Chapter 4

GENETIC ALGORITHM – AN OVERVIEW

4.1 GA VERSUS TRADITIONAL METHODS

Genetic algorithms (GAs) are so far generally the best and most robust kind of evolutionary algorithms, which are an alternative to traditional optimization methods. In GAs, a design variable is not generally presented by its actual design value, instead a sufficiently long arrays of bits (generally binary) is used to code it. A coded design variable is called a substring in GA terminology. This simple encoding strategy allows the implementation of crossover, mutation and other operations. Although a substantial amount of research has been performed on variable length strings and other structures, the majority of work with GAs is focused on fixed-length character strings.

The following list is a quick look at the essential differences between GAs and other forms of optimization.

- **GAs work with the coded form of the function values (parameter set), rather than with the actual values themselves.** For example, if one wants to find the minimum of the function $f(x) = x^3 + x^2 + 5$, the GA would not deal directly with, x or y values, but with strings that encode these values. For this case, binary strings should be used.
- **The trademark of a GA is that it is heuristic, which means it estimates a solution.** One won't know if one gets the exact solution, but that may be a minor concern. In fact, most real-life problems are like that: One estimate a solution rather than calculating it exactly.
- **GA use a set, or population, of points to conduct a search, not just a single point on the problem space.** This gives GAs the power to search noisy spaces littered with local optimum points. Instead of relying on a single point to search through the space, the GAs look at many different areas of the problem space at once, and use all of this information to guide it.
- **GAs use only payoff information to guide themselves through the problem space.** Many search techniques need a variety of information to guide the search process. Hill climbing methods require derivatives, for example. The only information a GA needs is some measure of fitness of a point in the space. Once the GA knows the current

measure of "goodness" about a point, it can use this to continue searching for the optimum.

- **GAs are probabilistic in nature, not deterministic.** This is a direct result of the randomization techniques used by GAs.
- **GAs are inherently parallel.** GAs, by their nature, are parallel, dealing with a large number of points (strings) simultaneously. A GA processing n strings at each generation, in reality processes n^3 useful substrings.

Even though GAs are different from the most traditional search algorithms there are some similarities between them. In traditional search methods a search direction is used to find a new point. In the gradient based methods, for example, the search direction requires derivative information which is usually calculated using function values at two neighbouring points. Similarly, in GAs the crossover operator requires two parent strings to create two new child strings. Thus, the crossover operation is similar to a directional search method except that the search direction is not fixed for all points in the population and that no effort is made to find the optimal point in any particular direction. The reproduction operator has an indirect effect of filtering the good search directions and help guide the search. The mutation operator tries to create a point in the vicinity of the current point. Thus, search in the mutation operator is similar to the local search method.

4.2 Natural Genetics and GA

All living organisms consist of cells. In each cell there is the same set of chromosomes which are strings of DNA and serve as a model for the whole organism. A chromosome consists of genes, blocks of DNA. Each gene has its own position in the chromosome. This position is called locus. Complete set of genetic material (all chromosomes) is called genome. Particular set of genes in genome is called genotype. The genotype is, with later development after birth, base for the organism's phenotype, its physical and mental characteristics, such as eye color, intelligence etc.

During reproduction, recombination (or crossover) first occurs. Genes from parents combine to form a whole new chromosome. The newly created offspring can then be mutated. Mutation means that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents. The fitness of an organism is measured by success of the organism in its life (survival).

The strings of artificial genetic systems are analogous to the chromosomes of the natural genetic systems. In the natural system, one or more number of chromosomes combines to form a total genetic prescription for the existence and operation of an organism. This total genetic package is called the genotype in the natural systems. Whereas, in case of artificial systems this total package is called a structure. In natural system, the organism formed by the interaction of the total genetic package with its environment is called the phenotype. In artificial genetic systems the structures decode to form a particular parameter set, solution alternative, or point (in solution space).

In natural genetics, the chromosomes are composed of genes, which may take on some number of values called alleles. The position of gene (its locus) is identified separately from the gene's function. In artificial genetics, one can consider that strings are composed of features or detectors that can take on different values. Features may be located at different positions on the string. Table 4.1 summarizes the main analogies between natural and artificial genetics.

Table 4.1 Analogies Between Natural and Artificial Genetics

NATURAL GENETICS	GENETIC ALGORITHMS
Chromosome	String
Gene	Feature, Character or Detector
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set, alternative solution, a decoded structure
Epistasis	Nonlinearity

4.3 ELEMENTS OF GA

4.3.1 Coding and Decoding of Variables

Binary Encoding: Binary coded strings having 1s and 0s are mostly used for structural optimization. The length of string is usually determined according to the desired accuracy of the solution. For example, if four bits are used to code each variable in two variable function optimization problem, the strings (0000 0000) and (1111 1111) would represent the points $(X_1(L), X_2(L))^T$ and $(X_1(U), X_2(U))^T$ respectively, because the substrings (0000) and (1111) have the minimum and the maximum decoded values. Any other eight-bit string can be found to represent a point in the search space according to a fixed mapping rule. Usually the following linear mapping rule is used:

$$X_i = X_i(L) + \frac{X_i(U) - X_i(L)}{2^{l_i} - 1} \cdot \text{decoded value}(X_i) \qquad \dots (4.1)$$

