Genetic Modelling

In the last Chapter, we discussed how variables in GA are encoded and how fitness function is calculated. Various selection methods such as Roulette-wheel selection, Boltzmann selection, tournament selection, and steady-state selection have been used to produce the population of the mating pool i.e. in reproduction, good strings in a population are probabilistically assigned a large number of copies and a mating pool is formed. It is important to note that no new strings are formed in the reproduction phase. There are many inheritance operators applied to the mating pool with a hope that it would create a better string. The aim of inheritance operators is to search the parameter space. In addition, search is to be in a way that the information stored in the present strings are maximally preserved, because these parent strings are instances of good strings selected using the reproduction operator.

9.1 INHERITANCE OPERATORS

As already seen, genetic algorithm makes use of the Darwinian survival of the fittest procedure. Genetic algorithms are search procedures based on mechanics of natural genetics and natural selection.

Genetic algorithm derives power from the genetic operators listed here. Low-level operators, namely

1. Inversion
2. Dominance
3. Deletion
4. Intrachromosomal duplication
5. Translocation
6. Segregation
7. Speciation
8. Migration
9. Sharing
10. Mating

A simple genetic algorithm largely uses three basic operators which are

1. Reproduction
2. Cross over
3. Mutation

First we will discuss the basic operators and then the low-level operators.

9.2 CROSS OVER

After the reproduction phase is over, the population is enriched with better individuals. Reproduction makes clones of good strings, but does not create new ones. Cross over operator is applied to the mating pool with a hope that it would create a better string. The aim of the cross over operator is to search the parameter space. In addition, search is to be made in a way that the information stored in the present string is maximally preserved because these parent strings are instances of good strings selected during reproduction.

Cross over is a recombination operator, which proceeds in three steps. First, the reproduction operator selects at random a pair of two individual strings for mating, then a cross-site is selected at random along the string length and the position values are swapped between two strings following the cross-site. For instance, let the two selected strings in a mating pair be A = 11111 and B = 00000. If the random selection of a cross-site is two, then the new strings following cross over would be A* = 11000 and B* = 00111. This is a single-site cross over. Though these operators look very simple, their combined action is responsible for much of GA's power. From a computer implementation point of view, they involve only random number of generations, string copying, and partial string swapping. There exist many types of cross over operations in genetic algorithm which are discussed in the following sections.

9.2.1 Single-site Cross Over

In a single-site cross over, a cross-site is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged as shown in Fig. 9.1. If an appropriate site is chosen, better children can be obtained by combining good substances of parents. Since the knowledge of the appropriate site is not known and it is selected randomly, this random selection of cross-sites may produce enhanced children if the selected site is appropriate. If not, it may severely hamper the string quality. Anyway, because of the crossing of parents better children are produced and that will continue in the next generation also. But if good strings are not created by cross over, they will not survive beyond next generation because reproduction will not select those strings for the next mating pool.

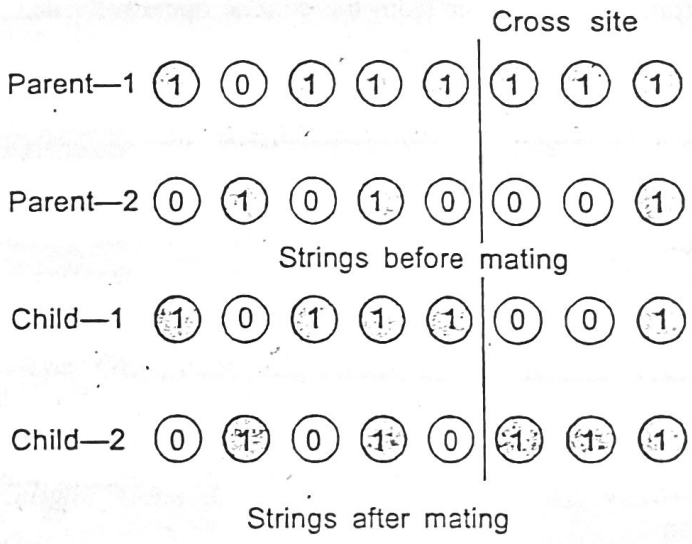


Fig. 9.1 Single-site cross over.

9.2.2 Two-point Cross Over

In a two-point cross over operator, two random sites are chosen and the contents bracketted by these sites are exchanged between two mated parents. If the cross-site 1 is three and cross-site 2 is six, the strings between three and six are exchanged as shown in Fig. 9.2.

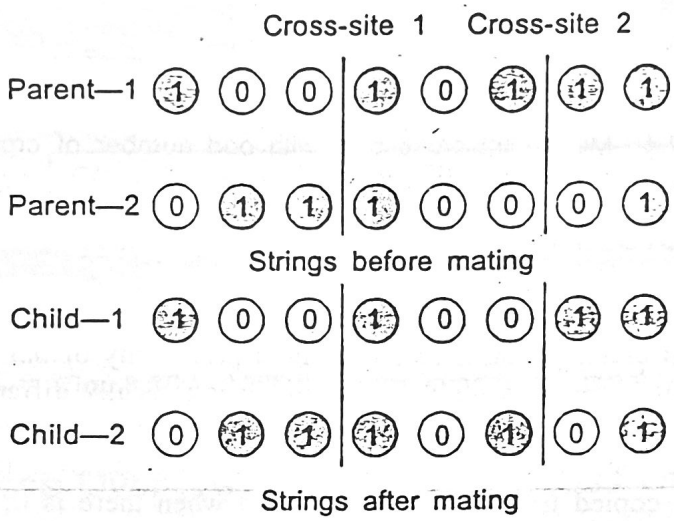


Fig. 9.2 Two-point cross over.

9.2.3 Multi-point Cross Over

In a multi-point cross over, again there are two cases. One is even number of cross-sites and second one is the odd number of cross-sites. In case of even numbered cross-sites, the string is treated as a ring with no beginning or end. The cross-sites are selected around the circle uniformly at random. Now the information between alternate pairs of sites is interchanged as shown in Fig. 9.3. If the number of cross-sites is odd, then a different cross-point is always assumed at the string beginning. The information (genes) between alternate pairs is exchanged as shown in Fig. 9.4.

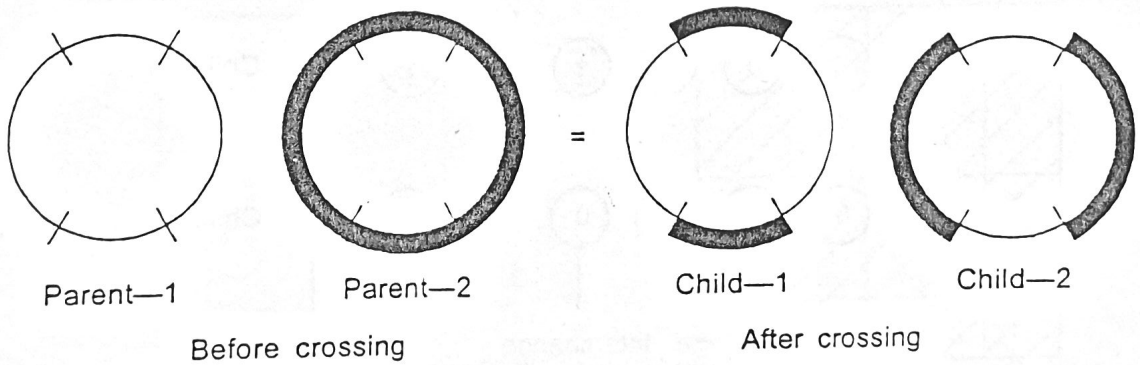


Fig. 9.3 Multi-point cross over with even number of cross-sites.

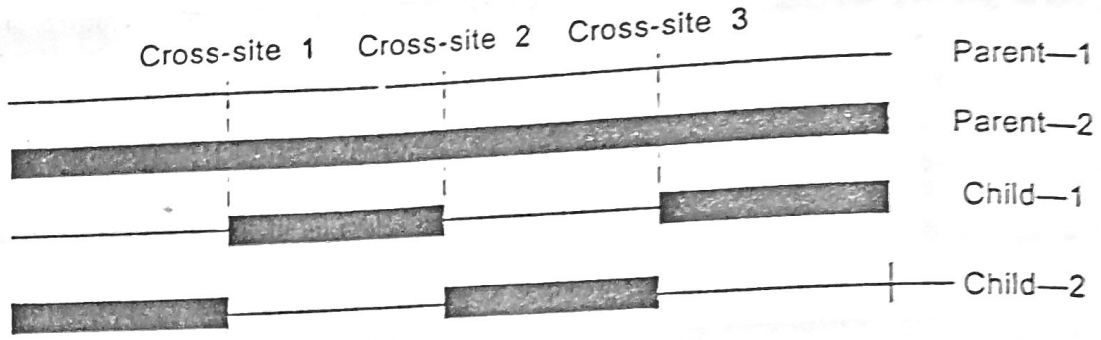


Fig. 9.4 Multi-point cross over with odd number of cross-sites.

9.2.4 Uniform Cross Over

An extreme of multi-point cross over is the uniform cross over operator. In a uniform cross over operator, each bit from either parent is selected with a probability of 0.5 and then interchanged as shown in Fig. 9.5(a). It is seen that uniform cross over is radically different from one-point cross over. Sometimes gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a randomly generated cross over mask. When there is 1 in the mask, the gene is copied from the first parent and when there is 0, the gene is copied from second parent as shown in Fig. 9.5(b). The process is repeated with the parents exchanged to produce the second offspring. A new cross over mask is randomly generated for each pair of parents. Offspring therefore contains a mixture of genes from each parent. The number of effective crossing points is not fixed but averages to L/2 (where L is chromosome length).

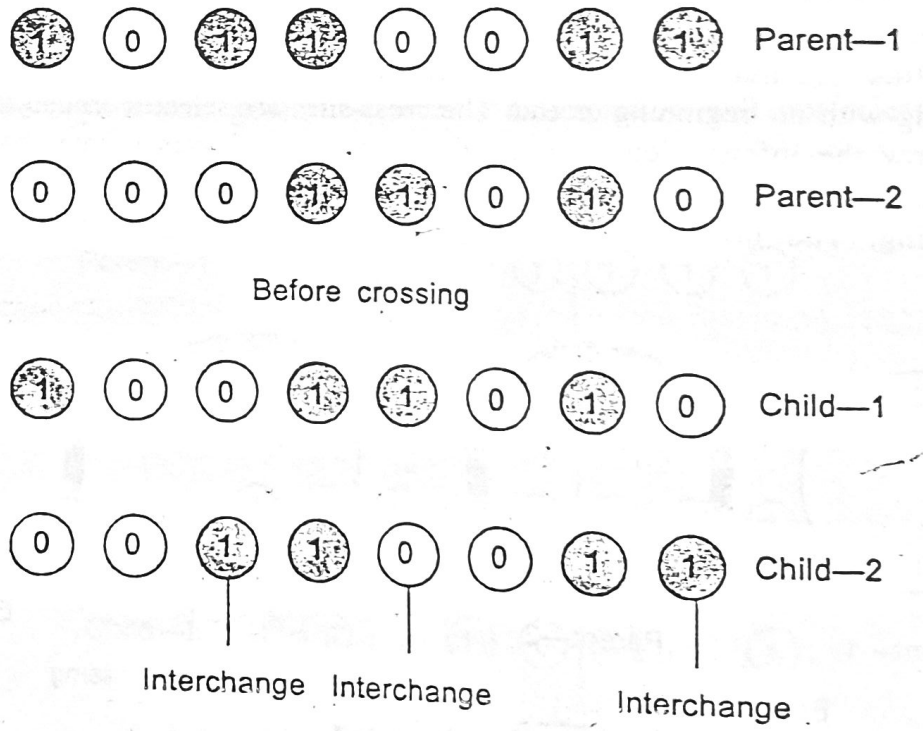


Fig. 9.5(a) Uniform cross over.

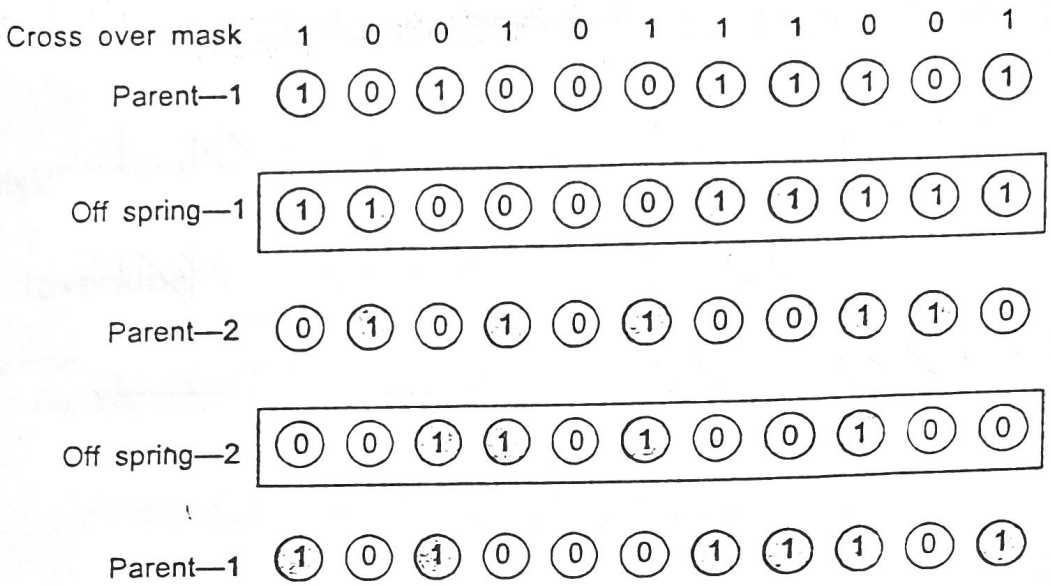


Fig. 9.5(b) Uniform cross over using mask.

