CHAPTER 5

Unit V

Genetic Algorithm

Syllabus Topic : Basic Concepts

5.1 Basic Concepts

5.1.1 What are Genetic Algorithms?

- Genetic Algorithms (GAs) were introduced in the United States in 1970s by **John Holland** at the University of Michigan.
- Genetic Algorithms are ***adaptive heuristic search*** algorithms based on the evolutionary ideas of natural selection and genetics.
- Evolution by natural selection is based on the principles of survival of the fittest by Charles Darwin.
- A GA can efficiently explore a large space of candidate designs and find optimum solutions.

5.1.2 Why Genetic Algorithms?

- Genetic algorithms are basically based on the Darwinian's "survival of fittest" concept of natural selection and evolution to produce the solution for a given problem.
- Genetic algorithms are one of the best ways to solve a problem for which little information is known, especially, where the search space is very vast and non-linear.
- GAs are best suited for an environment in which there is a very large set of candidate solutions and in which the search space is uneven and has many hills and valleys.

5.1.3 Biological Background : The Cell, Chromosomes, Genetics, Reproduction, Neural Selection

- All living organisms are made up of **cells**.
- Each cell comprises of a set of **chromosomes** which are strings of DNA.
- A chromosome is made up of several **genes**.

- Each gene encodes a particular trait, e.g. Colour of eyes is determined by some gene.
- **Alleles** are possible forms of the same gene. For e.g., eye colour can be brown, bluish- brown, black etc.
- The position of a gene on a chromosome is called **locus**.
- The complete set of genetic material is called a **genome**.
- A particular set of genome is called a **genotype**.
- The **phenotypes** are characteristics such as eye colour, hair texture etc. which are determined by the genotypes.
- During reproduction, first **recombination** (or **crossover**) occurs. Genes from parents form in some way the whole new chromosome.
- The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents.

Syllabus Topic : Working Principle

5.2 Working Principle

- To solve any problem, a genetic algorithm begins with a set of **solutions** (represented by **chromosomes**) called the **population.**
- Solutions from a particular population are taken and used to form a new population. The new population describes what we call the **next generation**. It is hoped that the next generation will be better than the previous one.
- Solutions are selected according to their **fitness** to form new solutions. Fitness is the ability of a solution to survive and pass on to the next generation. The more their fitness, the higher is their chance to reproduce.
- The next generation (their offspring) are created by exchanging individual genes of parents selected and also by changing genes randomly in individual chromosomes.
- This is repeated until some condition (E.g. number of generations or improvement in the best solution) is satisfied.

Syllabus Topic : Procedures of GA, Flow Chart of GA

5.3 Procedures of GA, Flow chart of GA

Step 1 : Initialize a population with randomly generated individuals and evaluate the fitness value of each individual.

Step 2 :

(a) Select two individuals from the population with probability proportional to their respective fitness values.

(b) Apply crossover on the two selected individuals (in step(a)) with a probability equal to the crossover rate.

(c) Apply mutation with a probability equal to the mutation rate.

(d) Repeat (a) to (d) until enough members are generated to form the next generation.

Step 3 : Repeat steps 2 and 3 until the stopping criterion is met.

Flowchart of GA is depicted in Fig. 5.3.1.

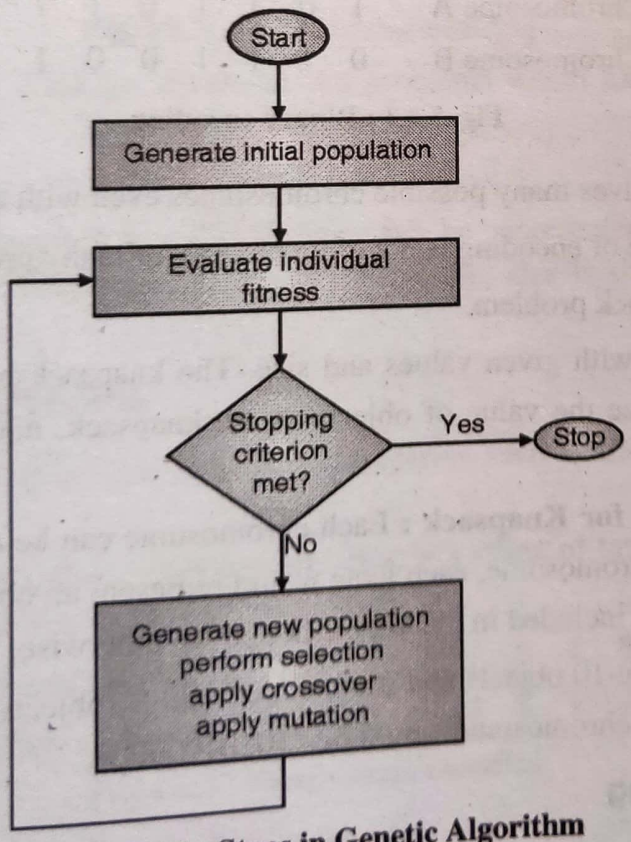


Fig. 5.3.1 : Steps in Genetic Algorithm

Syllabus Topic : Genetic Representations (Encoding)