The variable X_i is coded in a substring S_i of length l_i. The decoded value of a binary substring S_i is calculated as $\sum_{k=0}^{l-1} 2^k s_{i(l-1-k)}$. For example, a four bit string (0111) has a decoded value equal to $[(1).2^0 + (1).2^1 + (1).2^2 + (0).2^3]$ i.e. 7. With four bits to code each variable there are only 2^4 or 16 distinct substrings possible, because each bit position can take a value either 0 or 1. The accuracy that can be obtained with a four-bit coding is only approximately 1/16th of the search space. It is not necessary to code all variables in equal substring length. The length of substring representing a variable depends upon the desired accuracy of that variable. Thus, one can say that with l_i bit coding for a variable, the accuracy obtained in that variable is approximately [Xi (U) - Xi (L)] / $2.l_i$. After coding of variables has been done the corresponding point X = $(X1, X2, \dots\dots, Xn)^T$ can be found. Thereafter, the function value at that point X can be calculated by substituting X in the given objective function f(X).

Permutation Encoding: In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

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

Permutation encoding is useful for ordering problems. For some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have real sequence in it) for some problems.

Value Encoding: Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult. In the value encoding, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, characters or any objects. Value encoding is a good choice for some special problems. However, for this encoding it is often necessary to develop some new crossover and mutation specific for the problem.

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

Tree Encoding: Tree encoding is used mainly for evolving programs or expressions, i.e. for genetic programming. In the tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language. Programming language LISP is often used for this purpose, since programs in LISP are represented directly in the form of tree and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

4.3.2 Fitness Calculation

GAs mimic the survival of the fittest principle of the nature to make the search process. Thus, GA's are naturally suitable for solving maximization problems. The fitness is the measure of ability of and individual for survival. In general, a fitness function F(x) is first derived from the objective function and used in successive genetic operations. Some genetic operators require the fitness function to be non-negative. For maximization problems the fitness function can be considered the same as the objective function i.e. $F(x) = f(x)$. For minimization problems the fitness function transforms it in to an equivalent maximization problem such that the optimum point remains unchanged. The following transformation is normally used:

$$F(x) = 1 / (1 + f(x)) \qquad \ldots (4.2)$$

4.3.3 Fitness Scaling

It is very important to regulate the number of copies in a small population genetic algorithm. During the early stages of GA runs it is common to have a few extraordinary individuals in a population of mediocre colleagues. If normal selection rule is used, the extraordinary individuals would take over a significant proportion of finite population in a single generation, which may lead to premature convergence. In the later stages of GA runs, although there may be significant diversity within the population, the population average fitness may be close to the population best fitness. This will lead to a situation wherein the average members and the best members gets the same number of copies in the future generations, and the survival of fittest, necessary for improvement in solution, becomes a random walk among the mediocre. Scaling helps prevent early domination of extraordinary individuals, while later on encourages a healthy competition among near equals. Thus, in both the cases, i.e. at beginning of the run and as the run progresses fitness scaling can be of immense help.

❖ Linear Scaling Method

Let the raw fitness be f and the scaled fitness be f'. Linear scaling (Fig. 4.1) requires a relationship between f' and f as follows:

$$f' = a \cdot f + b \qquad \dots (4.3)$$

The coefficients a and b may be chosen in a number of ways; however in all the cases one requires that average scaled fitness f'_{avg} should be equal to the average raw fitness f_{avg} because the selection procedure will ensure that each average population member contributes one expected offspring to the next generation. To control the number of offspring given to the

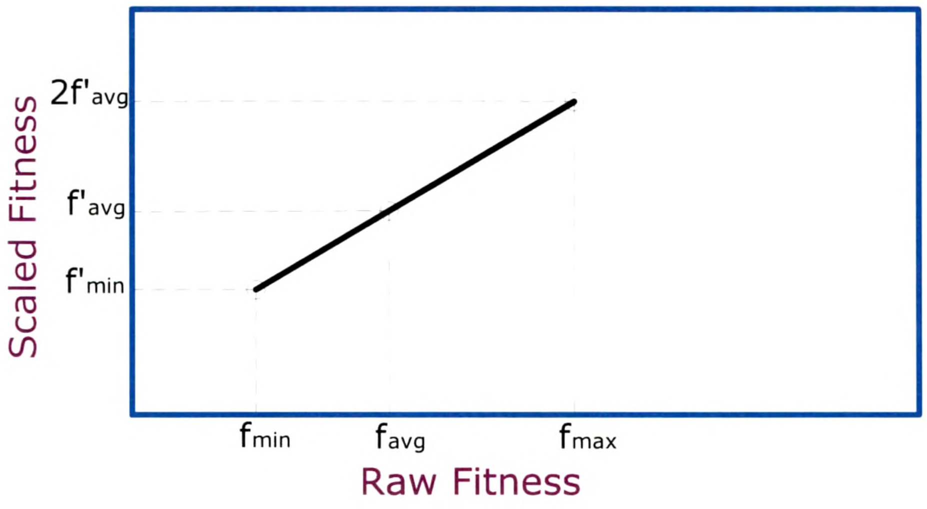


Fig. 4.1 Linear Scaling Under Normal Conditions

population member with maximum raw fitness, another scaling relationship is chosen to obtain a scaled maximum fitness,

$$F'_{max} = C_{mult} \cdot f_{avg} \qquad \dots (4.4)$$

where, C_{mult} is the number of expected copies desired for the best population member. For small populations (n=50 to 100) a C_{mult} =1.2 to 2 has been used.