9.2.5 Matrix Cross Over (Two-dimensional Cross Over)

Normally, the strings are represented as a single dimensional array as shown in Fig. 9.6. In the above case, two strings of length 4 are concatenated to form an individual. So, the cross-sites selected for this case are obviously single-dimensional whereas in the case of two-dimensional cross over, each individual is represented as a two-dimensional array of vector to facilitate the process. The process of two-dimensional cross over is depicted in Fig. 9.7.

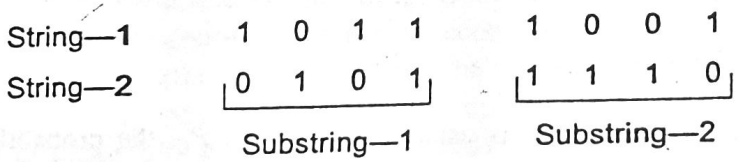
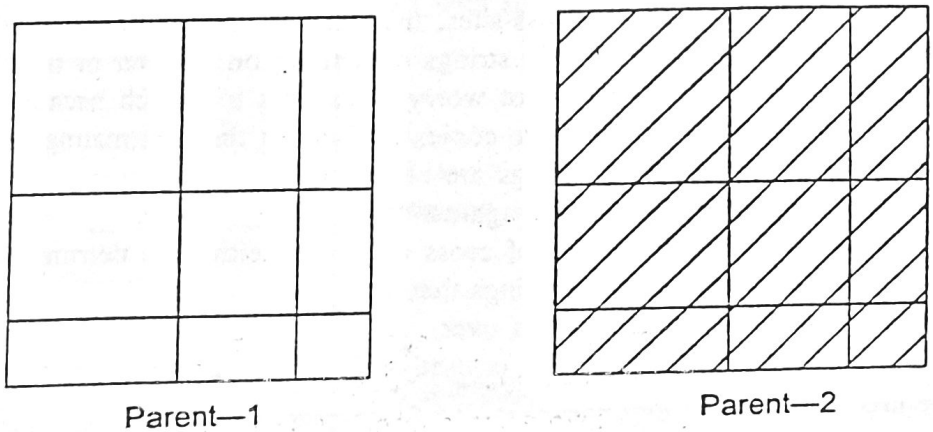


Fig. 9.6 Single-dimensional strings.



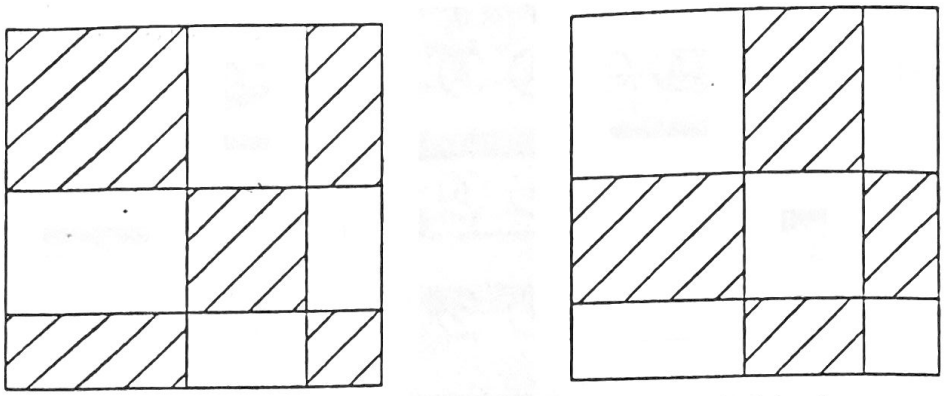


Fig. 9.7 Matrix cross over.

Two random sites along row and column are chosen, then the string is divided into utmost nonoverlapping rectangular regions. Two cross-sites, both row- and column-wise, will decide each individual into utmost nine overlapping rectangular regions. Two cross-sites, both row- and column-wise will decide each individual into three layers horizontally and vertically. Select any region in each layer, either vertically or horizontally and then exchange the information in that region between the mated populations. The selection of cross over operators is made such that the search in genetic space is proper. In case of a single-point cross over operator, the search is not extensive but maximum information is preserved between parents and children. Some studies have been made to find an optimal cross over operator. According to Deb (1995), it is difficult to generalize the optimal cross over operator selection. So, it is left to personal interest to select the cross over operator.

9.2.6 Cross Over Rate

In GA literature, the term cross over rate is usually denoted as P_C, the probability of cross over. The probability varies from 0 to 1. This is calculated in GA by finding out the ratio of the number of pairs to be crossed to some fixed population. Typically for a population size of 30 to 200, cross over rates are ranged from 0.5 to 1.

We have seen that with random cross-sites, the children strings produced may not have a combination of good substrings from parent strings depending on whether or not the crossing site falls in the appropriate place. But we do not worry about this too much because if good strings are created by cross over, there will be more copies of them in the next mating pool generated by the reproduction operator. But if good strings are not created by cross over, they will not survive too long, because reproduction will select against those strings in subsequent generations. It is clear from this discussion that the effect of cross over may either 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 cross over.

When a cross over probability of P_C is used only $100P_C$ percent strings in the population are used in the cross over operation and $100(1- P_C)$ percentage of the population remains as it is in the current population. Even though the best $100(1- P_C)\%$ of the current population can be

copied deterministically to the new population, this is usually preferred at random. A cross over operation is mainly responsible for the search of new strings.

9.3 INVERSION AND DELETION

9.3.1 Inversion

A string from the population is selected and the bits between two random sites are inverted as shown in Fig. 9.8.

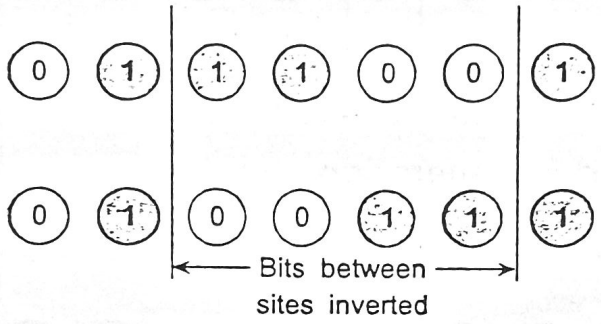


Fig. 9.8 Inversion.

Linear+end-inversion

Linear+end-inversion performs linear inversion with a specified probability of 0.75. If linear inversion was not performed, the end inversion would be performed with equal probability of 0.125 at either the left or right end of the string. Under end inversion, the left or right end of the string was picked as one inversion-point and a second inversion-point was picked uniformly at random from the point no farther away than one half of the string length. Linear+end-inversion minimizes the tendency of linear inversion to disrupt bits located near the centre of the string disproportionately to those bits located near the ends.

Continuous inversion

In *continuous inversion*, inversion was applied with specified inversion probability P_r to each new individual when it is created.

Mass inversion

No inversion takes place until a new population is created and thereafter, one-half of the population undergoes identical inversion (using the same two inverting points).

9.3.2 Deletion and Duplication

Any two or three bits at random in order are selected and the previous bits are duplicated and it is shown in Fig. 9.9.

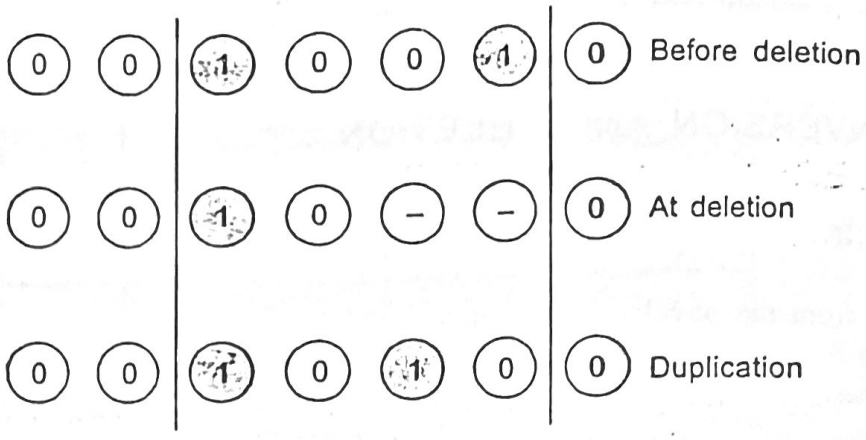


Fig. 9.9 Deletion and duplication.

9.3.3 Deletion and Regeneration

Genes between two cross-sites are deleted and regenerated randomly as shown in Fig. 9.10.

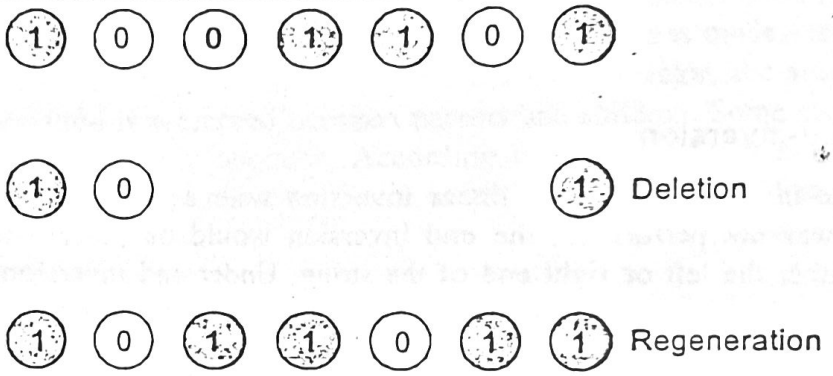


Fig. 9.10 Deletion and regeneration.

9.3.4 Segregation

The bits of the parents are segregated and then crossed over to produce offspring as shown in Fig. 9.11.

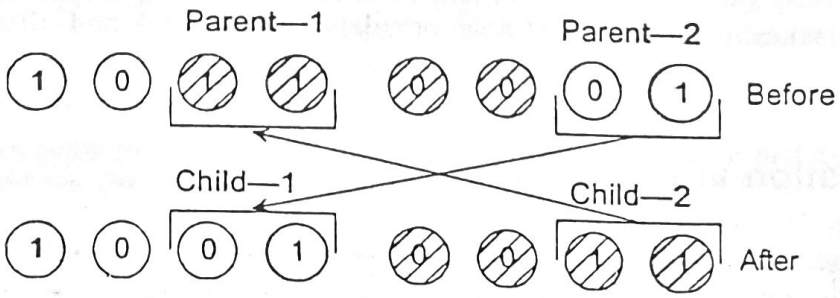


Fig. 9.11 Segregation.

9.3.5 Cross Over and Inversion

Cross over and inversion operator is the combination of both cross over and inversion operators. In this, two random sites are chosen, the contents bracketted by these sites are exchanged between two mated parents and, the end points of these exchanged contents switch place. For example, if the cross-sites in parents shown in Fig. 9.12 are 2 and 7, the cross over and inversion operation is performed in the way shown in Fig. 9.12.

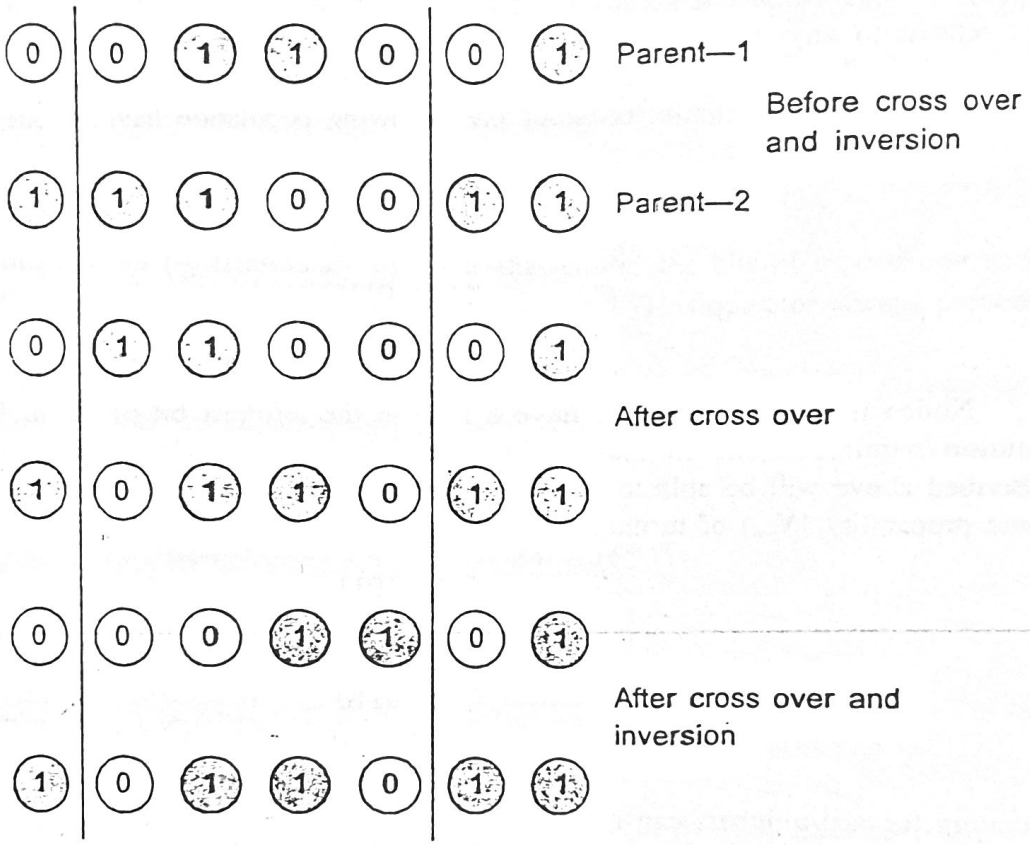


Fig. 9.12 Cross over and inversion.

