2. Genetic Algorithms – An Overview

2.1 GA Terminology

Genetic Algorithms (GAs), which are adaptive methods used to solve search and optimization problems, are based on the genetic processes of biological organisms [10]. As GA is quite efficient and easy to use in finding a nearly global optimum solution, it has been used by a number of researchers in a variety of structural engineering applications. In GAs, a design variable is not generally presented by its actual design value, instead a sufficiently long arrays of bits is used to code it.

The technical terms related to GA are as follows:

- **Coding** : The representation of problem specific variables in a form suitable for genetic operators to operate and produce better offspring. e.g. binary coding, real coding.
- **Decoding** : The reverse process of coding. The coded variables are converted to a form representing the actual value of the variables.
- **Mapping** : Mapping is the process of scaling the decoded values of various variables between their upper and lower limits.
- **Population** : Number of chromosomes or solution string in each GA generation.
- **Population Size** : Number of solution strings in one GA generation is known as population size. If there are too few chromosomes, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down.
- **Generation** : A generation is one complete cycle of various genetic operators like reproduction, crossover, mutation, selection, etc. on a set of population strings.
- **Objective Function** : It is the function indicating the main objective to be achieved i.e. representing the parameter to be optimized.
- **Constraints** : Specific conditions pertaining to the problem under consideration, which should not be violated.

- **Penalty :** It is the punishment bearded by the fitness value of a population for violating a constraint.
- **Fitness :** Fitness indicates the "goodness" or quality of the solution. Better the solution more is the fitness of the solution. It is used as a criterion for selecting strings that are to be used for creating a mating pool.
- **Fitness Scaling :** Fitness scaling is implemented to prevent some strong solution string from dominating the whole population in early generation and to give a fair survival chance to weaker population members.
- **Selection :** The technique used to determine which chromosomes in a population will serve as parents for the next generation.
- **Mating Pool :** It is a collection of strings, chosen on the basis of their fitness values, for the purpose of carrying out genetic operations like mutation and crossover in order to produce better off springs.
- **Roulette Wheel Selection :** It is used for selecting members of old population for mating and producing new members of next population, based on probability theory.
- **Parent String :** The string selected from the mating pool for producing new child strings.
- **Crossover :** The method used to exchange parts of two parent chromosomes to create two child chromosomes, which have attributes from both parents.
- **Mutation :** It is the process of occasional alteration of a bit of population strings in order to prevent the algorithm from getting trapped into local optimum solution and to maintain the diversity of population.
- **Child String :** The string resulting from the parent string after application of various genetic operators on the parent strings.
- **Elitism :** In this method, 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. Elitism can very rapidly increase the performance of GA because it prevents loosing the best found solutions.
- **Generation Gap :** The generation gap G is a parameter which controls the percentage of the population to be replaced during each generation. In order to minimize the disruption of the structure of the population and improve the exploitation efficiency, a small generation gap is beneficial.

2.2 Working of Genetic Algorithm

In the most common type of GA, a population is created with a group of individuals created randomly. The individuals in the population are then evaluated. The evaluation function is provided by the programmer which gives the individuals a score based on how well they perform at the given task. Two individuals are then selected based on their fitness. Higher is the fitness, higher is the chance of being selected. These individuals then "reproduce" to create one or more offsprings, after which the offsprings are mutated randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer. A schematic representation of the evolution process is shown in **Fig. 2.1.**

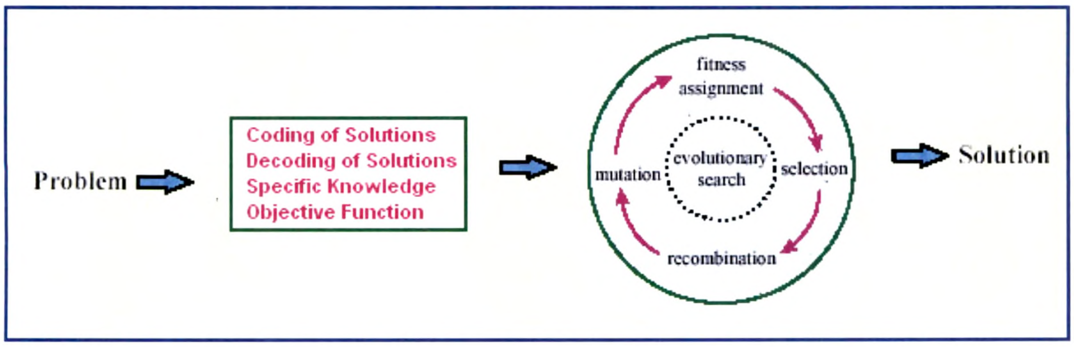


Fig 2.1 Schematic Diagram of GA Based Optimization

2.3 Coding of Design Variables

Encoding of chromosomes is one of the important tasks to be performed, when solving optimization problem using GA. There are a large number of coding schemes. The selection of a particular encoding scheme is based on the type of optimization problem under consideration. As binary encoding is generally used in structural optimization problems, it is discussed here in detail.