Toward the end stages of the run, choice of C_{mult} stretches the raw fitness significantly. This may cause difficulty in applying the linear scaling rule. At first there is no problem applying the linear scaling rule, because few extraordinary individuals get scaled down and the low members of the population get scaled up. The more difficult situation is shown if Fig. 4.2.

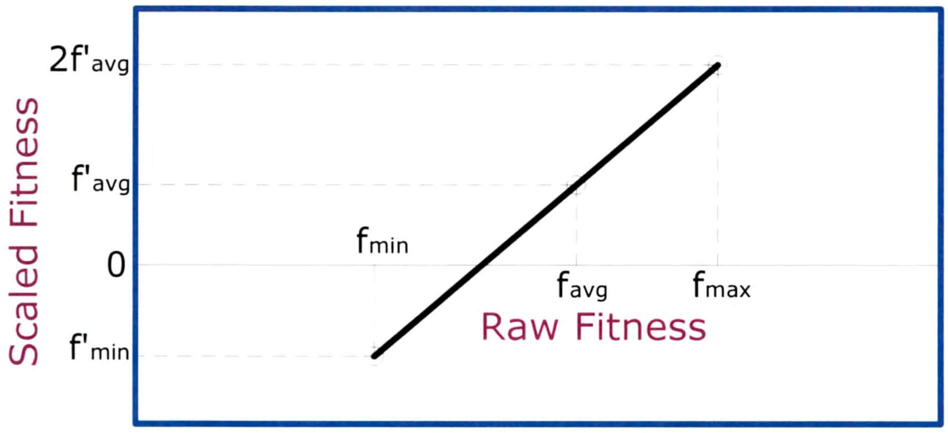


Fig. 4.2 Problems with Linear Scaling in Mature Run

This type of situation arises in a mature run when few bad strings are far below the population average and maximum, which are relatively close together. If the scaling rule is applied in this situation, the stretching required on the relatively close average and maximum raw fitness causes the low fitness values to go negative after scaling. A number of solutions are available to solve this problem. Here, when one cannot scale to the desired C_{mult}, one can still maintain equality of the raw and scaled fitness averages and one can map the minimum raw fitness f min to a scaled fitness f'min = 0

4.4 GENETIC OPERATORS

There are mainly three operators used in genetics. They are Reproduction, Crossover and Mutation. The reproduction operator selects good strings and the crossover operator recombines good strings together to hopefully create a better substring. The mutation operator alters a string locally to hopefully create a better substring.

4.4.1 Reproduction

Reproduction is the first operator applied to the population. The reproduction operator is sometimes also known as the selection operator. The basic idea in any reproduction operator is that above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. A string is selected for the matting pool with a probability proportional to its fitness. If H(x) is the objective function then the H(x) is converted in to a corresponding fitness values and is done in such a way that the best individual has maximum fitness. According to Goldberg [7], for minimization problems H(x) should be subtracted from a large constant so that all the fitness values are obtained according to the actual merit. The large constant is obtained by adding the maximum and minimum value of the H(x), the expression for the fitness becomes:

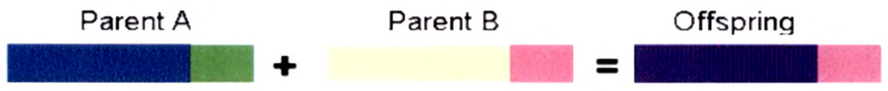
$$F_i = [H(x)_{max} + H(x)_{min}] - H_i(x) \qquad \ldots (4.5)$$

where, F_i is the fitness of the i_{th} population. The factor F_i/F_{avg} for all the individuals is calculated where F_{avg} is the average fitness. The factor is the excepted count of the individuals in the mating pool and is then converted in to the actual count by approximately rounding off so that individuals get copies in the mating pool according to their fitness. As the number of individuals (populations) in the next generation is the same the worst fit individuals die off. The next generation these best populations are mated randomly and crossed at random lengths of the full string.

4.4.2 Crossover

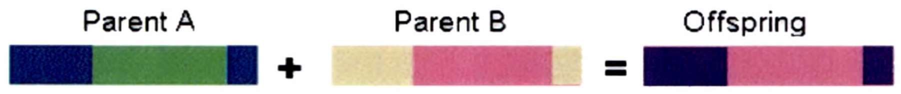
Crossover operator is applied to the mating pool with a hope that it would create a better string. In this, pair of strings is selected from the mating pool at random and some portion of the strings is exchanged between the strings. The two strings participating in the crossover operation are known as parent strings and the resulting strings are known as child strings or offspring. The populations obtained after crossing will form new population set for the next generation. The following are few types of crossover operators:

(i) Single point crossover - One crossover point is randomly selected, binary string from the beginning of the chromosome to the crossover point is copied from the first parent and the rest is copied from the other parent.