5.4 Genetic Representations (Encoding)

- Genetic algorithms need design space to be converted into genetic space. Thus genetic algorithms work with an encoding of variables.
- The advantage of working with an encoding of variable space is that encoding makes the search space discrete even though the function may be continuous.
- There are different ways in which individual genes can be encoded. Depending on the problem to be solved, we can choose one of the encoding schemes.

5.4.1 Binary Encoding

- In binary encoding, each chromosome is a string of 0s and 1s, i.e. each gene in a chromosome is either 0 or 1.
- This technique is the most commonly used technique in encoding a chromosome. Fig. 5.4.1 shows two chromosomes which use binary encoding.

Chromosome A	1	0	1	1	0	1	1	0
Chromosome B	0	1	1	1	0	0	1	1

Fig. 5.4.1 : Binary encoding

- Binary encoding gives many possible chromosomes even with a small number of alleles.
- However, this type of encoding is not often natural for many problems.
- **Example :** Knapsack problem.
- There are objects with given values and size. The knapsack has a given capacity. The goal is to maximize the value of objects in the knapsack, not extending the knapsack capacity.
- **Encoding scheme for Knapsack :** Each chromosome can be a possible solution to the problem. In this chromosome, each gene would represent an object and will take a value of '1' if the item is included in the knapsack and '0' otherwise.
- For e.g., if there are 10 objects and if the 1^{st}, 3^{rd} and 5^{th} objects are to be included in the knapsack, then the chromosome would be "1010100000".

5.4.2 Octal Encoding

- In this type of encoding, the string is made up of octal numbers (0 to 7).
- If integers are to be represented using octal encoding, the encoded chromosomes will be smaller than those encoded with binary encoding.

Chromosome A	2	0	3	4	6
Chromosome B	1	2	6	7	0

Fig. 5.4.2 : Octal Encoding

- **Example :** Maximizing a function.
- Given a function of 2 or more variables, we need to maximize it. The variables are integers which fall within a given range.
- **Encoding scheme**

If the function is, say, a single variable function and the range is from 0 to 4095, then we can have chromosomes from (00000000) to (77777777) to represent them.

5.4.3 Hexadecimal Encoding

- This encoding uses strings made up of hexadecimal numbers (0 – 9, A – F)
- The advantage of this encoding scheme over binary and octal encoding is its smaller size.

Chromosome A	A	0	9	B
Chromosome B	9	3	2	F

Fig. 5.4.3 : Hexadecimal encoding

- **Example :** Can be used for maximization problems which have variables in a larger range.
- **Encoding scheme :**

If the range is between 0 and 65535, they can be represented easily by an 8 - digit hexadecimal code (0000 0000) to (FFFF FFFF). So, size is shorter than with octal or binary encoding.

5.4.4 Permutation Encoding

- This type of encoding is used in ordering problems such as travelling salesman or task ordering.
- In a permutation encoding, every chromosome is a string of integer / real values, which represents a number in a sequence.

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

Fig. 5.4.4 : Permutation encoding

- **Example :** Traveling salesman problem

There are 'n' cities and given distances between them. The travelling salesman has to visit each city exactly once. Finding the sequence of cities which minimize the traveling distance is the goal.

- **Encoding scheme :**

  Chromosome for this problem would depict the ordering of cities to be visited and be of length 'n'. For example, if there are 8 cities, a chromosome may be 3 4 5 1 7 8 6 2 (say). Another chromosome can be 2 3 4 7 8 1 5 6 and so on.

5.4.5 Value Encoding

- In this type of eneoding, each chromosome is a string of some values.
- Values can be anything connected to the problem like integers, real numbers, characters, objects etc.

Chromosome A	1.321	5.426	2.319	0.465
Chromosome B		ABDJEIFDT		
Chromosome C	(back)	(right)	(left)	(forward)

Fig. 5.4.5 : Different types of value encoding

- **Example :** Finding weights of neural networks.

  The goal is to find the weights of synapses connecting input to hidden layer and hidden layer to output layer.

- **Encoding scheme :**

  Each gene in the chromosome is a real value representing a particular weight.