9.4 MUTATION OPERATOR

9.4.1 Mutation

After cross over, the strings are subjected to mutation. *Mutation* of a bit involves flipping it, changing 0 to 1 and vice versa with a small mutation probability P_m. The bit-wise mutation is performed bit-by-bit by flipping a coin with a probability of P_m. Flipping a coin with a probability of P_m is simulated as follows.

A number between 0 to 1 is chosen at random. If the random number is smaller than P_m then the outcome of coin flipping is true, otherwise the outcome is false. If at any bit, the outcome is

true then the bit is altered, otherwise the bit is kept unchanged. The bits of the strings are independently muted, that is, the mutation of a bit does not affect the probability of mutation of other bits. A simple genetic algorithm treats the mutation only as a secondary operator with the role of restoring lost genetic materials. Suppose, for example, all the strings in a population have conveyed to a zero at a given position and the optimal solution has a one at that position, then cross over cannot regenerate a one at that position while a mutation could. The mutation is simply an insurance policy against the irreversible loss of genetic material. The mutation operator introduces new genetic structures in the population by randomly modifying some of its building blocks. It helps the search algorithm to escape from local minima's traps since the modification is not related to any previous genetic structure of the population. It creates different structure representing other sections of the search space. The mutation is also used to maintain diversity in the population. For example, consider the following population having four eight-bit strings.

0110 1011

0011 1101

0001 0110

0111 1100

Notice that all four strings have a zero in the leftmost bit position. If the true optimum solution requires a one in that position, then neither reproduction nor cross over operator described above will be able to create one in that position. The inclusion of mutation introduces some probability (N_{pm}) of turning zero to one as

0110 1011

0011 1101

0001 0110

1111 1100

Mutation for real numbers can be done as

Before (1.38 –69.4 326.44 0.1)

After (1.38 –67.5 326.44 0.1)

Hence, mutation causes movement in the search space (local or global) and restores lost information to the population.

9.4.2 Mutation Rate P_m

Mutation rate is the probability of mutation which is used to calculate number of bits to be muted. The mutation operator preserves the diversity among the population which is also very important for the search. Mutation probabilities are smaller in natural populations leading us to conclude that mutation is appropriately considered a secondary mechanism of genetic algorithm adoption. Typically, the simple genetic algorithm uses the population size of 30 to 200 with the mutation rates varying from 0.001 to 0.5.

9.5 BIT-WISE OPERATORS

Usually, binary coding is used more extensively in the coding mechanism to generate algorithm structure. This involves the coding of real variables to binary strings and genetic operators work on these coded strings. In the present work, genetic algorithm program as written in "C" language with the help of built in operators in "C" (the bit-wise operators), we can directly manipulate the individual bits within a word of memory. These operators can be carried out easily and efficiently. These can operate on integers and characters but not on floats and doubles. Byron S. Gotfried (1990) categorizes bit-wise operators into three, namely, 1. The one's complement operator, 2. The logical bit-wise operator, and 3. The shift operator.

9.5.1 One's Complement Operator

The one's complement operator (~) is an unary operator that causes the bits of its operand to be inverted (i.e. reversed), so that 1 becomes zero and zero becomes 1. This operator always precedes its operand.

Consider the example problem 8.1, where two variables θ_1, θ_2, are four-bit strings each respectively and hence, the total string length is eight.

			θ_1	θ_2
a = 0100 0001 →	4	1	24°	16°
$\sim a$ = 1011 1110 →	11	14	66°	84°

9.5.2 Logical Bit-wise Operators

There are three logical bit-wise operators, namely, 1. Bit-wise AND, 2. Bit-wise Exclusive-OR, and 3. Bit-wise OR.

Each of these operators require two integer-type operands and hence, can be used instead of cross over. While operating upon two operands, they are compared on bit by bit basis. The truth table is shown in Table 9.1.

Bit-wise AND (&) operator

A bit-wise AND (&) expression returns 1 if both the bits have a value 1, otherwise it returns a value 0.

Parent 1a = 1010 1010 →	10	10
Parent 2b = 1100 0011 →	12	3
Child a&b = 1000 0010 →	8	2

Table 9.1 Truth table

a	b	AND '&' operator a&b	Exclusive OR '∧' operator a ∧ b	OR '|' operator a | b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Bit-wise exclusive-OR (∧) operator

A bit-wise exclusive-OR (∧) expression returns a 1 if one of the bits have a value of 1 and the other has a value of 0 otherwise it returns a value 0.

Parent 1a = 1010 1010 → 10 10
Parent 2b = 1100 0011 → 12 3
Child a&b = 0110 1001 → 6 9

Bit-wise OR (|) operator

A bit-wise OR (|) expression returns a 1 if one or more bits have a value of 1 otherwise it returns a value 0.

Parent 1a = 1010 1010 → 10 10
Parent 2b = 1100 0011 → 12 3
Child a&b = 1110 1011 → 13 11

The three bit-wise operators are summarized in Table 9.1. In this Table, A and B represent the corresponding bits within the first and second operands respectively.

9.5.3 Shift Operators

Two bit-wise shift operators are, *shift left* (<<) and *shift right* (>>) operators. Each operator operates on a single variable but requires two operands. The first operand is an integer type operand that represents the bit pattern to be shifted and the second is an unsigned integer that indicates the number of displacements (i.e. whether the bits in the first operand will be shifted by 1 bit position, 2 bit position and so on). This value cannot exceed the number of bits associated with the word size of the first operand.

Shift left operator (<<)

The shift left operator causes all the bits in the first operand to be shifted to the left by the number of positions indicated by the second operand. The leftmost bits (i.e. the overflow bits) in the original bit pattern is lost. The rightmost bit positions that become vacant are to be filled with zeroes.

a = 1010 0110 → 10 6
a << 2 = 1001 1000 → 9 8

Shift right operator (>>)

The shift right operator causes all the bits in the first operand to be shifted to the right by the number of positions indicated by the second operand. The right most bits (i.e. the underflow bits) in the original bit pattern are lost. The left most bit positions that become vacant are then filled with zeroes.

$$a = 1010\ 0110 \rightarrow \quad 10 \quad 6$$
$$a >> 2 = 0010\ 1001 \rightarrow \quad 2 \quad 9$$

Masking

Masking is a process in which a given bit pattern is transformed into another bit pattern by means of logical bit-wise operation. The original bit pattern is one of the operands in the bit-wise operation. The second operand called *mask*, is a specially selected bit pattern that brings about the desired transformation.

There are several different kinds of masking operations. For example, a portion of a given bit pattern can be copied to a new word, while the remainder of the new word is filled with 0. Thus, part of the original bit pattern will be "masked off" from the final result.

9.6 BIT-WISE OPERATORS USED IN GA

Logical bit-wise operators are used in different combinations. Each operator operates on two individuals and generates one resultant so as to keep the number of individuals in the population constant. Two different operators are used in GA process.

Populations are selected randomly for mating and on each pair bit-wise AND and bit-wise OR operators are performed. Similarly, AND and exclusive-OR or OR and exclusive-OR operations can be performed to produce children or population for the next generation.

9.7 GENERATIONAL CYCLE

Table 9.2 shows a *generational cycle* of the genetic algorithm with a population of four (P1 = 4) strings with 10 bits each. In this example, the objective functions which can assume values in the string 0 to 10, give the number of 1s in the decimal place. The fitness function performs "divide by 10" operation to normalize the objective function in the range of 0 to 1. The four strings thus have fitness values of 0.3, 0.6, 0.6, and 0.9. Ideally, the proportional selection scheme should allocate 0.5(0.3/0.6), 1.0(0.6/0.6), 1.0(0.6/0.6), and 1.5(0.9/0.6) values for selection to be offspring (since f(average) = (0.3 + 0.6 + 0.6 + 0.9)/4 = 0.6) to the strings. However, according to Darwinian theory of survival of the fittest, the strongest individual will have two copies, the weakest individual dies, and average individuals will have one copy each. Hence, the string with fitness value of 0.5 will have 0 copy with 1.5 has two copies and others 1 copy. In Table 9.2, the population P2 represents this selected set of strings. Next, the four strings are paired randomly for cross over. Strings 1 and 4 forms one pair, and 2 and 3 forms the other pair. At a cross over probability rate of 0.5, only the pair 2 and 3 is left intact. The cross over point falls between 1 and 5 and hence, portion of the strings between 1 and 5 are swapped.

The action of mutation on population P3 can be seen in population P4 on the sixth bit of string 2 and the first bit of string 4. Only two bits out of 40 have muted representing an effective

mutation probability rate of 0.05. Population P4 represents the next generation. In effect, P1 and P4 are the populations while P2 and P3 represent the intermediate stages in the generational cycle.

The parameters, namely the population size, mutation rate, and cross over rate are together referred to as the *control parameters* of the simple genetic algorithm and must be specified before its execution.

To terminate the execution of simple genetic algorithm, we must specify a stopping criterion. It could be terminated after a fixed number of generations, after a string with a certain high fitness value is located or after all the strings in the populations have attained a degree of homogeneity (a large number of strings have identical bits at most positions).

Table 9.2 A generational cycle of the simple genetic algorithm

Population P1

String	f = Fitness	$f/f(av)$	Copy
00000 11100	0.3	0.5	0
10000 11111	0.6	1.0	1
01101 01011	0.6	1.0	1
11111 11011	0.9	1.5	2

Population P2 after reproduction

String	f = Fitness	Cross over	CS1	CS2
10000 11111	0.6	4	1	5
01101 01011	0.6	–		
11111 11011	0.9	–		
11111 11011	0.9	1	1	5

Population P3 after cross over

String	f = Fitness
11111 11111	1.0
01101 01011	0.6
11111 11011	0.9
10000 11011	0.5

Population P4 after mutation

String	f = Fitness
01111 11111	0.9
01101 11011	0.7
11111 11011	0.9
10000 11011	0.5

Example Problem

Consider Example 8.1 (Two bar pendulum) using cross over and mutation. We have seen various selection procedures of obtaining the populations for the mating pool as given in Table 8.13 and as given in Table 9.3). One procedure is described here.

Table 9.3 Generation of population ($\bar{F} = 3.95$)

Pop. no.	Population	Mate with	CS1	CS2	Population after cross over	Random bits for mutation	Population for next generation	$F_i = \phi$	$F_i / \bar{F}_i$	Actual count	Population for next mating pool
1	2	3	4	5	6	7	8	9	10	11	12
1	0010 0001	5	2	6	0010 1001		0010 1001	4.11	1.04	1	0010 1001
2	0001 0101	6	1	5	0110 1101		0110 1101	4.53	1.14	2	0110 1101
3	0010 1000	7	4	8	0010 1100	21	0010 0100	3.15	0.796	0	0110 1101
4	0110 1010	8	4	6	0110 1110		0110 1110	4.40	1.11	1	0110 1110
5	0110 1010	1	2	6	0110 0010		0110 0010	3.00	0.759	0	0111 1100
6	1110 1101	2	1	5	1001 0101		1001 0101	3.49	0.883	1	1001 0101
7	0111 1100	3	4	8	0111 1000		0111 1000	4.44	1.12	1	0111 1000
8	0111 1100	4	4	6	0111 1000	62	0111 1100	4.55	1.15	2	0111 1100

Step 1: Randomly select eight populations of eight-bit strings and decode the population for angles, and substituting in the potential expression find the fitness function.

Step 2: Use any of the selection methods discussed in Chapter 8 to get the population for the mating pool.

(The above two steps have been performed in Chapter 8 and the column 8 of Table 8.13 gives the populations for the mating pool.)

Step 3: Randomly select the parents for the mating pool such as 1 with 5, 2 with 6, 3 with 7, and 4 with 8.

Step 4: Two-point cross over is selected such that bits between strings 2–6 of the population parents 1 and 5, are swapped. The population after cross over is shown in Table 9.3 in the sixth column. We used the cross over probability of 100% in all the parent pairs that are crossed. Figure 9.13 shows how points cross over and form new points. The points marked with small boxes are the points in the mating pool and the points marked with small circles are children points created after cross over operation. The complete population at the end of cross over operation is shown as last column in Table 9.3. Figure 9.13 shows that some good points and some not-so-good points are created after cross over. In some cases, points far away from the parent points are created and in some cases, points close to the parent points are created.