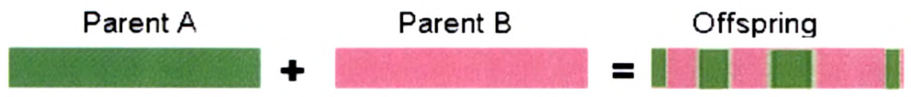
$$\begin{array}{r|l} \mathbf{11001} & 011 \\ + & \\ 11011 & \mathbf{111} \end{array} \quad = \mathbf{11001111}$$

(ii) Two point crossover - Although one-point crossover is inspired by biological processes, its algorithmic counterpart has some drawbacks. One of the most important is that one-point crossover cannot combine certain combinations of features encoded on chromosomes: schemata with a large defining length are easily disrupted. It is also possible that certain elements are not allowed to appear more than once. Therefore, over the past few years, several other crossover techniques have been used. Some of them are: Two point crossover and uniform crossover. In two point crossover two crossover points are selected and binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again.



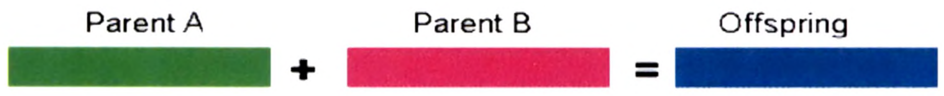
11|0010|**11**
+
11 |**0111**|11
= **11011111**

(iii) Uniform crossover - Bits are randomly copied from the first or from the second parent.



1|**10**|010|**11**
+
1|10|**111**|01
= **11011111**

(iv) Arithmetic crossover - Some arithmetic operation is performed to make a new offspring.



11001011 + 11011111 = **11001001**

The effect of crossover may be detrimental or beneficial. Thus in order to preserve some of the good strings that are already present in the mating pool not all strings in the mating pool are used in crossover. When a crossover probability of P_c is used only $100.P_c$ percent strings in the population are used in the crossover operation and the $100(1-P_c)$ percent of the population remains as they are in the current population.

4.4.3 Mutation

Selection and crossover alone can obviously generate a staggering amount of differing strings. However, depending on the initial population chosen, there may not be enough variety of strings to ensure the GA sees the entire problem space. Or the GA may find itself converging on strings that are not quite close to the optimum it seeks due to a bad initial population. These problems are overcome by introducing a mutation operator into the GA. The performance of the operator depends on mutation probability P*m*, which dictates the frequency at which mutation occurs. Mutation can be performed either during selection or crossover. For each string element in a string in the mating pool, the GA checks to see if it should perform a mutation. If it should, it randomly changes the element value to a new one. In binary strings, 1s are changed to 0s and 0s to 1s. For example, if the GA decides to mutate bit position 4 in the string 10000, the resulting string is 10010 as the fourth bit in the string is flipped. The mutation probability should be kept very low (usually about 0.001%) as a high mutation rate will destroy fit strings and degenerate the GA algorithm into a random walk, with all the associated problems. But mutation will help prevent the population from stagnating. Much of the power of a GA comes from the fact that it contains a rich set of strings of great diversity. Mutation helps to maintain that diversity throughout the GA's iterations. Thus, mutation creates a point in the neighbourhood of the current point, thereby achieving a local search around the current solution.

4.5 The Chromosome Selection Schemes

The chromosomes are selected from the generated population to be parents for crossover. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offsprings. There are many methods in selecting the best chromosomes. Some of them are discussed below.

4.5.1 The Roulette Wheel Selection Scheme

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel (RW) where all the chromosomes in the population are placed. The size of the section in the roulete wheel is proportional to the value of the fitness function of every chromosome (Fig. 4.3). This can be done by marking the circumference of the wheel in proportion with the fitness value of each chromosome - the bigger the value is, the larger is the section. The roulette wheel is spun n times, each time selecting a chromosome chosen by the roulette wheel pointer. Clearly, the chromosomes with bigger fitness value will be selected more times.

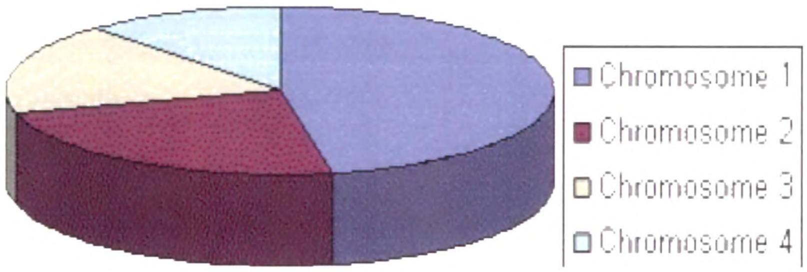


Fig. 4.3 RW Marked with Fitness of Four Individuals

This process can be described by the following algorithm:

1. [Sum] Calculate the sum of all chromosome fitnesses in population - sum *S*.
2. [Select] Generate random number from the interval *(0,S)* - *r*.
3. [Loop] Go through the population and sum the fitnesses from *0* - sum *s*.
4. When the sum *s* is greater then *r*, stop and return the chromosome.

The step 1 is performed only once for each population.