5.4.6 Tree Encoding

- This technique is used mainly for evolving program expressions for genetic programming.
- Here, every chromosome is a tree of some objects such as functions and commands in a programming language.
- Tree encoding is shown in Fig. 5.4.6.

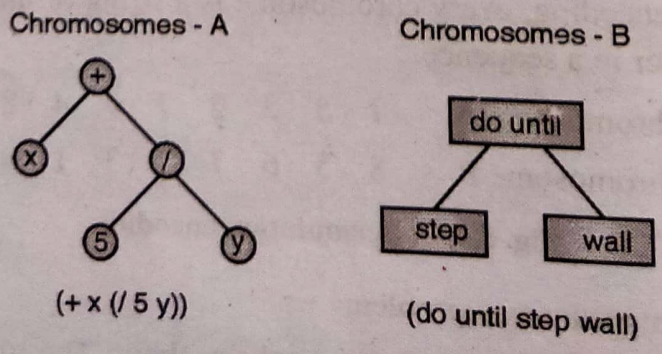


Fig. 5.4.6 : Tree Encoding

- Tree encoding is good for evolving programs in a programming language.
- **Example :** Find the function from input to output values.
- Some input and output values are given. The task is to find the function which will give the best relation to satisfy all values.
- **Encoding scheme :**
- Each chromosome will be a function represented as a tree.

Syllabus Topic : Initialization and Selection

5.5 Initialization and Selection

- Selection is the process of selecting two or more parent chromosomes from a given generation of population for mating.
- After deciding on an encoding scheme, the next step is to decide how to perform selection i.e. choosing chromosomes in the current generation that will create offspring for the next generation.
- In general, selection strategies must favour fitter candidates over weaker candidates.
- The most commonly used strategies are :

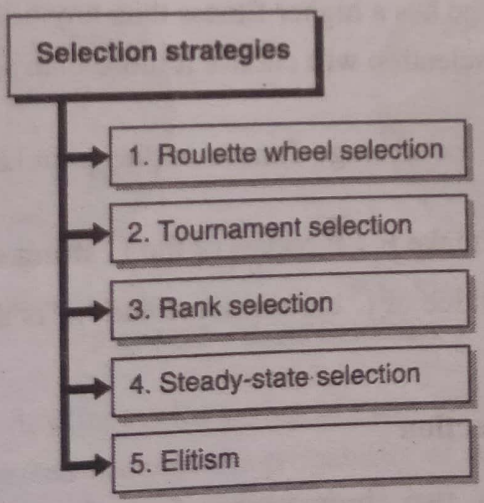


Fig. 5.5.1

1. Roulette Wheel Selection

- In this selection scheme, parents are selected according to their fitness values.
- The higher the fitness value of a chromosome, the more is the chance that it will be selected.

- Conceptually, each chromosome of the population is allocated a section of an imaginary roulette wheel.
- Unlike a real roulette wheel, the sections are of different sizes, proportional to the chromosome's fitness, such that the fittest candidate has the biggest slice of the wheel and the weakest chromosome has the smallest.
- The wheel is then spun and the chromosome associated with the winning section is selected. This process is repeated as many times as necessary to select the entire set of parents for the next generation.

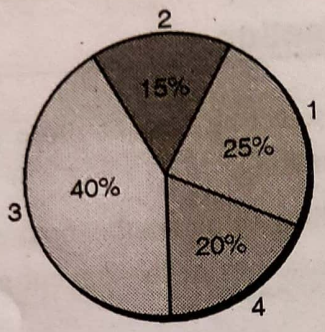


Fig. 5.5.2 : Roulette Wheel Selection

- Fig. 5.5.2 shows the Roulette - Wheel for 4 chromosomes according to their fitness. Since the 3rd chromosome has a higher fitness than any other chromosome, it is expected that the roulette-wheel selection will choose it more than any other chromosome.
- If $\left(\bar{F} = \sum_{j=1}^{n} F_j / n \right)$ is the average fitness of the population, then the Roulette-wheel selection is expected to make $F_j / \bar{F}$ copies of the j^{th} string in the population.

Where F_j is the fitness value of j^{th} chromosome and 'n' is the number of chromosomes in a chosen population.

