRAN for creating initial population (for n variables) with n-bits string for each variable. The values of each variable can be selected from a table of data. Assume an objective function, find fitness value, and get the offspring using Roulette-wheel selection.

REFERENCES

Davis, L. (1991), *Handbook of Genetic Algorithms*, New York, Van Nostrand, Reinhold.

Deb, K. (1991), Optimal Design of a Welded Beam Structure via Genetic Algorithms, *AIAA Journal*, Vol. 29, No. 11, pp. 2013–2015.

Deb, K. (1995), Optimization for Engineering Design—Algorithms and Examples, Prentice-Hall of India, New Delhi.

Dorigo, M. and G.D. Caro (1999), Ant Algorithm for Discrete Optimization, *Artificial Life*, Vol. 5, pp. 137–172.

Goldberg, D.E. and M.P. Samatini (1986), Engineering Optimization via Genetic Algorithm, *Proc. of the Ninth Conference on Electronic Computation*, pp. 471–482.

Goldberg, D.E. (1989), Genetic Algorithms in Search, Optimization and Machine Learning, Reading, Mass., Addison-Wesley.

Goldberg, D.E. and K. Deb (1991), A Comparative Analysis of Selection Schemes Used in GA, *Foundations of Genetic Algorithms*, I. pp. 53–69.

Holland, J.H. (1975), Adaptation of Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

Jenkins, W.M. (1991), Towards Structural Optimization via the Genetic Algorithms, *Computers and Structures*, Vol. 40, No. 5, pp. 1321–1327.

Kirkpatrik, S., C.D. Gelatt, and M.P. Vecchi (1983), Optimization by Simulated Annealing, *Science* 220, pp. 671–680.

Kost, B. and Baumann (1999), Structural Topology Optimization by Stochastic Algorithms, *In Computer Aided Optimum Design of Structures*, WIT Press. Eds., S. Hernandez, A.J. Kassab and C.A. Brebbia, pp. 13–22.

Kost, B. (1995), Evolution Strategies in Structural Topology Optimization of Trusses, Eds. P.J. Pahl and H. Werner, *Proc of 6th Int. Conf. on Computer in Civil and Building Engg*, A.A. Balkimy, Rotter Dama, pp. 675–681.

Michalewicz, Z. (1992), *Genetic Algorithm + Data Structures = Evolution Program*, Berlin: Springer Verlag.

Rajeev, S. and G.S. Krishnamoorthy (1992), Discrete Optimization of Structures Using Genetic Algorithms, *Jol. of Structural Engineering*, ASCE, Vol. 118, No. 5, pp. 1223–1250.

Whitley, D. (1987), Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery, J.J. Greffenstette (Ed.), *Proc. of Second Intl. Conf. on Genetic Algorithms*, Lawrence Eralbaum Associates, pp. 108–115.

Whitley, D. (1989), The GENITUR Algorithm and Selection Pressure—Why Rank-based Allocation of Reproduction Trials is Best, *Proc. of Int. Conf. on Genetic Algorithms, Schaffered,*. Morgan Kaufmann Publishers, Los Altos, Cal, pp. 10–19.

Wolfram, S. (1994), *Cellular Automata and Complexity*, First ed., Addison-Wesley,