4.5.2 Rank Selection Scheme

The previous type of selection will have problems when there are big differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses then the other chromosomes will have very few chances to be selected. Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population). However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

4.5.3 Steady-State Selection Scheme

The main idea of this type of selecting to the new population is that a big part of chromosomes can survive to next generation. The steady-state selection GA works in the following way. In every generation a few good (with higher fitness) chromosomes are selected for creating new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

4.5.4 Stochastic Remainder Selection Method

The basic idea of this selection is to remove or copy the string depending on the value of the reproduction counts. The reproduction count is computed based on the fitness value by stochastic remainder method. First the probability of selection Ps is calculated as $Ps = f(i) / \Sigma f(i)$ where f(i) is the fitness value of the individual and the sigma f(i) is the sum of the fitness of the population. The expected number of the individuals of each string is calculated as $e_i = Ps \times p$, where p is population. The fractional parts of e_i are treated as probabilities with which individuals are selected for reproduction. Individuals with 0 counts are eliminated from the population.

4.5.5 The Tournament Selection Scheme

This is a well known selection scheme where an individual (i) is selected randomly and an individual (ii) is also randomly selected and the best of (i) and (ii) goes forward.

4.5.6 The Elitist Selection Scheme

One copy of best member in a population passes unchanged to the next population to ensure that any optimized value is no worse than the best previously attained. And the rest of the new population is filled by the traditional selection, crossover and mutation.

4.6 GAs in Constrained Optimization Problems

Most of the practical problems in engineering require one or more constraints to be satisfied. For example, in case of optimization of truss problems, the main constraints are stresses in the members, displacement of the nodes, slenderness ratio, etc. Thus while using GA for such problems the violation of such constraints must be suitably handled.

Constraints are usually classified as equality or inequality relations. Since equality constraints can be easily subsumed into a system, the only constraint with which one is really concerned is the inequality constraints. GA generates a sequence of parameters to be tested using the system model, objective function, and the constraints. If constraints are violated, the solution is infeasible and thus has no fitness. If not, the parameter set is assigned the fitness value corresponding to the objective function evaluation. This procedure is fine except that many practical problems are highly constrained; finding a feasible point is almost difficult as finding the best. As a result, one usually wants to get some information out of infeasible solutions also. Penalty Method is usually applied for this which degrades fitness ranking in proportion to the degree of constraint violation.

In the penalty method, a constrained problem in optimization is transformed in to an unconstrained problem by penalizing the objective function for violation of any constraints. The value of the penalty applied is related to the degree of constraints violation. The resulting penalized objective function quantitatively represents the extent of the violation of the constraints and provides a relatively meaningful measurement of the performance of each solution string. Several penalty function schemes are available, but selection of a scheme depends upon the type of problem to be solved. Two well known penalty function schemes proposed for structural optimization are as follows:

4.6.1 Multiple Segment Penalty Function

Multiple linear segment penalty function is one of the simplest penalty function schemes. Consider a problem where displacement and stress constraints are imposed. Each structural element is checked for stress violation and each model node is checked for displacement violation. If no violation is found, then no penalty is imposed on the objective function. If a constraint is violated, then the penalty is defined as,

$$\varphi_i = \begin{cases} 1 & \text{if } |p_i| / p_{max} \leq 1 \\ k_1 \, |p_i| / p_{max} & \text{if } |p_i| / p_{max} > 1 \end{cases} \qquad \dots (4.6)$$

where φ_i is the penalty value for constraint i; p_i = structural parameter or response (deflection, stress, etc.); p_{max} = maximum allowable value of each p_i; and k_1= penalty rate.

The function defined above can be expanded. Consider a triple segment penalty function defined as

$$\varphi_i = \begin{cases} 1 & \text{if } |p_i| / p_{max} \leq 1 \\ k_1 |p_i| / p_{max} & \text{if } 1 < |p_i| / p_{max} \leq c_1 \\ k_2 |p_i| / p_{max} & \text{if } c_1 < |p_i| / p_{max} \end{cases} \qquad \dots (4.7)$$

where, k_1 and k_2 = penalty rates and c_1 = limiting percentage for minor constraint violations. The main advantage of triple segment penalty function is that solutions that slightly violate the constraints may be penalized at a different rate than solutions with severe violations.

4.6.2 Quadratic Penalty Function

The quadratic penalty funtion is defined as,

$$\varphi_i = 1 + k_3\,(q_i - 1)^2 \qquad \dots (4.8)$$

where k3 = quadratic penalty rate; and q_i is defined as

$$q_i = \begin{cases} 1 & \text{if } |pi| / p_{max} \leq 1 \\ |pi| / p_{max} & \text{if } |pi| / p_{max} > 1 \end{cases} \qquad \dots (4.9)$$

Having obtained penalty function factors, the fitness value of a particular solution string is obtained by multiplying the objective function by the corresponding penalty factors:

$$O_p = W\prod_{i=1}^{m}\varphi_i\,; \qquad W = \sum_{j=1}^{n}\rho L_j A_j \qquad \dots (4.10)$$

where O_P = penalized objective function, m = total number of points where the constraints are checked, W = weight of the entire structure, n is total number of members, ρ is the material density, L_j and A_j are length and area if ith member and the product represents the total penalty.