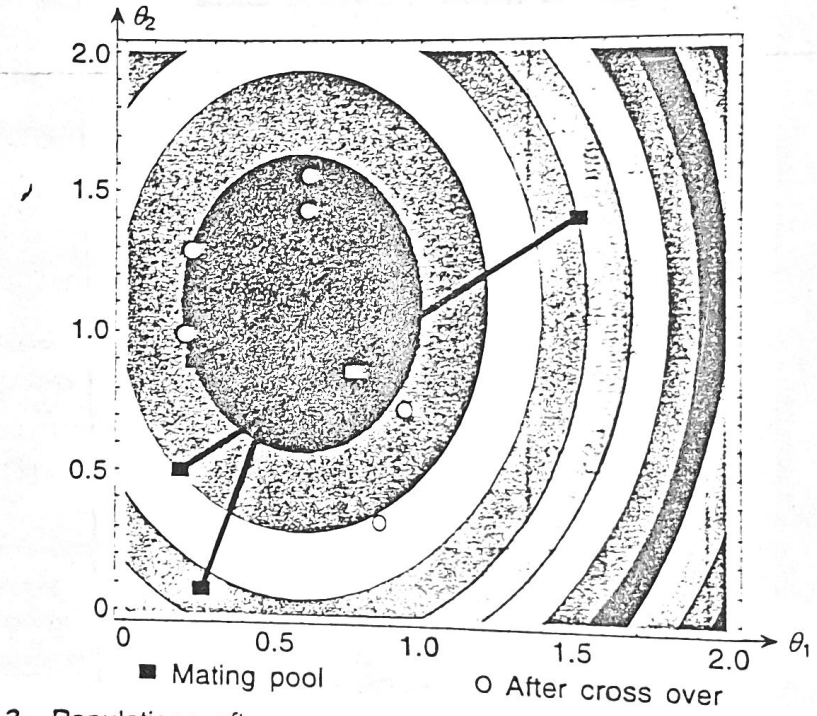


Fig. 9.13 Populations after cross over. (Two-point cross over for all populations)

Step 5: The next step is to perform mutation on strings in the intermediate population. For bit-wise mutation, we flip a coin with a probability of $P_m = 3\%$ for every bit. If the outcome is true, we alter the bit to 1 or 0 depending the bit value with a probability of $P_m = 0.03$ and for a population size of 8 and a string length of 8, we can expect to alter a total of about $0.03 \times 8 \times 8 = 1.92$ or two bits in the population. These two bits are selected at random as 21 and 62. The 21st bit which is 1, is flipped to 0 and 62nd bit which is zero, is flipped to zero as shown in Table 9.3. Figure 9.14 shows the effect of mutation on the intermediate population. In some

cases. the mutation operator changes a point locally and in others it can bring a large change. The points marked with a circle are the points of intermediate population and the points marked with a small box constitute new population (obtained after reproduction, cross over, and mutation). It is interesting to note that if only one bit is muted in a string, the point is moved along a particular variable only. Similar to the cross over operator, the mutation operator has created some points worse than the original points. This flexibility enables GA operators to explore the search space properly before converging to a region prematurely. Although this requires extra computation, this flexibility is essential to solve global optimization problems.

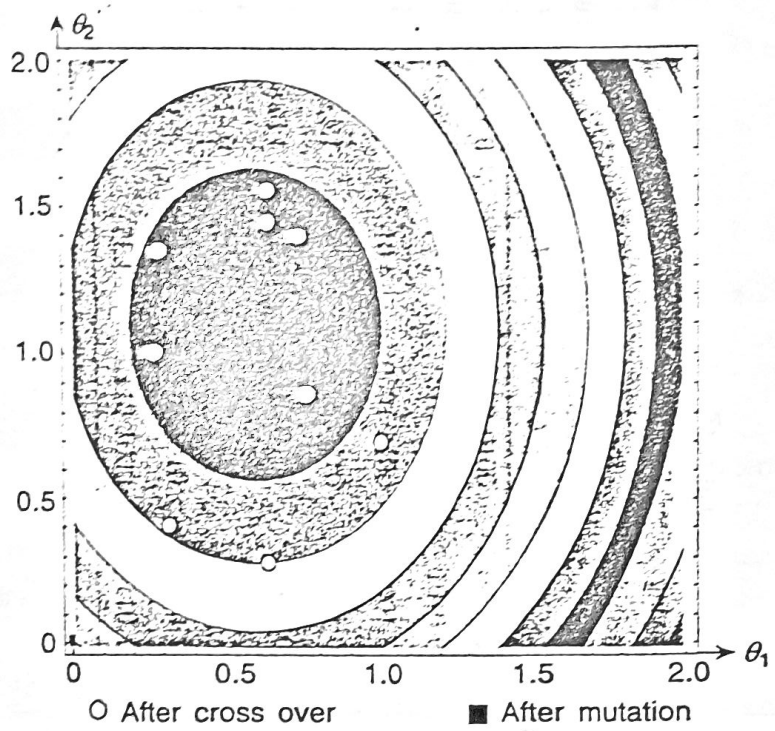


Fig. 9.14 Population after mutation operation. (Average fitness value increased from 3 to 3.95)

Step 6: The resulting population becomes a new population as shown in column 8 of Table 9.3. We now evaluate each string as before by first identifying substrings for each variable and mapping the decoded values of the substrings in the chosen intervals. This completes one iteration of genetic algorithm. We increment the generation counter to $t = 1$ and proceed to step 2 for next generation. The new population after one iteration of GA is shown in Fig. 9.14 (marked with empty boxes). The figure shows that in one iteration, some good points have been found. Table 9.3 also shows the fitness values and objective function value of the new population number.

The average fitness of the new population is calculated to be 3.95 (compared to 3 to start with), a remarkable improvement from that in the initial population. The best point in the population is found to have fitness value equal to 4.67. This process continues until the maximum allowable generation is reached or some other criterion is met. The population after 5 iterations, the best point is found to be (35°, 66°) with a function value of 11.67. In our process the total number of function evaluations required to obtain this solution is $8 \times 5 = 40$ (including the evaluation of the initial population).

Computer program

A computer program (GAOPT) for optimization of a function subjected to constraints by GA is developed and is given in CD-ROM attached with this book.

Table 9.4 Generation of population ($\bar{F} = 3.15$)

Pop. no.	Population	Mate with	Population after cross over	Applying right shift operator to 5	Population for next generation	$F_i = \phi$	$F_i/\bar{F}_i$	Actual count	Population for next mating pool
1	2	3	4	5	6	7	8	9	10
1	0010 0001	5 &	0010 0000	0010 0000	0010 0000	1.7	0.539	0	0110 1000
2	0001 0101	6 &	0000 0101	0000 0101	0000 0101	2.73	0.866	1	0000 0101
3	0010 1000	7 &	0010 1000	0010 1000	0010 1000	4.01	1.27	1	0010 1000
4	0110 1010	8 &	0110 1000	0110 1000	0110 1000	4.51	1.438	2	0110 1000
5	0110 1010	1 ∧	0100 1011	0010 0101	0010 0101	3.43	1.088	1	0010 0101
6	1110 1101	2 ∧	1111 1000	1111 1000	1111 1000	1.31	0.415	0	0101 0110
7	0111 1100	3 ∧	0101 0110	0101 0110	0101 0110	4.16	1.32	2	0101 0110
8	0111 1100	4 ∧	0001 0110	0001 0110	0001 0110	3.35	1.06	1	0001 0110

Example Problem Using Bit-wise Operators

Let us start with the populations obtained in Chapter 8 for the mating pool. First two steps are the same as previous example.

Step 3: Randomly select the parents from the mating pool such as 1 with 5. 2 with 6, 3 with 7. and 4 with 8.

Step 4: All the populations are selected for mating since the cross over probability is 100. Since there are eight individuals, there are four pairs.

The operator which is responsible for search in the genetic space bit-wise AND (&) and exclusive-OR ($\wedge$) is carried out as follows:

$$\begin{aligned} 1 &\rightarrow 0010\ 0001 \quad 2 \quad 1 \\ 5 &\rightarrow 0110\ 1010 \quad 6 \quad 10 \\ 1\ \&\ 5 &\rightarrow 0010\ 0000 \quad 2 \quad 0 \\ 1 \wedge 5 &\rightarrow 0100\ 1011 \quad 4 \quad 11 \end{aligned}$$

Similar to mutation, we can also apply shift operators. Assuming probability is 10%, we can apply to one string say random 5 as

$$\begin{aligned} a &\rightarrow 0100\ 1011 \\ a >> 1 &\rightarrow 0010\ 0101 \end{aligned}$$

Compared to genetic operators, bit-wise operators do not carry genetic information all through generations and hence, it takes longer time to converge.

9.8 CONVERGENCE OF GENETIC ALGORITHM

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

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

The convergence criteria can be explained from schema point of view in the lines of Goldberg (1989). A *schema* is a similarity template describing a subset of strings with similarities at certain positions. In other words, a schema represents a subset of all possible strings that have the same bits at certain string positions. As an example, consider a string with five bits. A schema **000 represents the strings 00000, 01000, 10000, and 11000. Similarly a schema 1*00* represents the strings 10000, 10001, 11000, and 11001. Each string represented by a schema is called an *instance* of the schema. The symbol * signifies that a 0 or 1 could occur at the string position. Thus, the schema ***** represents all possible strings of five bits. The fixed positions of a schema are the string positions that have 0 or 1 (In **000, the third, fourth and fifth positions).

The number of fixed positions in a schema is its *order* (**000 is of order 3). The *defining length* of schema 1*00* is 3. Any specific string is simultaneously an instance of 2^p schemata (p is the string length).

Since schema represents a robust of strings, we can associate a fitness value with a schema, i.e. the *average fitness* of the schema. Hence, a schema's average fitness value varies with the population's composition from one generation to another.

One can visualize GA's search for the optimal strings as a simultaneous competition among schemata increases the number of their instances in the population. If we describe the optimal string as just a position of schemata with short defining lengths and high average fitness values, then the winners of individual schema competitions could potentially form the optimal string. Such schemata with high fitness values and small defining lengths are appropriately called *building blocks*. While genetic operators are applied on a population of strings, a number of such building blocks in various parts along the string get emphasized. Individual processing shows that the worst schema will die. But there is every danger that the cross over operator may destroy good schemata. So, selection of good appropriate cross over operator plays a vital role here. If the cross-site cuts the well defined position in the schema, this may not be preserved in the next generation unless otherwise. Second parent also will have the same schema in that position. In case of single-point cross over, falling of cross-sites within the defined positions has less probability while in the case of uniform cross over, disturbance of good schema takes place with a higher probability. As far as GA to engineering field is concerned single- and two-point cross over are common. The meaning of search in the genetic space is the development of building blocks. Building blocks are combined together due to combined action of genetic operators to form bigger and better building blocks and finally converge to the optimal solution. The GA cycle is shown in Fig. 9.15.

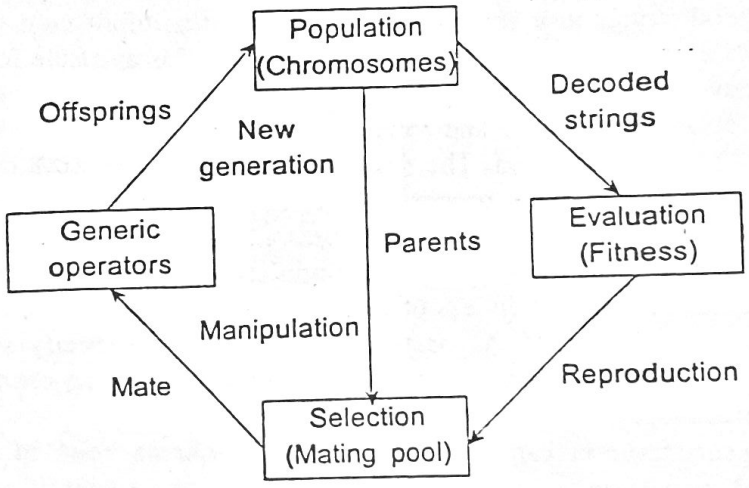


Fig. 9.15 The GA cycle.