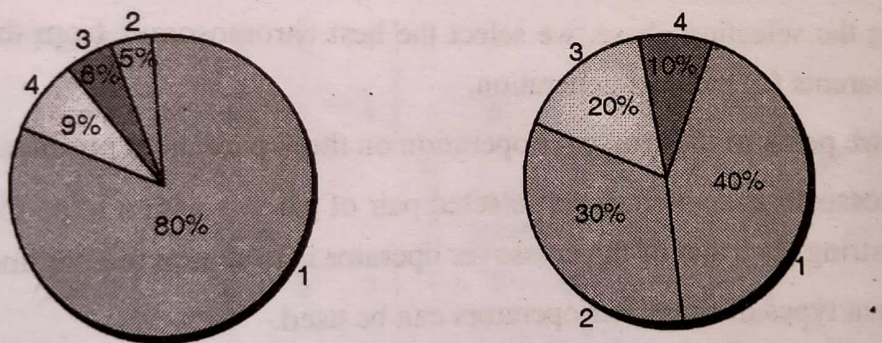
→ 2. Tournament Selection

- In tournament selection, two chromosomes from the population are randomly picked and a tournament is staged to determine which one gets selected.
- The winner of the "tournament" may be selected directly as the chromosome having the higher fitness among the two or it may be selected probabilistically.
- The probabilistic tournament selection involves generating a random value between 0 and 1 and comparing it to a pre-determined selection probability.

- If the random value is less than or equal to the selection probability, the fitter candidate is selected, otherwise the weaker candidate is selected.
- The tournament can be extended to involve more than two individuals if desired.

3. Rank Selection

- Rank selection attempts to remove problems of roulette wheel selection by basing selection probabilities on relative rather than absolute fitness.
- The problem with roulette wheel selection is that when fitness values differ very much, then it is biased towards the chromosome having the highest fitness and other chromosomes have very few chances to be selected.
- Rank selection first ranks the population and then every chromosome receives fitness from this ranking.
- The worst chromosome will have fitness 1, 2nd worst will have fitness 2 and the best will have fitness 'n' (Number of chromosomes in the population).
- After the fitness has been allocated, the chromosomes are then selected using the same mechanism of roulette wheel selection.



(a) Situation before ranking **(b) Situation after ranking**

Fig. 5.5.3 : Rank Selection

- As shown in Fig. 5.5.3, after ranking step of rank selection, all chromosomes have a good chance to be selected because the probability of selection is as per relative rather than absolute fitness values.

4. Steady - State Selection

- Here, the goal is to allow a major part of the chromosomes to survive and pass to the next generation.
- As part of this strategy, in every generation, a few good (high fitness) chromosomes are selected for creating new offspring.

- Then, some bad (low fitness) chromosomes are removed and the new offspring are placed in their place.
- The rest of the population survives to the next generation.

➜ **5. Elitism**

- In elitism, the best (or a few best) chromosome(s) are copied directly to the next generation.
- The remaining chromosomes are then selected using one of the above strategies.
- Elitism ensures that good chromosomes are not lost and can rapidly increase the performance of GA.

Syllabus Topic : Genetic Operators

5.6 Genetic Operators

5.6.1 Crossover

- After the selection phase is over, we perform crossover operation.
- During the selection phase, we select the best chromosomes from the given population to be parents for the next generation.
- Now, we perform the crossover operation on these parents to produce offspring.
- The crossover is applied to the selected pair of parents with a hope that it would create a better string. The aim of the crossover operator is to search the parameter space.
- Different types of crossover operators can be used.

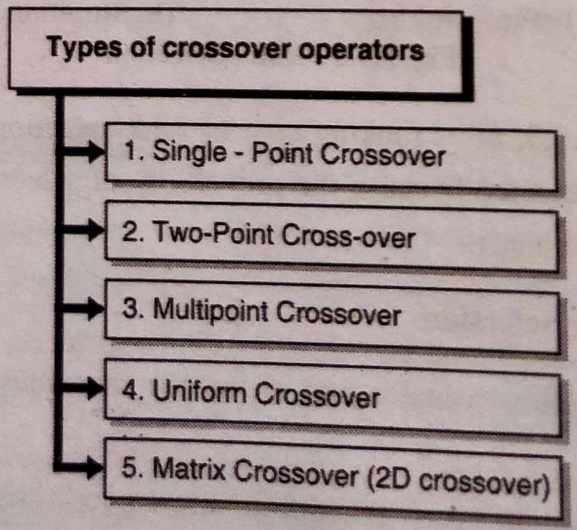


Fig. 5.6.1

1. Single - Point Crossover

Here, a crossover point on the strings to be mated is selected at random and bits next to the crossover point are exchanged. This is shown in Fig. 5.6.2.