4.7 THE WORKING OF GA IN A NUT SHELL

The problem of GA has a general form,

Maximize: f(x),

Subject to: $l_i \leq x_i \leq u_i$, i = 1 to n

In the most commonly used GAs each variable is represented as a binary number say of m bits. This is carried out by dividing the feasible interval of variable X_i, into 2m-1 intervals. For m = 6 the number of intervals will be N = 2m-1 = 63 Then each variable X_i can be represented by any of the discrete representations. For example, the binary number 110110 can be decoded as,

$$X_i = X_l + S_i\,(1.2^5+1.2^4+\overset{0}{\cancel{1}}.2^3+1.2^2+1.2^1+\overset{0}{\cancel{1}}.2^0) = X_l + 54\,S_i \qquad \ldots (4.11)$$ 4

where, S_i is the interval step for the variable X_i, given by,

$$S_i = (U_i - L_i)/(2^m-1) \qquad \ldots (4.12)$$

Step 1: Creation of the initial population

The first step in the development of GA is the creation of initial population. Each member of the population is a string of size n*m bits. The variables in the binary form are attached end to end as follows:

Member 1	101101	101001	001010	...	101111
	X1	X2	X3		Xn
Member 2	101001	100001	001110	...	101001
	X1	X2	X3		Xn
Member Z	101001	101101	101010	...	101100
	X1	X2	X3		Xn

Step 2: Evaluation

In the evaluation phase, the values of the variables for each member are extracted. Sets of six binary digits are read, and decoding is done. The function values f_1, f_2, ...f_z are evaluated. The function values are referred to as 'fitness values' in GA language. The average fitness f is calculated. The reproduction takes the steps of creation of a mating pool, crossover operation, and mutation.

Step 3: Creation of a mating pool

The reproduction phase involves the creation of a mating pool. Here the weaker members are replaced by stronger ones based on the fitness values. One makes use of a simple chance selection process by simulating a roulette wheel. The first step in this process is to have all the function values as positive entities. Some convenient scaling scheme may be used for this. Linear scaling scheme has been found satisfactory in most of the optimization problems solved by many research workers. After performing the scaling operations the sum of the scaled fitness values S is calculated using

$$S = \sum f'_i \qquad \ldots (4.13)$$

Roulette wheel selection is now used to make copies of the fittest members for reproduction. One now creates a mating pool of Z members as follows. The roulette wheel is turned Z times (equal to number of members in the population). A random number r in the range of 0 to 1 is generated at each turn. Let j be the index such that

$$r <= f'_1 + f'_2 + \ldots.. + f'_J \qquad \ldots (4.14)$$

The j_{th} member is copied into the mating pool. By this simulated roulette wheel selection, the chance of a member being selected for the mating pool is proportional to its scaled fitness value.

Step 4: Crossover Operation

The crossover operation is now performed on the mating parent pool to produce off-spring. A simple crossover operation involves choosing two members and a random integer k in the range of 1 to n.m-1 (6n-1 for six bit choice). The first k positions of the parents are exchanged to produce the two children. An example of this one point crossover for six bits of three variables with crossover site k = 8 is shown below.

Parent 1	**10110110|1001001010**
Parent 2	**11010001|0110111000**
Child 1	**110100011001001010**
Child 2	**101101100110111000**

Step 5: Mutation

Mutation is a random operation performed to change the expected fitness. Every bit of a member of the population is visited. With a bit permutation probability m_p of 0.005 to 0.1, a random number r is generated in the range of 0 to 1. If $r < m_p$, a bit is changed to 0 if it is 1 and 1 if it is a 0.

Step 6: Evaluation

The population is evaluated using Step 2. The highest fitness value and the corresponding variable values are stored in f_{max} and X_{max} respectively and a generation is now completed. If the preset number of generations is complete, the process is stopped. This X_{max} is the required near optimal solution to the problem at hand. Figure 4.4 shows the flowchart of GA for constrained optimization problem.