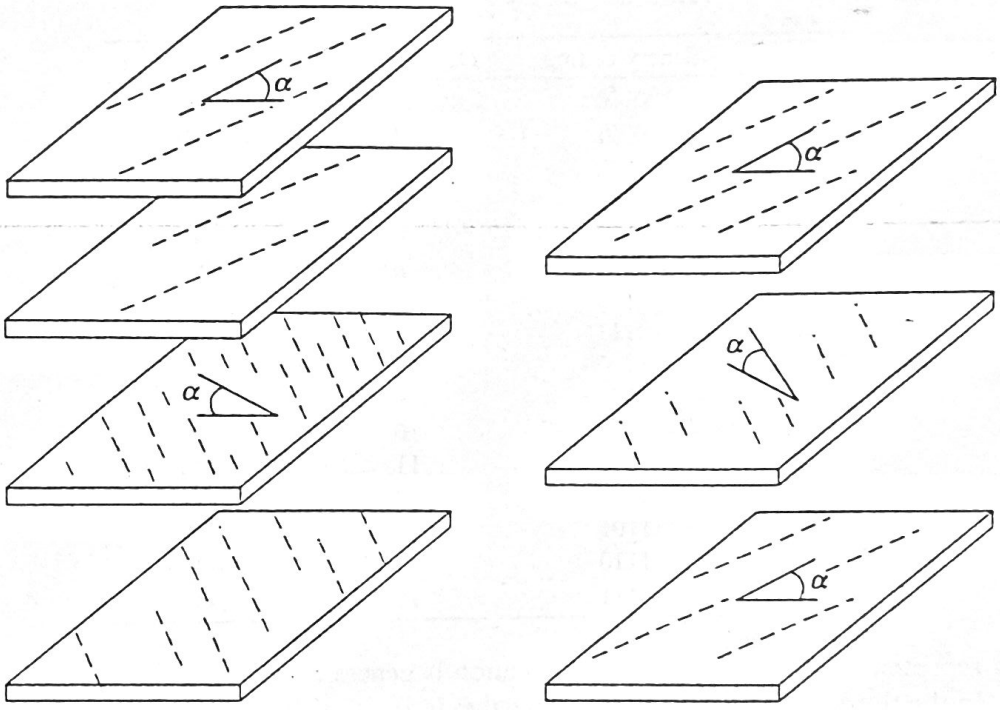
9.9 APPLICATIONS

9.9.1 Composite Laminates

Composite laminates are used for various applications. They are ideal for structural applications

where high strength to weight ratio is required. Aircraft and other space vehicles are typical weight sensitive structures in which composite materials such as Boron/Epoxy, Carbon/Epoxy, Graphite/Epoxy has resulted in the use of laminated fibre composites and shells. Laminated composites are made by binding together number of layers of at least two different materials. By lamination one can achieve two aspects of constituent layers in order to improve the material quality. Fibre-reinforced composites can be tailored to achieve the required stiffness and strength in a structure by a laminate. This is because the mechanical properties of each ply constituting the laminate can be altered by varying its fibre orientation. A composite offers a weight saving of 24–40% as compared to metallic material if used efficiently.

In a symmetric-angle ply laminate, the fibres are oriented, as shown in Fig. 9.16, symmetrically with respect to the middle layer whereas in an antisymmetric-angle ply laminate, the fibres are oriented as shown in Fig. 9.16. Antisymmetric-angle ply laminate can have only even number of layers whereas symmetric-angle ply laminate can have both odd and even number of layers.



Antisymmetric-angle ply laminate Symmetric-angle ply laminate

Fig. 9.16 Layered composite plate.

Vibration problem is a predominant problem in aerospace structures where minimum weight design is an important criterion. The frequency of a composite laminate varies with the orientation of fibres in the laminates. The basic design problem of a composite structure is to find the optimum orientation of fibres for maximum frequency. Fibre angles, in case of even number of layers including the middle surface, are taken as design variable for GA. The bottom half of layers for symmetric orientation (the same number of layers as the top half) and for antisymmetric orientation, the layers in the top half with negative sign are used. Only one half of the layers orientations are used in the coding of GA alongwith the well known operations such as reproductions and mutations.

In this subsection, working of GA is explained with reference to a three-layered symmetric orientation of a thin composite square plate with carbon/epoxy subjected to free vibrations. The assumed data for square plate is—side of square plate = 40 mm, thickness = 0.8 mm.

In this example design variable for the three-layered plate is 2. Since the plate has a symmetric orientation, only half of the layers above the middle layer including the middle are considered. The orientation of fibres can be varied as discrete values from +90 to –80. A four-bit substring is used to code each variable and in this case, a variable can take 16 discrete values (since the angle varies form 90 to –80, it is divided as –80, –75, –60, –45, –30, –20, –10, 0, 10, 20, 30, 45, 60, 75, 80, 90, i.e. sixteen angles to select a four-bit binary string) as shown in Table 9.5. Here, eight concatenated strings are adopted to represent two design variables.

The number of populations depends on the importance of the problem and the complexity involved. The number should be even to facilitate mating. In this example, number of populations is limited to eight for the purpose of illustration.

Table 9.5 Binary representation of angles

S.no.	Binary coding	Decoding angle	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	75
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	–75
16	1111	15	–80

The string representing individuals in the population is generated randomly as shown in column 2 of Table 9.6. In the third column, first value shows the design variable corresponding to layer 1. This is obtained by decoding the first substring of length four of column 2 and getting the corresponding angle from Table 9.5. For example, the substring corresponding to layer 1 from the first string is 1000. The decimal equivalent of the binary number is 8 and the corresponding angle is 90 degrees. The fibre orientation of second layer is 1001, i.e. binary number is 9 and the angle is –10 degrees. Similarly, other strings are also decoded and the corresponding angles from the list are obtained. After knowing the angles, they are repeated for the second half, which is the mirror image of first half being symmetric. For the above angles in fibres, the frequency is calculated using FEAST-C (1997) for each population. Column 4 shows the frequency for each population.

Having obtained the fitness values for all the populations, the next step is to generate the population for the next generation which are the offsprings of the current generation. The genetic operators, namely reproduction and cross over are used to generate population for the next generation. The reproduction operator selects the best individuals from the current population and

Table 9.6 Details of computations

S.no.	Population	Decoded angles	Frequency	Count	Mating pool	Mate	CS1	CS2	Population after cross over	Angles decoded	Frequency	Count	Mating pool	Mate	CS1	CS2	Population after cross over
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1000 1001	90, −10, 90	.000535	1	1000 1001	3	1	4	1100 1001	−45, −10, −45	.000646	1	1100 1001	3	1	3	1100 1001
2	1010 1100	−20, −45, −20	.000571	1	1010 1100	6	2	4	1010 1100	−20, −45, −20	.000571	1	1010 1100	8	2	4	1001 1100
3	0100 1001	45, −10, 45	.000662	1	0100 1001	1	1	4	0000 1001	0, −10, 0	.000534	0	1100 0101	1	1	3	1100 0101
4	1100 0101	−45, 60, −45	.000685	2	1100 0101	7	2	4	1100 0101	−45, 60, −45	.000685	1	1100 0101	7	1	3	1100 0101
5	0001 0010	10, 20, 10	.000542	1	0001 0010	8	1	4	0110 0010	75, 20, 75	.000553	1	0110 0010	6	1	3	0110 0010
6	0110 0101	75, 60, 75	.000555	1	0110 0101	2	2	4	0110 0101	75, 60, 75	.000555	1	0110 0101	5	1	3	0110 0101
7	1000 0111	90, 80, 90	.000534	0	1100 0101	4	2	4	1100 0101	−45, 60, −45	.000685	2	1100 0101	4	1	3	1100 0101
8	1110 1111	−75, −80, −75	.000551	1	1110 1111	5	1	4	1001 1111	−10, 80, −10	.000541	1	1001 1111	2	2	4	1010 1111

places them in the mating pool. The reproduction process is carried out in the following manner. The population, which gives the highest frequency, gets two copies and the population which gives the lowest dies off. The other populations get single copy in the mating pool. This process of reproduction confirms the Darwinian principle of survival of the fittest. The mating pool is shown in column 6 of Table 9.6.

The operator responsible for search in genetic space called cross over, is carried now. The cross over rate is selected as 1, that is, all the populations in the mating pool are to be mated. Before selecting the cross-sites, the individuals in the mating pool are matched. Since there are eight individuals there are eight matching pairs. The pairs are selected randomly are shown in column 7 of Table 9.6. Two cross-sites are chosen randomly along the length of the string for each pair as shown in column 8 and 9. Column 10 shows the population after cross over, which is the population set for Generation 2. Now the same process is repeated for Generation 2 and the details are shown in Table 9.6.

It can be observed from the Table 9.6 that the average frequency is more than the previous generation. It clearly shows the improvement among the set of populations. As one proceeds with more generations, there may not be much improvement in the populations and the best individual may progress. The population gets filled by more fit individuals with only slight deviation from the fitness of the best individual so far found and the average fitness comes very close to the fitness of the best individual. Number of generations is left to the personal interest. If a satisfactory result is obtained, iterations can be stopped or it can be stopped when there is no significant improvement in the performance from generation to generation for a particular number of generations. In the present study, the convergence of 80% to 85% of the population matrix becomes the same when compared to the previous generation and the iteration has been stopped. Table 9.7 shows the optimum orientation of fibres for maximum frequency for various layers.

Table 9.7 Optimum fibre orientation for max frequency

No. of layers	Symmetric		Antisymmetric	
	Fibre angle	Max freq	Fibre angle	Max freq
2	45, 45	0.000646	90, –90	0.000533
3	–45, 30, –45	0.000686		
4	–45, 45, 45, –45	0.000782	45, –45, 45, –45	0.000808
5	–45, 45, 30, 45, –45	0.000846		
6	–45, 45, 45, 45, 45, –45	0.000888	–45, 45, 45, –45, –45, 45	0.000920
7	45, –45, –45, 60, –45, –45, 45	0.000907		
8	–45, 45, 45, 45, 45, 45, 45, –45	0.000921	–45, 45, 45, –45, 45, –45, –45, 45	0.000927
9	–45, 45, 30, 45, –60, 45, 30, 45, –45	0.000914		

The convergence history, in number of iterations vs frequency for antisymmetric orientation of fibres for 8 layers is shown in Fig. 9.17.

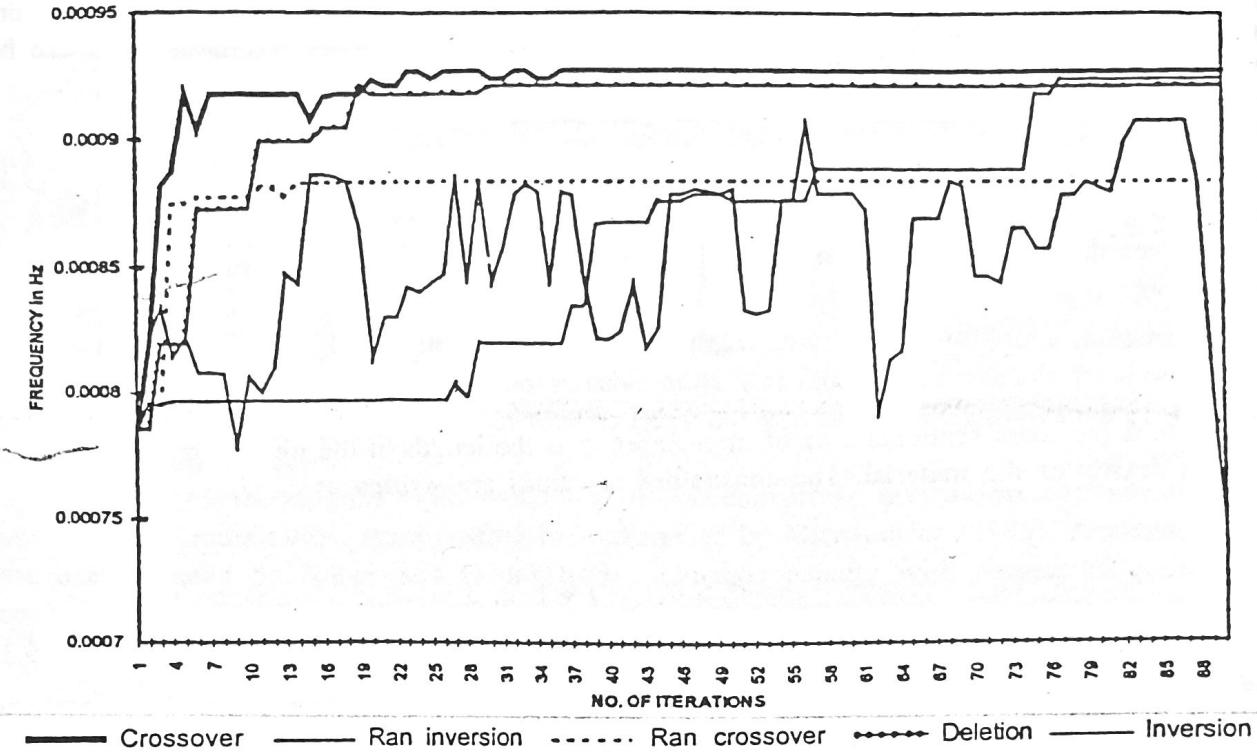
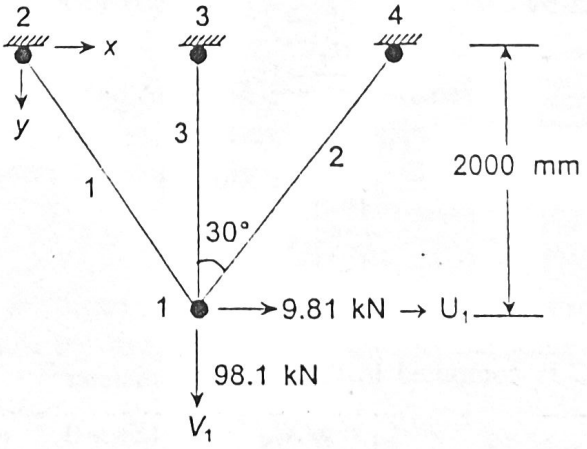


Fig. 9.17 Convergence history (frequency versus no. of iterations).

9.9.2 Constrained Optimization

Application from structural engineering

Consider a three bar truss shown in Fig. 9.18. The data assumed is E = 200.8 GPa, Density ρ = 7850 kg/cu m (78.5 kN/cu m), maximum stress, σ_{max} = 147.15 MPa, and maximum displacement, u_{max} = 5 mm. The truss is symmetric.