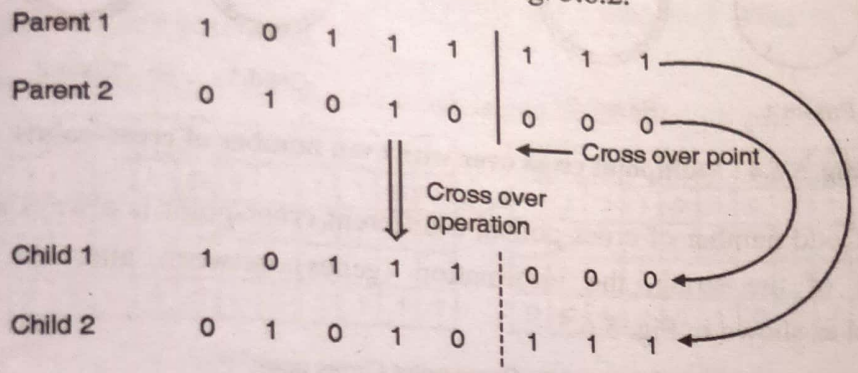


Fig. 5.6.2 : Single-point crossover

2. Two-Point Cross-over

In two point crossover, two crossover points are chosen at random and the parts of the chromosome strings between these two points is swapped. This is illustrated in Fig. 5.6.3.

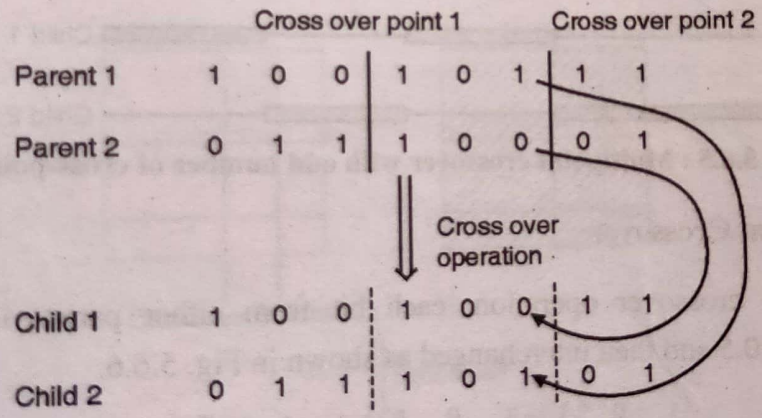


Fig. 5.6.3 : Two-point crossover

3. Multipoint Crossover

- Multipoint crossover uses more than two crossover points. There could be two cases again. We can have even number of crossover points or odd number of crossover points.
- In even numbered cross-points, the string is treated as a ring with no beginning or end.
- The cross-points are selected around the circle uniformly at random.
- Now, the information between alternate pairs of cross-points is interchanged as shown in Fig. 5.6.4.

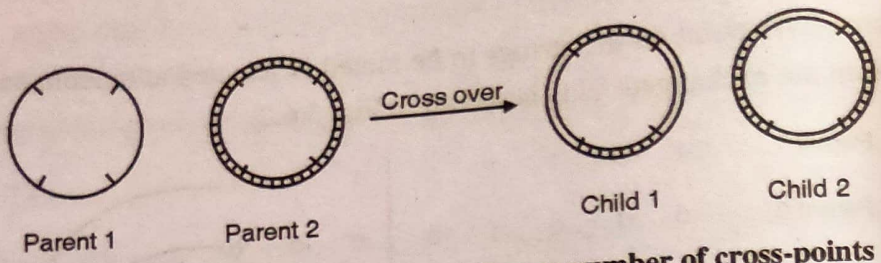


Fig. 5.6.4 : Multipoint cross over with even number of cross-points

- In case of odd number of cross-points, a different cross-point is always assumed at the beginning of the string, the information (genes) between alternate pairs is then exchanged as shown in Fig. 5.6.5.

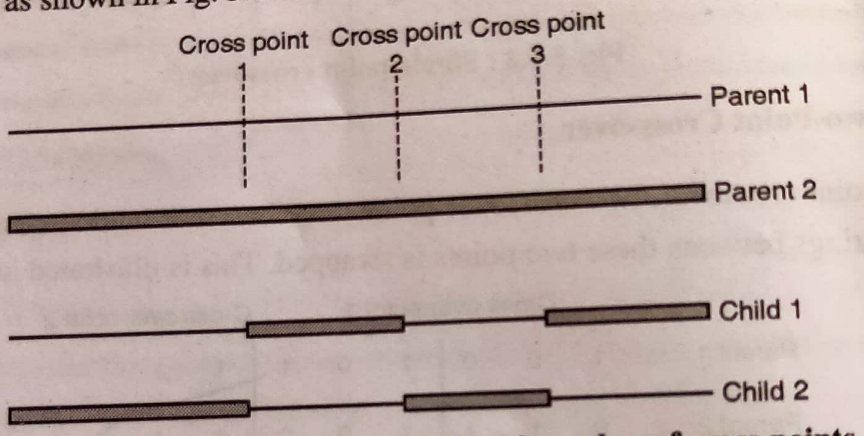


Fig. 5.6.5 : Multipoint crossover with odd number of cross-points

4. Uniform Crossover

- In a uniform crossover operation, each bit from either parent is selected with a probability of 0.5 and then interchanged as shown in Fig. 5.6.6.