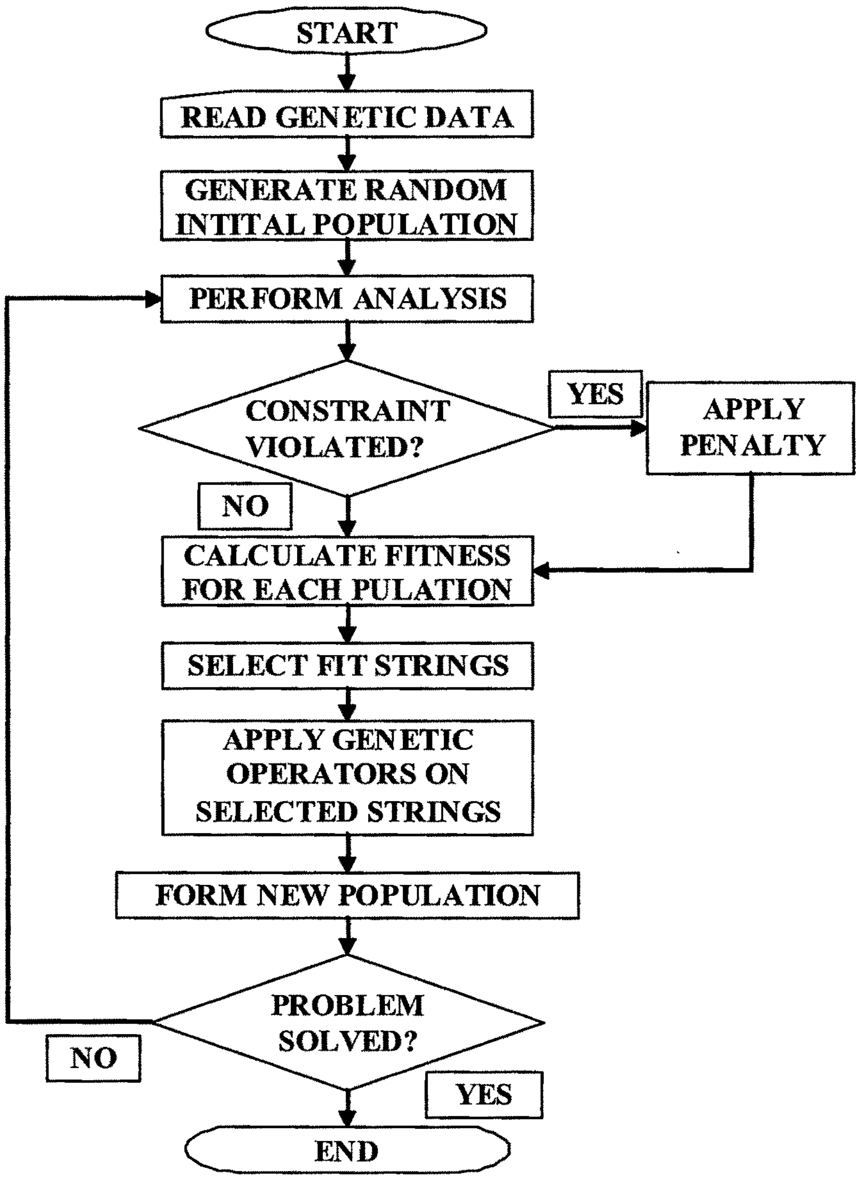


Fig 4.4 Flowchart of GA based Optimization Procedure

4.8 Some Advanced Techniques in GA

4.8.1 Schema Theorem

Although the simple GA is a very powerful tool on its own, there are some ways to improve the technique, resulting in a faster convergence. Before these 'advanced techniques' are discussed, it is important to know the mathematical foundation of the GA.

GAs do not operate over the space of solutions, but over the space of schemata. A schema is a similarity template describing a subset of strings with similarities at certain string positions. A schema over the binary alphabet without loss of generality is a string of the type $(a_1, a_2, ..., a_i, ..., a_l)$, $a_i \in \{0, 1, *\}$.

The '*' symbol is a 'don't care' symbol which accepts both '1' and '0'. The more '*' symbols a schema contains the less specific it becomes, i.e. the more strings it describes. With an alphabet of 3 symbols a schema having *m* '*' symbols will describe 2^m strings. For example, a schema (01*0*1) describes the set

{(010001), (010011), (011001), (011011)}.

The idea of a schema results in a powerful and compact way to talk about well-defined similarities among finite length strings over a finite alphabet. Usually, by examining the fitness of any one string, one might expect to obtain information about other strings which have a structure similar to it. The total number of different schemata resulting from a string of length l and an alphabet of cardinality (number of alphabet characters) k, is $(k+1)^l$. The size of the actual search space is k^l, since the '*' symbols are not involved in the actual strings GA process; they only serve as a tool to explain the chromosome structure and to describe string similarity templates in a compact format. The space of schemata is thus bigger than the space of solutions. In general, any particular binary string belongs to exactly 2^l of varying degrees of specificity, so each string is able to contain 2^l possible schemata. This means that, in testing a string, one derives a great deal of information regarding the fitnesses of schemata it belongs to. Holland calls this implicit parallelism and this observation as a major part of the explanation of the power of the GA search: One can regard GAs as a competition between schemata. At the same time 2^l competitions are being held.

Holland proposed the 'fundamental theorem of Genetic Algorithms' which proves that short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations. Schemata having these features are called building blocks. When one defines the *order* of a schema as the number of fixed positions (for example, the schema 1**0*0 has an order of 3) and the defining length of a schema as the distance between the outermost fixed positions (for example, the schema 1**0** has defining length of 4 - 1 = 3), the following equation holds:

$$m(H,i+1) = m(H,i) * \frac{f(H)}{f_{avg}}\left[1 - p_c \frac{\delta(H)}{1-1} - o(H)p_m\right] \qquad \ldots (4.15)$$

where m(H, i + 1) = the expected number of instances of schema H in generation i + 1, f(H) = the average fitness value of schema H, f_{avg} = the average fitness value of the population, p_c = the crossover probability, p_m = the mutation probability, $\delta(H)$ = the defining length of schema H and o(H) is the order of schema H.

One of a GA's most important qualities is its ability to evaluate many possible solutions simultaneously. This ability, called implicit parallelism, is the cornerstone of the GA's power. Implicit parallelism results from simultaneous evaluation of the numerous building blocks that comprise the string. Each string may contain millions of these building blocks, and the G A assesses them all simultaneously each time it calculates the string's fitness. In effect, the algorithm selects for patterns inside the string that exhibit high worth and passes these building blocks on to the next generation. This selection process enables GAs to perform well where traditional algorithms flounder, such as problems with huge search spaces.