E = 200.8 GPa

ρ = 78.5 kN/m^3

σ_{max} = ±147.15 MPa

u_{max} = ±5 mm

Fig. 9.18 Three bar truss.

It is necessary to find out the optimum cross-sectional areas of members 1, 2, and 3 for the given loading conditions. The mathematical programming formulation of this problem can be written as follows.

$$\text{minimize } f(X) \text{ subject to}$$
$$g_j(X) \le 0;\ j = 1, 2, \ldots, m \tag{9.1}$$

where m is the number of constraints.

Since the objective function is to minimise the weight $f(X)$, three bar truss problem can be written as

$$f(X) = \sum_{i=1}^{3} \rho A_i L_i \tag{9.2}$$

where A_i is the cross sectional area of ith member, L_i is the length of the ith member, and ρ is the weight density of the material. The constrained equations are written as

$$g_i(X)$$
$$\sigma_j \le \sigma_a \qquad j = 1, 2, 3$$
$$u_1 \le u_a; \quad v_1 \le u_a \tag{9.3}$$

where σ_j is the stress in jth member and σ_a is the allowable stress, u_1 and v_1 are the horizontal and vertical displacements of node 1 and u_a is the allowable displacement.

Here, all the constraints cannot be directly described in terms of design variables hence, they are implicit and their evaluation requires analyzing a truss. Assume, for this purpose, a program is available for analyzing the truss to give the stresses in various members as well as displacements at the nodes. We have seen that GAs are ideally suited to unconstrained optimization problems. As the present problem is a constrained optimization problem, it is necessary to transform it to an unconstrained optimization problem to be able to solve it using GA. Transformation methods achieve this either by using exterior or interior penalty functions. Such transformations are ideally suited for sequential searches. GA performs the search in parallel using populations of points in search space. Hence, traditional transformations using penalty or barrier functions are not appropriate for genetic algorithm. A formulation based on the violation of normalized constraints is generally adopted. It is found to work very well for the class of problems. The constraint in normalized form is given by

$$\frac{\sigma_j}{\sigma_a} - 1 \le 0$$
$$\left|\frac{u_1}{u_a}\right| - 1 \le 0$$
$$\left|\frac{v_1}{v_a}\right| - 1 \le 0 \tag{9.4}$$

A violation coefficient C is computed in the following manner

$$C_i = g_i(X), \quad \text{if} \quad g(X) > 0$$
$$C_i = 0, \quad \text{if} \quad g(X) \le 0 \tag{9.5}$$

$$C = \sum_{j=1}^{m} C_j \tag{9.6}$$

where m is the number of constraints.

Now the modified objective function $\phi(X)$ is written as

$$\phi(X) = f(X)\ \{1 + KC\} \tag{9.7}$$

where parameter K has to be judiciously selected depending on the required influence of a violation individual in the next generation. A value of 10 was found to be suitable for most of the problems. Now the genetic algorithm is used to carry out unconstrained optimization of $\phi(X)$ as seen for two bar pendulum.

Similar to the approach discussed above for converting constrained optimization to unconstrained optimization, many approaches are studied by Michalewicz (1995). Amongst them, the one proposed by Joines and Houck uses a dynamic penalty with respect to generation count 't' as

$$\phi(X) = f(X) + (\gamma_0 \times t)^{\alpha} \sum_{j=1}^{m} g_j \beta(X) \tag{9.8}$$

In Eq. (9.8), g_j is the jth constraint function, which is zero in case of no violation, and is positive otherwise. The $(\gamma_0 \times t)^{\alpha}$ component of the penalty term is the penalty multiplier, whereas γ_0 stands for the penalty coefficient. For γ_0, α, and β the values of 0.5, 2, and 2 are recommended respectively (Erbatur et al., 2000).

Modifying the Eq. (9.8) as

$$\phi(X) = f(X) + (\gamma_0 \times t)^2 \sum_{j=1}^{m} g_j(K) \tag{9.9}$$

Hasancebi and Erbatur (1999) suggested for $\phi(X)$ as

$$\phi(X) = f(X) + Cp \sum_{j=1}^{m} g_j(X) K \tag{9.10}$$

where 'C' is the control parameter and 'p' is the penalty parameter.

Let us use the approach given by Rajeev and Krishnamoorthy (1992). Going back to truss shown in Fig. 9.18, the objective function is taken as

$$\phi(X) = f(X)\ (1 + 10C) \tag{9.11}$$

Because the design variables are discrete it is necessary to supply a list of values that the design variables can take. The available sections assumed for assigning the value for design variables are given in the list S as

S = {1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2, 3.4, 3.6, 3.8, 4.0, 4.4} sq.cm which are given in Table 9.8.

Table 9.8 The available sections for truss members

S.no.	Bit	Decoded value	Area	S.no.	Bit	Decoded value	Area
1	0000	0	1.2	9	1000	8	2.8
2	0001	1	1.4	10	1001	9	3.0
3	0010	2	1.6	11	1010	10	3.2
4	0011	3	1.8	12	1011	11	3.4
5	0100	4	2.0	13	1100	12	3.6
6	0101	5	2.2	14	1101	13	3.8
7	0110	6	2.4	15	1110	14	4.0
8	0111	7	2.6	16	1111	15	4.4

It is to be noted that the areas are not incremented uniformly. From 1 to 15, increment is 0.2 and from 15 to 16, the increment is 0.4. If the increment is uniform we can write as

$$X_{\text{inc}} = \frac{(X_u - X_L)}{(2^n - 1)} \tag{9.12}$$

where 'n' is the number of digits and

$$X_i = X_L + (\text{Decoded value}) \times X_{\text{inc}} \tag{9.13}$$

A four-bit string can define sixteen different values. The design variables are the areas of two members. Since the truss is symmetric, the area of third member is the same as the first member and hence, there are only two design variables for the optimization problem corresponding to two groups. A binary string of length four is capable of representing 16 different values, and since there are two groups, the total length of the string becomes eight with two substrings of length four each as shown in Fig. 9.19.

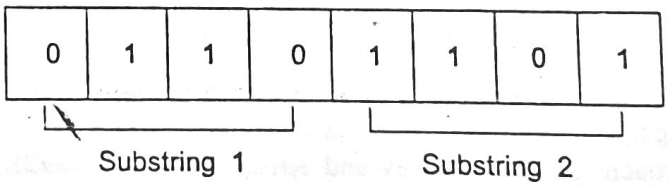


Fig. 9.19 Individual string of length 8.

In this problem the fitness function can be calculated as

$$F_i = [\phi(X)_{\max} + \phi(X)_{\min}] - \phi_i(X) \tag{9.14}$$

The procedure is exactly same as two bar pendulum except that GA will call *analysis program* for the analysis of three bar truss to get the stresses in all the members and the displacements. The details of computations for three generations are given in Tables 9.9, 9.10, and 9.11, respectively.

Table 9.9 Details of computations—Generation 1

S.no. 1	Population 2	A_1 3	A_2 4	$f(X)$ 5	σ_1 6	σ_2 7	σ_3 8	u_1 9	u_2 10	c 11	$\phi(x)$ 12	F 13	$F/\bar{F}$ 14	Count 15	Mating pool 16	Mate 17	CS1 18	CS2 19	Pop. gen. 2 20
1	0111 0111	2.6	2.6	13.50	160.8	85.3	164.1	0.87	−1.63	.208	41.6	88.9	1.15	1	0111 0111	3	4	8	01111111
2	1001 1111	3.0	4.4	17.78	121.4	55.97	118.2	0.75	−1.18	.000	17.7	112.7	1.46	2	1001 1111	7	2	4	10101111
3	0100 0111	2.0	2.6	11.33	190.6	92.48	188.67	1.13	−1.88	.578	76.8	53.61	0.69	1	1001 1111	1	4	8	10010111
4	1111 0011	4.4	1.8	18.78	120.2	75.58	130.48	0.51	−1.30	.000	18.8	111.74	1.45	1	0011 1101	8	1	4	00110111
5	0011 0110	1.8	2.4	10.29	209.8	100.8	206.99	1.25	−2.06	.833	96.0	34.48	0.45	0	1111 0011	6	2	6	11101011
6	0110 0000	2.4	1.2	10.59	211.29	129.5	227.13	0.09	−2.26	.979	114.	16.31	0.31	0	1010 1001	5	2	6	10110011
7	1010 1001	3.2	3.0	16.31	133.5	72.14	137.03	0.71	−1.36	.000	16.3	114.21	1.48	2	1010 1001	2	2	4	10011001
8	0011 1101	1.8	3.8	12.49	174.4	65.35	159.8	1.25	−1.59	.271	46.4	84.165	1.09	1	0011 1101	4	1	4	01001101

Table 9.10 Details of computations—Generation 2

S.no. 1	Population 2	A_1 3	A_2 4	$f(X)$ 5	σ_1 6	σ_2 7	σ_3 8	u_1 9	u_2 10	c 11	$\phi(x)$ 12	F 13	$F/\bar{F}$ 14	Count 15	Mating pool 16	Mate 17	CS1 18	CS2 19	Pop. gen. 2 20
1	0111 1111	2.6	4.4	16.33	143.3	56.9	126.1	0.87	−1.26	.000	16.3	87.6	1.18	1	01111111	4	2	5	01011111
2	1010 1111	3.2	4.4	18.51	116.1	55.32	114.6	0.71	−1.14	.000	18.5	85.4	1.15	1	10101111	8	4	8	10001111
3	1001 0111	3.0	2.6	14.96	146	80.53	150.9	0.75	−1.5	.026	18.8	85.1	1.15	1	10010111	5	4	8	10011011
4	0011 1011	1.8	2.6	10.61	203.5	94.48	198.6	1.25	−1.98	.733	88.4	15.6	0.21	0	10011001	1	2	5	10111001
5	1110 1011	4.0	3.4	19.84	110.1	61.06	114.1	0.56	−1.14	.000	19.8	84.1	1.13	1	11101011	3	4	8	11100111
6	1011 0011	3.4	1.8	15.15	147.2	89.48	157.8	0.66	−1.57	.073	26.1	77.8	1.05	1	10110011	7	2	7	10011001
7	1001 1001	3.0	3.0	15.59	139.4	73.97	142.2	0.75	−1.42	.000	15.6	88.4	1.19	2	10011001	6	2	7	10110011
8	0100 1101	2.0	3.8	13.21	164.1	65.94	153.	1.13	−1.53	.157	34.0	70.0	0.94	1	01001101	2	2	4	01101101

Table 9.11 Details of computations—Generation 2

S.no. 1	A_1 2	A_2 3	$f(X)$ 4	σ_1 5	σ_2 6	σ_3 7	u_1 8	u_2 9	c 10	$\phi(x)$ 11
1	2.2	4.4	14.88	146	56.8	135.1	1.03	−1.35	0.0	14.88
2	2.8	4.4	17.1	126.6	56.5	122.0	0.81	−1.22	0.0	17.06
3	3.0	3.4	16.21	133.6	68.1	134.5	0.75	−1.34	0.0	16.21
4	3.4	3.0	17.03	128.1	70.3	132.2	0.66	−1.32	0.0	17.03
5	4.0	2.6	18.58	118.9	69.8	125.8	0.56	−1.25	0.0	18.58
6	3.0	3.0	15.58	139.4	73.9	142.2	0.75	−1.42	0.0	15.58
7	3.4	1.8	15.15	147.2	89.5	157.8	0.66	−1.57	0.072	26.13
8	2.4	3.8	14.67	147.3	65.5	141.8	0.94	−1.41	0.0007	14.78

The flow chart of the genetic algorithm to solve the truss problem is shown in Fig. 9.20.