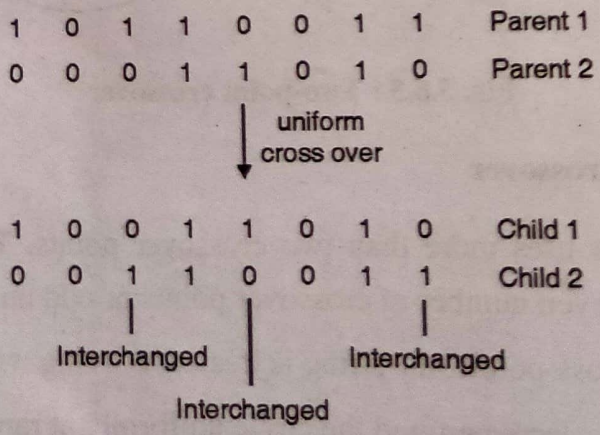


Fig. 5.6.6 : Uniform crossover

- Another variation of uniform crossover is to define a crossover mask. Whenever there is a 1 in the mask, the gene is copied from the first parent and when there is 0 in the mask, the gene from the second parent is copied to produce the first offspring.
- To produce the second offspring, the parents are exchanged. That is, parent 1 now becomes parent 2 and vice versa.
- This is illustrated in Fig. 5.6.7. A new mask is generated for each pair of parents.

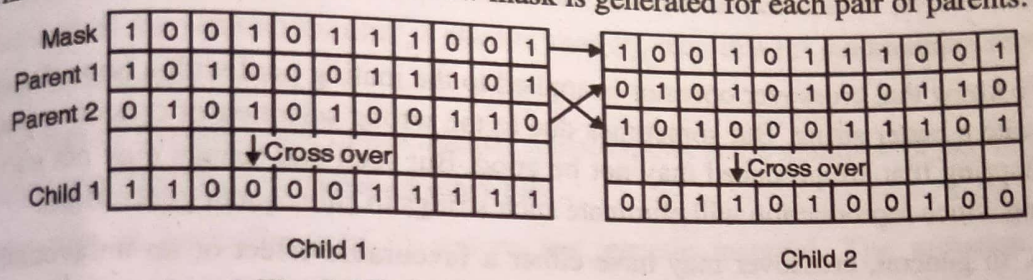


Fig. 5.6.7 : Uniform crossover using mask

5. Matrix Crossover (2D crossover)

- In case of matrix crossover, each individual is represented as a 2D array of vector. The process of 2D crossover is depicted in Fig. 5.6.8.

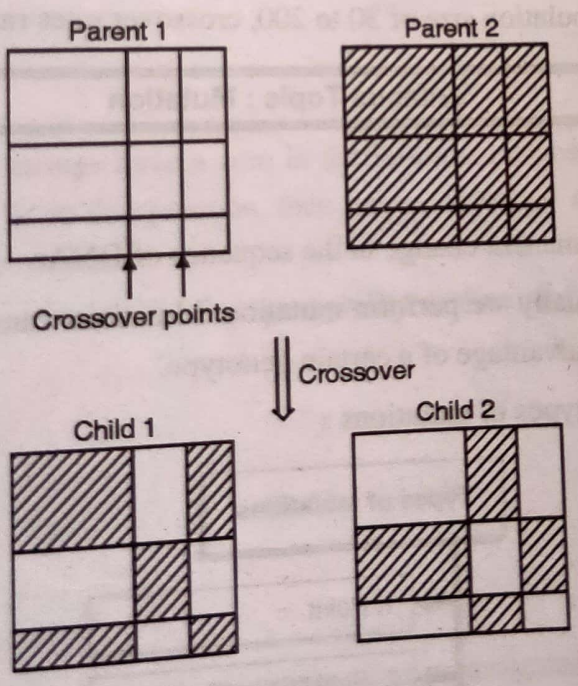


Fig. 5.6.8 : Matrix crossover

- Here two random crossover points along row and column are chosen, then string is divided into non overlapping rectangular regions.
- As shown in Fig. 5.6.8, two crossover points both row wise and column wise divide the rectangle into nine regions.