4.8.2 Fitness Techniques

4.8.2.1 Windowing

Windowing is a technique for assigning fitnesses to a population of chromosomes to boost the fitnesses of the weaker members, in order to prevent their elimination and the resulting dominance by a small number of chromosomes. It works in the following manner: **"Find the minimum evaluation in the population. Assign each chromosome a fitness equal to the amount that it exceeds this minimum. Optionally, a minimum amount greater than the minimum value may be created as a guard against the lowest chromosomes having no chance of reproduction."**

4.8.2.2 Linear normalization

Linear normalization takes the fairness inherent in windowing to an extreme by use of the following routine: **"Order the chromosomes by decreasing evaluation. Create fitnesses that begin with a constant value and decrease linearly by a user defined amount."**

Sometimes it is required that the fitness function does not exceed a certain maximum value. If it does, this results in a constraint violation. Linear normalization takes care of this with its maximum evaluation but scaling and windowing don't. GA researchers have explored many approaches to constraint violation. Here are three possibilities:

❖ **Elimination:** This technique attempts to determine if a string violates the constraint before it is ever created. It has several problems. For starters, it may be too expensive to perform, or simply impossible. Second, preventing the creation of violators may cause genetic algorithms to overlook perfectly valid solutions. That's because violators could produce legal (non-violating) offspring that would lead to a satisfactory solution more quickly.

❖ **High Penalty:** This approach imposes a high penalty on violators. It reduces violators' worth while allowing them to occasionally propagate offspring. A weakness of this approach becomes apparent when a population contains a large percentage of violators. In this case, legal strings will dominate the following generations and the violators will be left unexploited. This effect could lead to population stagnation.

❖ **Moderate Penalty:** This approach imposes a moderate penalty on violators. It increases the probability that violators will procreate, thus reducing the chance of population stagnation. This approach exhibits its own problems, especially when violators rate higher than legal strings. In this case, if the violators do not create legal strings then the criteria for ending the search must incorporate a mechanism for detecting violators.

4.8.3 Linear Probability Curves and Steady State

The technique for giving the better members of the population a higher survival probability (and the worse members a lower probability) could be achieved by scaling the fitness value using a linear probability curve. The best member could, for example, be assigned to a probability value of 0.9, and the worst member to a probability value of 0.1. This way, not all least fit individuals would necessarily perish, and not all fittest individuals would survive and reproduce. If, for instance, the best member of the population is assigned a probability value of 1, it will certainly be reproduced, like the case with elitism.

When the simple GA reproduces, it replaces its entire set of parents by their children. This technique has some potential drawbacks. One is that even with an elitism strategy, many of the best individuals found may not reproduce at all and their genes may be lost. It is also possible that mutation or crossover may alter the best chromosomes' genes so whatever was good about them may be destroyed. One solution to these problems is to use the steady-state reproduction: as pairs of solutions are created they are inserted into the population from which their parents were taken. The new solutions replace the two worst solutions in the population and this is repeated until the number of new solutions added to the population since the last generation is equal to the number of solutions in the population.

4.8.4 Advanced Crossover Methods

❖ **Partially Mixed Crossover:** Under partially mixed crossover (PMX), two strings are aligned and two crossing sites picked uniformly at random along the strings. These two points define a matching section that is used to affect a cross through position-by-position exchange operations, PMX proceeds by positionwise exchanges as shown below.

	Before crossover	After crossover
1.	I H D|E F G |A C B J	I H D|B C J |A F E G
2.	H G A|B C J |I E D F	H J A|E F G|I B D C

❖ **Uniform Order Based Crossover:** This method preserves the ordering of elements when crossed over by generating a template string with the same length as its parents and filling in some of the positions in the child by copying them from parent 1 wherever the bit string contains a '1'. After that, a list of the elements in parent 1 associated with a '0' in the template string is made and these elements are permuted so that they appear in the same order they appear in parent 2. Finally, these permuted elements are filled into the gaps in the child in the order generated in the previous step.

Before crossover	After 2nd step	After crossover
I H D E F G A C B J	I - - E F G - - B -	I H A E F G C J B D
H G A B C J I E D F	H - - B C J - - D -	H I E B C J F G D A

mask: 1 0 0 1 1 1 0 0 1 0

4.8.5 Advanced Mutation Method

Sometimes it is required that a certain item appears only once in a string. Like uniform order-based crossover, uniform order-based mutation is a way to deal with this problem. This operator selects a section of the chromosome and mixes up the elements within the section.

Before mutation	After mutation
I D E|F G A C|B J	I D E|C G F A|B J

4.8.6 Inversion, Addition and Deletion

The techniques such as inversion, addition and deletion are the techniques of which their usefulness are still being questioned. Inversion is a reordering technique in which two points are chosen along the length of the chromosome and all the elements between these points are swapped so that they appear in reverse order. Although the results of the inversion technique were often a bit disappointing, this technique did lead to the PMX and uniform order-based crossovers described earlier.

Before inversion	**After inversion**
A B|C D E F|G H I	A B|F E D C|G H I

Sometimes addition and deletion are used i.e. a random change in a string length is imposed by either deleting an element of the chromosome or adding a new element to the chromosome. The deletion operator simply chooses at random an element to delete. With addition however, precautions have to be made that only allowed elements are added to the chromosome. This can be achieved by, for example duplicating a randomly chosen element to a randomly chosen point. Using addition and deletion operators could result in variable-length strings which call for modification of crossover, mutation and fitness techniques.