In binary encoding every chromosome is a string of **bits, 0** or **1**. Binary encoding gives many possible chromosomes even with a small number of alleles.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

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 substring (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:

$$X_i = X_i(L) + \left[\frac{X_i(U) - X_i(L)}{2^{li} - 1}\right] \times decoded\ value(S_i) \qquad ...(2.1)$$

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 $\Sigma\ 2^j.S_{ij}$ j = 0 to l_i-1. For example, a four bit string (1110) has a decoded value equal to $[(0).2^0 + (1).2^1 + (1).2^2 + (1).2^3]$ i.e. 14. With four bits to code each variable there are only 2^4 or 16 distinct substring 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 and is given by $[X_i (U) - X_i (L)] / 2.l_i$. After coding of variables the corresponding point $X = (X_1, X_2, ..., X_n)^T$ can be found. Thereafter, the function value at that point can be calculated by substituting X in the given objective function f(X).

2.4 Fitness Evaluation and Fitness Scaling

2.4.1 Fitness Evaluation

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 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:

$$F_x = \frac{1}{1 + f(x)} \qquad ...(2.2)$$

2.4.2 Fitness Scaling

It is very important to regulate the number of copies in a small population of GA. 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 to 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.

2.5 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.

2.5.1 Reproduction

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 [9], for minimization problems, H(x) should be subtracted from a large constant so that all the fitness value are obtained according to the actual merit. The large constant is obtained by adding the maximum and minimum value of the H(x). Thus, the expression for the fitness becomes:

$$F_i = [H(x)_{max} + H(x)_{max}] - H_i(x) \qquad \text{...(2.3)}$$

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. In the next generation these best populations are mated randomly and crossed at random lengths of the full string.

2.5.2 CROSSOVER

Crossover operator is applied to the mating pool with a hope that it would create a better string. In this, pair of strings known as parent strings is selected form the mating pool at random and some portion of the strings is exchanged between the strings. The two resulting string obtained from the crossover operation are known as child strings or offsprings. Thus, good substrings from parent strings can be combined to form a better child string, if an appropriate crossover site is chosen. The populations obtained after crossing will form new population set for the next generation. The following are few types of crossover operators available in genetic algorithm:

2.5.2.1 Single Point Crossover

One crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent as shown in **Fig. 2.2.**

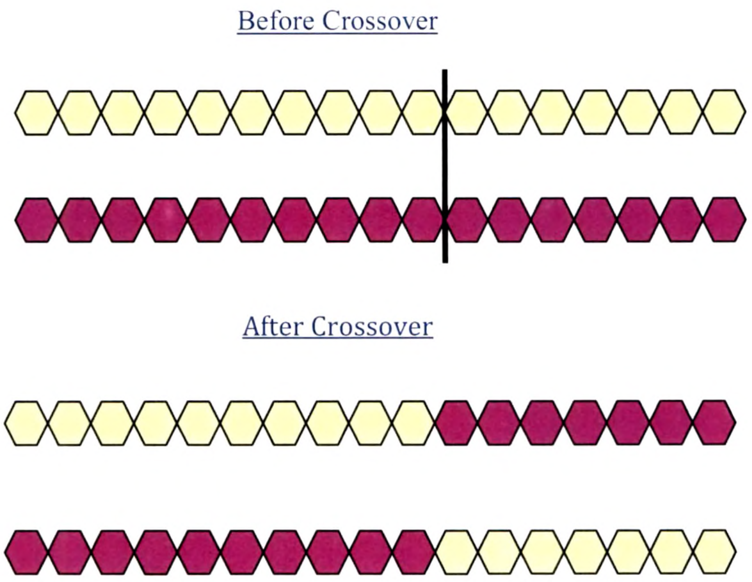


Fig. 2.2 Single Point Crossover Example

2.5.2.2 Two Point Crossover

Two crossover points are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent again as depicted in **Fig. 2.3**