- Select any region in each layer, either vertically or horizontally and then exchange the information in that region between the mated populations.

☞ **Crossover Rate**

- The crossover rate is the probability of crossover denoted by P_c. The probability varies from 0 to 1. It is calculated as the ratio of the number of pairs to be crossed to some fixed population.
- We know that crossover operator is applied to the mating pool with a hope that it would create a better string. But sometimes due to the wrong selection of crossover points, the offspring that are produced may not be good. But such bad strings may not survive too long, since reproduction will eliminate such strings in subsequent generations.
- So, in general, crossover may have either a favourable effect or an unfavourable one. Thus, from among all the strings present in the mating pool, a few good strings are preserved and crossover is only performed on the remaining.
- Thus, crossover is performed according to a probability Pc. Only $(100 \times P_c)$ % strings are used for performing crossover and the remaining strings i.e. $(100 \times (1 - P_c))$ % remain as they are.
- Typically for a population size of 30 to 200, crossover rates range from 0.5 to 1.

Syllabus Topic : Mutation

5.6.2 Mutation

- A mutation is a permanent change in the sequence of DNA.
- After crossover, usually we perform mutation. Mutations can be advantageous and lead to an evolutionary advantage of a certain genotype.
- There are different types of mutations :

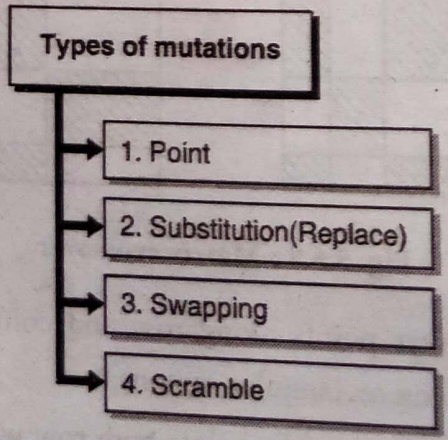


Fig. 5.6.9

1. Point Mutation

- In case of binary encoding, point mutation is a process of flipping a bit in a chromosome, that is, changing a bit from 0 to 1 and vice versa.
- Whether the particular bit is to be flipped or not, that is decided by flipping a coin with a probability of P_m.
- In coin flipping method, for a particular bit, a number between 0 and 1 is chosen at random. If the random number is smaller than P_m then that bit is flipped, otherwise it is kept unchanged.
- The bits of the string are independently mutated, that is the mutation of one bit does not affect probability of mutation of other bits.
- Mutation is basically used to restore lost 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 the local minima's trap.
- **Example :** Consider the following population having four eight – bit strings.

0110 1001
0011 1011
0001 1000
0111 1001

- Notice that all four strings have a zero in the leftmost bit position. If the true optimum solution requires a '1' in that position, then neither selection nor crossover operator can create a '1' in that position.
- The mutation flips each bit with some probability so the strings may now become,

0110 1001
0011 1011
0001 1000
1111 1001

2. Replace

- In this type of mutation, a single mutation point is generated randomly. Next, a gene is generated randomly. The gene at the mutation point is then replaced with the randomly generated gene to form the new chromosome.
- Consider the following chromosome.

1 2 3 4 5 6 7 8 9

- Assume that the random mutation point generated was 4. Further assume that the random gene that was generated to be "7". After replace mutation, the chromosome would be, 1 2 3 7 5 6 7 8 9.

3. Swapping

- In this type of mutation, first, two mutation points are selected randomly. The genes at these two points are then swapped to form the new chromosome.
- Consider the following chromosome.

  1 2 3 4 5 6 7 8 9
- Assume that the random mutation points generated were 2 and 5. So, the chromosome becomes,

  1 5 3 4 2 6 7 8 9

4. Scramble

- In this type of mutation, first, two mutation points are selected randomly. The genes between these two points are then shuffled in random order to form the new chromosome.
- Consider the following chromosome.

  1 2 3 4 5 6 7 8 9
- Assume that the random mutation points generated were 2 and 5. Assuming some random shuffling, the chromosome in after mutation would be,

  1 3 5 4 2 6 7 8 9

Mutation Rate

- Mutation rate is the probability of mutation.
- Typically for a population size of 30 to 200, mutation rate ranges between 0.001 and 0.5.

Other operators

1. Inversion

- Inversion is a kind of reordering technique. It operates on a single chromosome and inverts the order of the elements between two randomly chosen points on the chromosome.
- For example consider the following chromosome,

  0 1 1 0 0 1 0 1 1

Then after inversion, the new string is,

0 1 0 1 0 0 1 1 1

This is shown in Fig. 5.6.10.