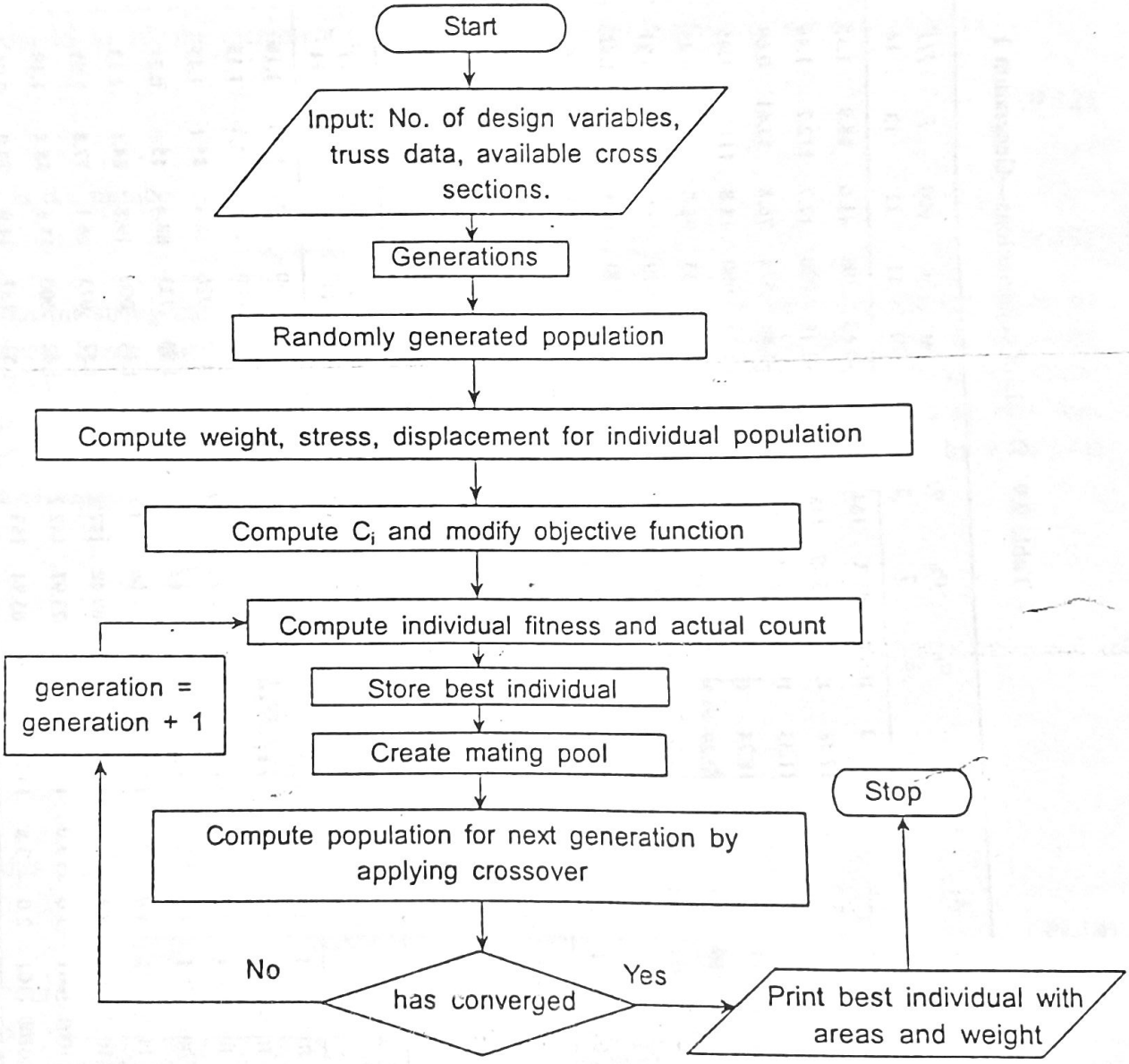


Fig. 9.20 Flowchart for truss optimization by GA.

$\bar{F}$ reduces from 53.48 in first generation to 17.53 in the third generation. From Table 9.11, we can accept $\phi(X)$ = 14.778 with slight violation, C = 0.00074 and hence the minimum weight of the truss is 14.667 kg (0.14 kN) with the area of inclined member as 2.4 sq cm and the vertical member as 3.8 sq cm.

9.10 MULTI-LEVEL OPTIMIZATION

A drawback of GA

Optimization of any system comprising numerous design variables is a challenging problem due to its huge size of search space. In fact, it is not easy for any optimization algorithm to carry out an effective exploration without locating local optimum. The dimensions of the search space grow exponentially either with the addition of extra design variables or with the enlargement of profile lists. The tests performed show that GA is generally quite successful in locating the regions of the search space containing the global optimum, but not the true optimum itself.

Here, multi-level optimization approach as given by Erbatur et al. (2000), is implemented and proved to eliminate the effect of such a major drawback. The approach rests on reducing the size of the search space for individual design variables in each successive level of the optimization process. In this approach an initial optimization, named the first level optimization, is carried out with an original profile list (discrete set) used by all the design variables. An original discrete set must be arranged in such a manner that the ready sections are placed in increasing order of the cross-sectional areas. In other words, the first entry of the discrete set is the smallest section and the last entry is the largest section. Following the first level of optimization, the algorithm automatically divides the original discrete set into several subsets (sub-profile lists with smaller sized search space, to be employed in the second level optimization).

The procedure used to create these subsets is as follows.

1. The best design obtained in the first level optimization is taken as the reference design.
2. The original discrete set is equally divided into prescribed number of subsets.
3. Individual design variables are directed to appropriate subsets according to the values obtained in reference design.
4. The enlargement of subset is performed.

Such a treatment of subsets avoids the risk of locating local optimum. In the second level optimization, the design variables use smaller sized subsets. Hence, the second level optimization is performed on more restricted regions of the search space. The process continues in a similar fashion by dividing the subsets into new subsets and directing the design variables to the most appropriate search space.

The characteristics of multi-level optimization are:

1. Firstly, it encourages the optimization process to investigate better solutions in more restricted favourable regions of the search space. Therefore, each optimization level may be interpreted as one step of climbing up a hill towards the summit. Also, it performs well in each search space which is occupied by closely placed optima.
2. Secondly, since the capacity of a discrete set is kept constant, any subset formed by dividing a discrete set must contain fewer ready sections than its capacity. That means, excess slots are produced in the subsets. For continuous optimization problems, these slots can be utilized to obtain better approximation to the continuous solution.

It is observed from experience that two to three levels of optimization are adequate for the convergence to true optimum for discrete and continuous optimizations respectively.

9.11 REAL LIFE PROBLEM

Figure 9.21 shows the 112 bar steel dome that has been previously discussed by Saka (1990) and Erbatur et al. (2000). When applying GA to 112 bar dome, it is not practical to find the areas of 112 members individually. Consider a 4-bit string in each unknown, each population will consist of 112 × 4 = 448 bits. Even during construction, the dome is assembled using 10 or 15 types of members instead of 112 types of members. In the present study, the members are linked into 10 groups and string length of 4 × 10 = 40 bits is assumed such that the length of the substring corresponding to each design variable is four. The design variable for the problem is to find the optimal area of these ten types of members so that we achieve minimum weight design subjected to the constraints. The design data for 112 bar dome is shown in Table 9.12.

Table 9.12 Loading and properties data set

Case no.	Joint no.	Force in *X* dirn	Force in *Y* dirn	Force in *Z* dirn
1	1	0	0	−500 kg (−5 kN)
	17, 23, 29, 35	0	0	−40 kg (−0.4 kN)
	16, 18, 22, 24	0	0	−120 kg (−1.2 kN)
	28, 30, 31, 32	0	0	−120 kg (−1.2 kN)
	Rest	0	0	−200 kg (−2 kN)
	Modulus of elasticity			210 GPa
	Displacement <			0.5 m in any direction
	Allowable tensile stress <			1650 kg/sq cm (165 MPa)
	Allowable compressive stress<			400 kg/sq cm (40 MPa)
	Density of steel			7850 kg/cu m (78.5 kN/cu m)

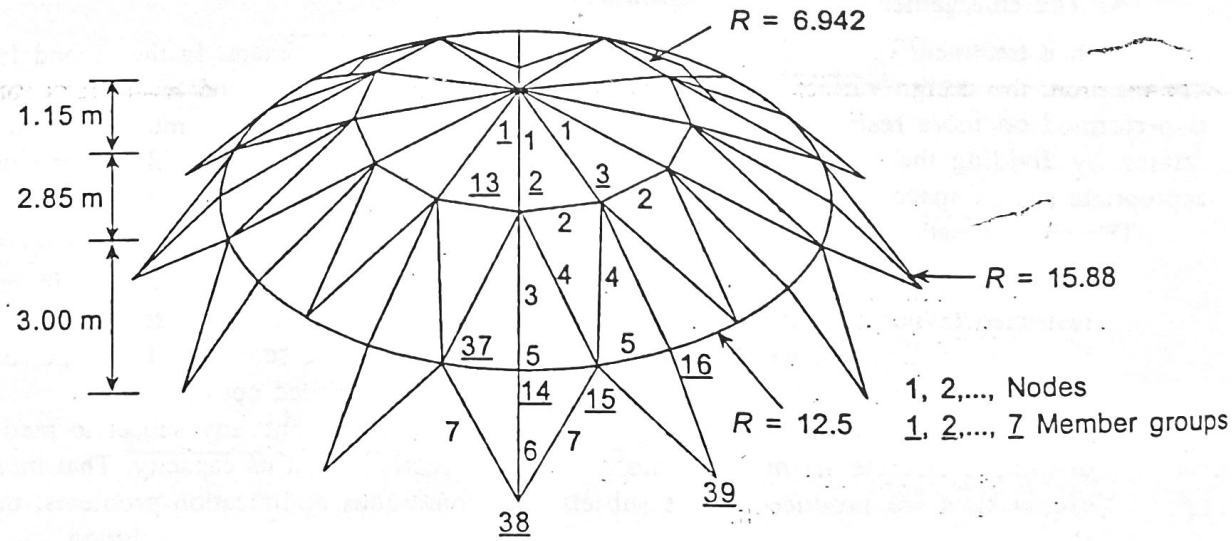


Fig. 9.21 Dimensions and member groups of the 112 bar dome.

The objective function is to minimize the weight of the dome and is given by

$$f = \rho \sum_{i=1}^{i=112} A_i L_i \tag{9.15}$$

Constraints for tensile stress are given as

$$\begin{cases} C_{i1} = 0 & \text{if} \quad \dfrac{\sigma_{iT}}{\sigma_{\text{all(ten)}}} - 1 \le 0 \\ C_{i1} = \dfrac{\sigma_{iT}}{\sigma_{\text{all(ten)}}} - 1 & \text{if} > 0 \end{cases} \tag{9.16}$$

Constraints for compressive stress are given as

$$\begin{cases} C_{i2} = 0 & \text{if} \quad \dfrac{\sigma_{iC}}{\sigma_{\text{all(com)}}} - 1 \le 0 \\ C_{i2} = \dfrac{\sigma_{iC}}{\sigma_{\text{all(com)}}} - 1 & \text{if} > 0 \end{cases} \tag{9.17}$$

Constraints for displacements in X, Y and Z directions are

$$\begin{cases} C_{i3} = 0 & \text{if} \quad \left|\dfrac{u_i}{u_{\text{all}}}\right| - 1 \le 0 \\ C_{i3} = \left|\dfrac{u_i}{u_{\text{all}}}\right| - 1 & \text{if} > 0 \end{cases} \tag{9.18}$$

Hence, the violation coefficient is given as

$$C = \sum_{i=1}^{112} \sum_{j=1}^{2} C_{ij} + \sum_{i=1}^{49} C_{i3} \tag{9.19}$$

The objective function for constrained problem is

$$\phi(X) = \phi(1 + KC) \tag{9.20}$$

and the value of K can be taken as 10.

Along with genetic algorithm program, the analysis program FEAST (FEAST, 1997) is combined to analyze the 112 bar dome to get the displacements of various nodes and stresses in various members.

Since we do not have any idea of the areas of members, the lowest areas of all groups of members may be assumed to be 400 sq mm and the upper bound for the areas of all groups of members may be assumed to be 1300 sq mm. After two generations, we get areas for the ten groups and the objective function is given as 3850 (i.e. the weight of the dome = 0.3850 kg (38.5 kN)). The areas of ten groups are given in Table 9.13. In the second level optimization, we assume the lower bound for areas of first six groups to be 500 sq mm and for the next three groups to be 900 sq mm, and the upper bounds for the above two as 800 sq mm, and 1300 sq. mm respectively. The objective function is given as 3390 kg (33.90 kN) and the areas for the ten groups are shown in Table 9.13. In the third level sub-optimization, the lower bound and the upper bounds for areas of ten groups are shown in Table 9.13. Since we are narrowing down

Table 9.13 Multi-level optimization (112 bar dome)

Level	Details	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Weight in kg (kN)
I	min	400	400	400	400	400	400	400	400	400	400	
	max	1300	1300	1300	1300	1300	1300	1300	1300	1300	1300	
First itn	obtained	1010	913	1130	1130	1350	400	1060	1130	913	1060	4881 kg (48.81 kN)
Second itn	obtained	693	1130	1130	1130	1350	1210	840	913	1350	620	3850 kg (38.5 kN)
	min	500	500	500	500	500	500	900	900	900	900	
	max	800	800	800	800	800	800	1300	1300	1300	1300	
	obtained	640	640	780	600	520	540	1030	980	1100	1060	3390 kg (33.90 kN)
	min	600	600	700	580	500	500	1000	450	1000	1000	
	max	700	700	800	650	650	650	1100	1100	1100	1100	
	obtained	600	625	720	580	540	580	1030	970	1050	1000	3300 kg (33.0 kN)
Other	Saka (1990)	714	778	–	–	–	–	–	–	–	–	3468 kg (34.68 kN)
investigators	Erbatur (2000)	707	557	667	707	707	523	1120	–	–	–	3390 kg (33.90 kN)

the genetic space, we are able to direct the path towards the minimum. The obtained areas are also shown in the table. Similarly, further level sub-optimization is performed. The minimum weight of the 112 bar dome is obtained as 3300 kg (33 kN) and the corresponding areas are also shown in Table 9.13. The values obtained by present analysis are compared with earlier investigators.

It is seen from the real life problem that GA does not need any gradient or either supplementary problem information. This, in fact makes the optimization process easy with respect to more conventional techniques and explains why GA is applied to a wide range of problems. We can also conclude that GA is one of the most robust and promising strategies among discrete optimization techniques and the multi-level optimization approach makes it possible for the GA to compete with continuous optimization techniques.