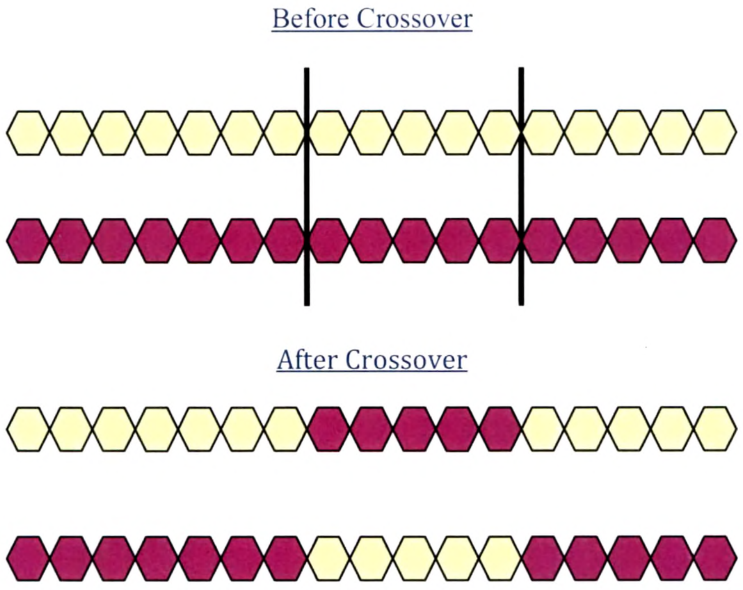


Fig. 2.3 Two Point Crossover Example

2.5.2.3 Uniform Crossover

Bits are randomly copied from the first or the second parent. This type of crossover, assign "head" to one parent and "tail" to the other parent. Then, a coin is flipped for each gene of first child. An inverse copy of first one is made for the second child as shown in **Fig. 2.4.**

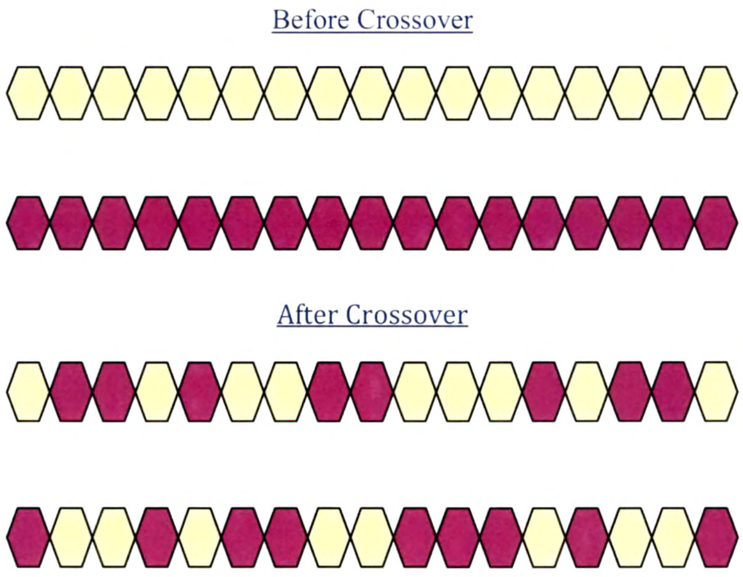


Fig. 2.4 Uniform Crossover Example

2.5.2.4 Arithmetic Crossover

Some arithmetic operation is performed to make a new offspring. A crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the following equations:

$$\text{Offspring1} = a * \text{Parent1} + (1 - a) * \text{Parent2}$$

$$\text{Offspring2} = (1 - a) * \text{Parent1} + a * \text{Parent2}$$

where, a = random weighting factor (chosen before each crossover operation).

The effect of crossover may be detrimental or beneficial. Thus, in order to preserve some of 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 Pc is used, only 100.Pc percent strings in the population are used in the crossover operation and the 100 (1 - Pc) percent of the population remains as they are in the current population.

Crossover probability means how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population. But this does not mean that the new generation is the same.

2.5.3 MUTATION

Purpose of mutation is to simulate the effect of error that happens with low probability during duplication. One can restore lost information to the population by mutation. Mutation also helps to 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.

The GA uses a mutation probability, Pm, which dictates the frequency at which mutation occurs. For each string element in each 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, the GA decides to mutate bit position 5 in the string **100000** as follows.

$$\mathbf{100000} \xrightarrow{\textbf{Mutate}} \mathbf{100010}$$

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.

2.6 TERMINATION CRITERION

Termination is the criterion by which the GA decides whether to continue searching or stop the search. Some of the well-known termination methods are:

i. **Generation Number** : A termination method that stops the evolution when the user-specified maximum numbers of evolutions have been run.
ii. **Evolution Time** : A termination method that stops the evolution when the elapsed evolution time exceeds the user-specified maximum evolution time.
iii. **Fitness Threshold** : A termination method that stops the evolution when the best fitness in the current population becomes greater than the user-specified fitness threshold.
iv. **Fitness Convergence** : A termination method that stops the evolution when the fitness is deemed as converged.
v. **Population Convergence** : A termination method that stops the evolution when the population is deemed as converged.

2.7 RECOMMENDED VALUE OF GA PARAMETERS

Here some general recommendations are given to select various GA parameters.

Crossover rate : Crossover rate generally should be high, about 80% - 95%. However, some results show that for some problems crossover rate about 60% is the best.