9.12 DIFFERENCES AND SIMILARITIES BETWEEN GA AND OTHER TRADITIONAL METHODS

As seen from the working principle of GA, GAs are radically different from most of the traditional optimization methods. Genetic algorithms work with a string coding of variables instead of the variables. The advantages of working with a coding of variable is that coding discretizes the search space even though the function may be continuous. On the other hand, since GA requires only function values at discrete points, a discrete or discontinuous function can be handled with no extra cost. This allows GA to be applied to a wide variety of problems. Genetic algorithm operators exploit the similarities in string structure to make an effective search. Genetic algorithm works with a population of points instead of a single point. In GA, previously found good information is emphasized using reproduction operator and propagated adaptively through cross over and mutation operators. Genetic algorithm is a population based search algorithm and multiple optimal solutions can be captured as shown in Fig. 9.22, thereby reducing the effort to use the algorithm many times.

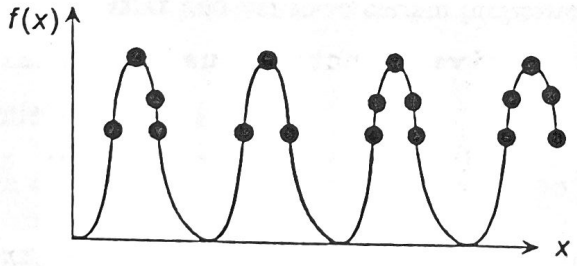


Fig. 9.22 Multi-modal functions.

Even though GAs, are different than most traditional search algorithms, there are some similarities. In traditional search methods, where a search direction is used to find a new point, at least two points are either implicitly or explicitly used to define the search direction. In the cross over operator, (which is mainly responsible for GA search) two points are used to create new points. Thus, cross over operator is similar to a directional search method with an exception 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. Since two points used in cross over operator are chosen at random, many search directions are possible. Among them, some may lead to global basin and some may not. The reproduction operator has an indirect effect of filtering the good search direction and helps to guide the search. The purpose of mutation operator is to create a point in the vicinity of the current point. The search in the mutation operator is similar to a local search method such as exploratory search used in Hooke-Jeeves method.

9.13 ADVANCES IN GA

Recently, new and efficient cross over operators have been designed so that search along variables is also possible. Let us consider $X_i^{(j)}$, $X_i^{(k)}$ values of design variables X_i in two-parent string j and k. The cross over between these two values may produce the new value as

$$X_i^{new} = (1 - \lambda)X_i^{(j)} + \lambda X_i^{(k)} \tag{9.21}$$

Here, the parameter λ is a random number between 0 and 1.

The above equation calculates new value bracketting the above two-parent values. This calculation is performed for each variable in the string. This cross over has uniform probability of creating a point inside the region bounded by two parents. An extension of the cross over to create points outside the range is bounded by the parents.

GA with memory

A local improvement procedure for GA is described in this section. The binary tree data structure given by Kernighan and Ritchie (1988) provides a way to store previous designs which permit efficient search of duplicate designs. The binary tree structure is used to store all the data pertinent to the design of the encoded design string. Figure 9.23 shows the pseudo code for the calculation of the objective function using binary tree. After a new generation of design strings is created, the binary tree is searched for the new design. If the design is found, the objective function value is obtained from the tree without analysis, otherwise the tree is searched for design with identical parameters. New designs and their relevant data are then inserted into the binary tree. This algorithm is given in Fig. 9.23.

Algorithm

```
Evaluation of objective function using binary tree.
begin
    Search for the given design in binary tree
    if found;
        Get objective function value from tree;
    else
        Search for the design having identical parameters
    if found
        Get normalised values from the tree
    else
        Perform analysis
    end if
        Add design to binary tree
    end if
end
```

Fig. 9.23 Finding objective function using binary tree.

Multi-modal Optimization

Many real world problems contain multiple solutions that are optimal or near optimal. The knowledge of multiple optimal solutions in a problem provides flexibility in choosing alternate yet good solutions as and when required. The idea of a number of optimal solutions coexisting in a population requires some change in the simple genetic algorithm described in the previous section. In nature, for example, we recognize that multiple niches (humans and animals) exist by sharing available resources. A similar sharing concept is introduced artificially in GA population in sharing functions as given by Deb (1989) which calculate the extent of sharing that needs to be done between two strings. If d_{ij} is the distance between ith and jth string. Then

$$Sh(d_{ij}) = \begin{cases} 1 - d_{ij}/\sigma & \text{if} \quad d_{ij} < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{9.22}$$

Here, the parameter σ is the maximum distance between two strings for them to be shared and is fixed beforehand. *Shared fitness* (*Sh*) is calculated and this is used for reproduction. Using with this sharing strategy, GAs have solved a number of multi-modal optimization problems having more than five million local optima, of which 32 are global optima as given by Goldberg, et al. (1992).

Searching for optimal schedule

Job shop scheduling, time tabling, and travelling salesman problems are solved using GA. A solution in these problems is a permutation of N objects (names of machines or cities). Although reproduction operator is based on fitness function which is nothing but the distance travelled by salesman, the cross over and mutation operators are different. The operators are designed to produce offsprings which are valid and yet have certain properties of both parents as suggested by Goldberg (1989).

Non-stationary function optimization

Diploid and dominance concepts can be implemented in GA to solve non-stationary optimization problem. Information about earlier good solutions can be stored in recessive alleles and when needed, can be expressed by suitable genetic operators.

Multi-objective optimization

There are many objective functions in multi-objective optimization. The usual practice is to convert multiple objectives into one objective function as

$$\phi = W_1 f_1 + W_2 f_2 + \ldots + W_n f_n \tag{9.23}$$

where W_1, W_2 are the weights and $f_1, f_2, \ldots, f_n$ are multi-objective functions. Equation (9.23) is one way of converting multi-objective function into single-objective optimization problem. The solution of multi-objective optimization problem can be considered as a collection of optimal solutions obtained by solving different single-objective functions formed using different weight vectors. These solutions are known as *Paretooptimal solutions*. This can be solved using the concept of non-dominated sorting of population members. For details one may refer to Srinivas and Deb (1995).

Issues for GA practitioners

The following issues are important while applying GA to practical problems, namely

1. Choosing basic implementation issues such as
   (a) Representation,
   (b) Population size and mutation rate,
   (c) Selection, deletion policies, and
   (d) Cross over and mutation operators.
2. Termination criterion
3. Performance and scalability
4. Solution is only as good as the evaluation functions (often a difficult task).

Benefits of GA

The concept of genetic algorithms is

1. easy to understand,
2. modular, separate from application,
3. supports multi-objective optimization,
4. good for noisy environment,
5. we always get an answer and the answer gets better with time,
6. inherently parallel and easily distributed,
7. there are many ways to speed up and improve a GA's basic applications as knowledge about the problem domain is general,
8. easy to exploit for previous or alternate solutions,
9. flexible in forming building blocks for hybrid applications, and
10. has substantial history and range of use.

When to use GA

Genetic algorithm should be used in case,

1. alternate solutions are too slow or overly complicated,
2. need an exploratory tool to examine new approaches,
3. problem is similar to one that has already been successfully solved by using GA,
4. we want to hybridize with an existing solution, or,
5. benefits of GA technology meet key problem requirements.

Table 9.14 gives the fields where GA is successfully applied.

Table 9.14 GA applications

Domains	Application types
Control	Gas pipe line, pole balancing, missile evasion, pursuit
Design	Semi conductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	Manufacturing, facility scheduling, resource allocation
Robotics	Trajectory planning
Machine learning	Designing neural networks, classification algorithms
Signal processing	Filter design
Game playing	Poker, checker, prisoner's dilemma
Combinatorial optimization	Set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

Research directions

During the course of a run, the optima value for each operator probability may vary. When the population converges, the cross over rate is reduced to give more opportunity to mutation to find new variations. In the past, much research has been empirical, but gradually theoretical insights are being gained. In many cases, it is still too early to say which techniques are robust or general purpose and which are special purpose. The most promising ideas are steady-state replacement, fitness ranking, two-point cross over and are often good methods to use. Knowledge-based operators and dynamic probabilities are help to solve real world problems. If the fitness function has many local maxima, no search technique is ever going to perform will on it. Better methods for designing fitness functions are needed which can avoid such pitfalls. There is no doubt that research into GA will be a very active area for some time to come. Genetic algorithms can perform multi-modal optimization technique which is beyond the scope of this book. Besides, a plethora of other applications including non-stationary function, optimization job scheduling problems, routing problem, travelling salesman problems have been tried. The growing list of successful applications of GA to different engineering problems confirms robustness of GAs and shows promise of GA in solving engineering design problems. According to Goldberg "Genetic Algorithms are rich-rich in applications across a large and growing number of disciplines".

SUMMARY

In this chapter,

- The cross over and mutation operators are discussed.
- Bit-wise operators are illustrated.
- Examples of unconstrained optimization are solved.
- GA to constrained optimization is described.
- GA is applied to real life problems and the results are compared.
- Advances in GA are also illustrated.

PROGRAMMING ASSIGNMENT

P9.1 Solve the nonlinear optimization problem
Minimize

$$(X_1 - 1.5)^2 + (X_2 - 4)^2$$

subject to

$$4.5X_1 + X_2^2 - 18 \leq 0$$

$$2X_1 - X_2 - 1 \geq 0$$

$$0 \leq X_1, X_2 \geq 4$$

Show calculations for three generations. Use cross over probability as 80% and a mutation probability of 3%.

P9.2 Use the Program "GAOPT" given in CD-ROM to solve the Problem 9.1.

(a) Use 4 bits for each variable.

(b) Use 5 bits for each variable. Use multilevel optimization technique.

P9.3 Minimize the function

$$f(X_1, X_2) = (X_1^2 + X_2 - 1)^2 + (X_1 + X_2^2 - 7)$$

in the interval $0 \leq X_1, X_2 \leq 6$. Use bit-wise operators and show atleast two generations.

P9.4 Use the program "GAOPT" given in CD-ROM to solve the Problem P9.3.

P9.5 Modify the program by including a program to solve a truss and hence find the optimal design for the truss shown in Fig. 9.18.

SUGGESTED FURTHER READING

Deb, K. (1999), *An Introduction to Genetic Algorithms*, Sadhana, Vol. 24, Parts 4 and 5, Aug. and Oct., pp. 293–315.

Gen, M. and R. Cheng (1997), *Genetic Algorithm and Engineering Design*, John Wiley, New York.

Goldberg, D.E. and J. Richardson (1987), Genetic Algorithms with Sharing for Multi-modal Function Optimization, *Proc. of the Second Intl. Conf. On Genetic Algorithms*, (Ed.) J.J. Greffenstette, pp. 41–49.

Starkweather, T., S. McDaniel, K. Mathias, D. Whitley, and C. Whitley (1991), A Comparison of Genetic Scheduling Operators, *Proc. of Fourth Intl. Conf. on Genetic Algorithms*, (Eds.) R. Belew, L.B. Booker, (San Mateo, C.A., Morgan Kaufmann), pp. 69–76.

SOME USEFUL WEBSITES

1. http://cs.felk.cvut.cz/~xobitko/ga
2. http://www.elec.gla.ac.uk/~yuunli/ga_demo
3. http://www.eng.buffalo.edu/Research/MODEL/wcsmo3/proceedings/numlist.html
4. http://ls11.www.informatik.uni-dortmund./de/evenet/coordinator/resources/softwareanddemos.html
5. http://lancet.mit.edu/~mbwall/presentations/Introto GA

REFERENCES

Annonymous (1997), *FEAST-C User Manual,* SEG, SDS group, V.S.S.C, ISRO, Trivandrum.

Deb, K. (1989), Genetic Algorithms in Multi-modal Function Optimization, *Masters thesis* (TCGA report No. 89002), Tuscaloosa, University of Alabama.

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

Erbatur, F., O. Hasancebi, I. Tutuncu, and H. Kilic (2000), Optimal Design of Planar and Space Structures with Genetic Algorithms, *Computers and Structures*, Vol. 75, pp. 209–224.

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

Goldberg, D.E., K. Deb, and J. Horn (1992), Massive Multi-modality, Deception, and Genetic Algorithms in Parallel Problem Solving from Nature, (Eds.) R. Manner, B. Manderich, Berlin Springer Verlag, pp. 37–46.

Gottfried, B.S. (1990), *Programming with C*, Schaum's outline series, Tata McGraw-Hill, New Delhi, 1990.

Hasancebi, O. and F. Erbatur (1999), Constraint Handling in Genetic Algorithm Integrated Structural Optimization", *Acta Mechanica.*

Kernighan, B.W. and D.M. Ritchie (1988), *The C Programming Language,* 2nd ed., Prentice Hall, Englewood Cliffs, N.J., USA., pp. 139–143.

Michalewicz, Z. (1995), Genetic Algorithms, Numerical Optimization and Constraints, *Technical Report*, University of North Carolina, USA.

Rajeev, S. and C.S. Krishnamoorthy (1992), Discrete Optimization of Structures using Genetic Algorithms, *Joul. of Structural Engg*, ASCE, Vol. 118, No. 5, pp. 1223–1250.

Saka, M.P. (1990), Optimum Design of Pin-jointed Steel Structures with Practical Applications, *Joul. of Structural Engineering*, ASCE, Vol. 116, No. 10, pp. 2599–2620.

Srinivas, N. and K. Deb (1995), Multi-objective Function Optimization using Non-dominated Sorting Genetic Algorithm, *Evolutionary Computation*, Vol. 2, pp. 221–225.