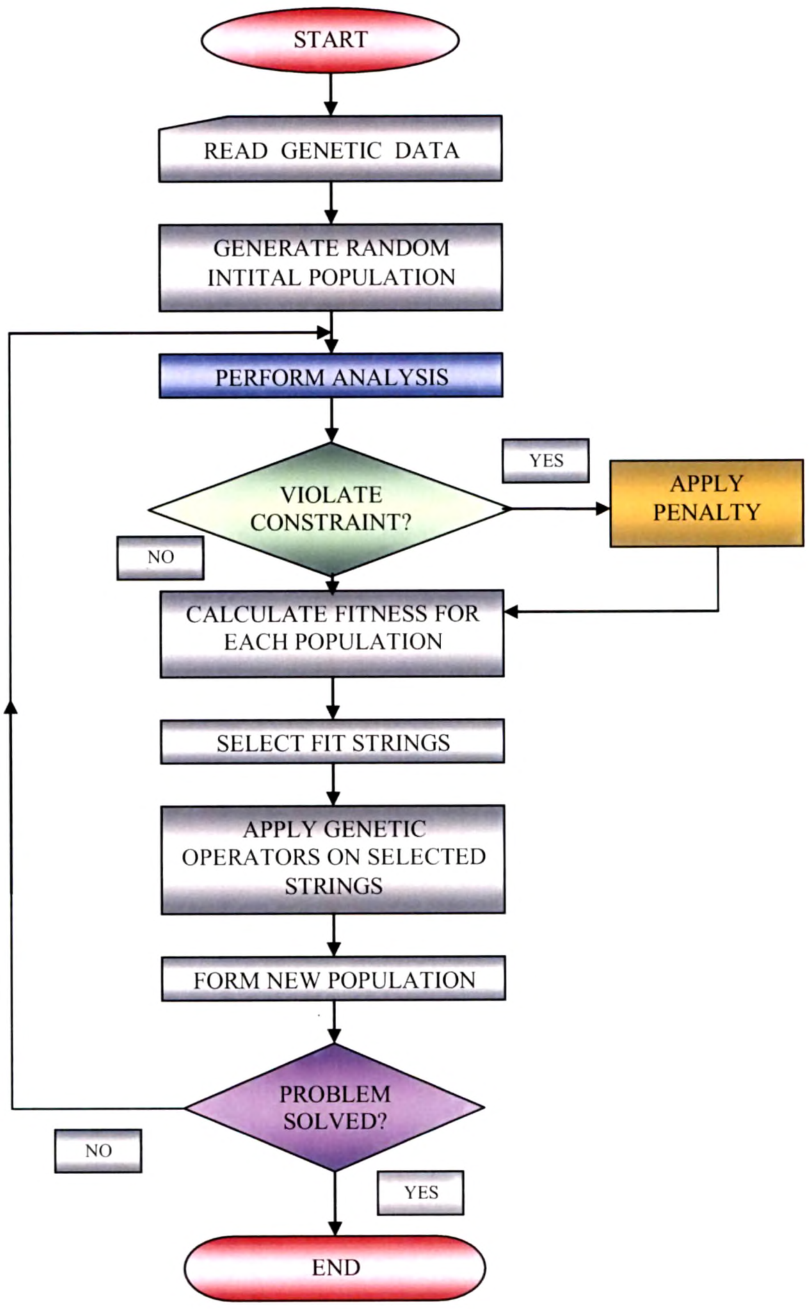
Mutation rate : Mutation rate should be low. Best rates reported are about 0.5% -1%.

Population size : It may be surprising, that very big population size usually does not improve performance of GA (in terms of speed of convergence). Good population size is about 20-30, however, sometimes sizes 50-100 are reported as the best. Some research also shows that best population size depends on encoding and on size of encoded string.

Selection : Basic roulette wheel selection can be used, but sometimes rank selection can be better. There are also some more sophisticated methods, which changes parameters of selection during run of GA. But surely elitism should be used (if any other method for saving the best found solution is not used). The Tournament selection method is seemed to have been used more often in structural optimization problems.

Encoding : Encoding depends on the problem and also on the size of instance of the problem. Selection of type of crossover and mutation operators also depends on encoding and on the type of problem.

2.8 GA FLOW CHART FOR SOLVING A PROBLEM


START
READ GENETIC DATA
GENERATE RANDOM INITAL POPULATION
PERFORM ANALYSIS
VIOLATE CONSTRAINT?
YES
APPLY PENALTY
NO
CALCULATE FITNESS FOR EACH POPULATION
SELECT FIT STRINGS
APPLY GENETIC OPERATORS ON SELECTED STRINGS
FORM NEW POPULATION
PROBLEM SOLVED?
NO
YES
END