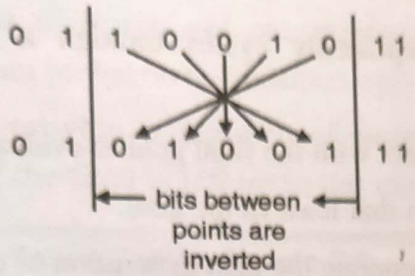


Fig. 5.6.10 : Inversion

2. Deletion

Usually deletion operator is performed with combination of other operators such as, duplication, regeneration, addition etc.

3. Deletion and Duplication

Here two or three bits are selected at random and the previous bits are duplicated.

This is shown in Fig. 5.6.11.

0	0	1	0	0	1	0	Before deletion
0	0	1	0	–	–	0	At deletion
0	0	1	0	1	0	0	Duplication

Fig. 5.6.11 : Deletion and Duplication

4. Deletion and Regeneration

Here bits between two cross-points are deleted and regenerated randomly. This is shown in Fig. 5.6.12.

1	1	0	1	0	1	1	0	
1	1	–	–	–	–	–	0	Deletion
1	1	1	0	1	0	1	0	Regeneration

Regenerated bits

Fig. 5.6.12 : Deletion and Regeneration

Syllabus Topic : Generational Cycle

5.7 Generational Cycle

- A genetic algorithm repeatedly cycles through a loop which defines successive generations of objects :
  1. Compare each object with the final goal to evaluate fitness.
  2. Discard the objects that least fit the goal.
  3. Use crossover to generate the next generation of objects.
  4. Simply copy some objects--as is--to the next generation.
  5. Add some objects with random mutations into the next generation.
- There is a common structure to most all genetic algorithms.
  - Initialize population with random individuals (candidate solutions)
  - Evaluate (compute fitness of) all individuals
  - WHILE not stop DO
  - Select genitors from parent population
  - Create offspring using variation operators on genitors
  - Evaluate newborn offspring
  - Replace some parents by some offspring.

Syllabus Topic : Traditional Algorithm vs. Genetic Algorithm, Differences and Similarities between GA and Other Traditional Method

5.8 Traditional Algorithm vs Genetic Algorithm

- GA has the following significant differences over the traditional search techniques.
  1. A GA works on coded versions of the problem parameters instead of the actual parameters themselves.
  2. Unlike other conventional methods that search from a single point, a GA always operates on a whole population of points (strings). This makes the GA more robust.
  3. Operating on the entire population also makes it possible to reach the global optimum and reduces the risk of becoming trapped in a local stationary point.

4. Usually genetic algorithms do not use any auxiliary information or about the objective function value such as derivatives. Hence, they can be applied to any kind of continuous or discrete optimization problems.
5. Conventional methods of search and optimization use deterministic transition operators while GA uses probabilistic transition operators.

- Genetic algorithms may provide a number of potential solutions to a given problem and the choice of the final is left up to the user.

Syllabus Topic : Simple GA

5.9 Simple GA

- A simple form of GA is given by the following steps :
  1. Initially we start with the randomly generated population
  2. Calculate the Fitness of each chromosome in the population.
  3. Repeat the following steps until n number of offspring are created.
- Select the pair of chromosome from the current population for mating
- Crossover the pair of parent chromosome from the current population with probability P_c.
- Perform the mutation with probability P_m.
- Replace the current population with the new population.
- Go to step 2
- Selection is done by comparing the fitness function of each chromosome in the current population.
- We need to define the appropriate fitness function.
- Each chromosome has an associated value corresponding to the fitness of the solution it represents.
- The optimize solution is the one which maximize the fitness function. If the problem is of minimization, then the fitness function can be transformed accordingly.
- GA evolves according to its basic structure. It starts by generating an initial population of chromosome. The initial population must be chosen so that a wide diversity of genetic materials is involved. Generally, the initial population is generated randomly.
- Then the GA gradually evolves by looping over an iteration process.

- Each iteration has following steps :

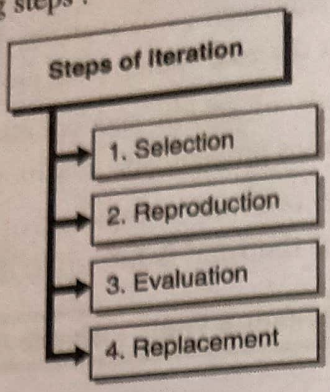


Fig. 5.9.1

➜ **1. Selection**

This step consists of selecting individuals for reproduction. Various selection methods are employed and depending upon the relative fitness of the individuals, the best ones are chosen for reproduction of new generation.

➜ **2. Reproduction**

This operation exchanges information between two potential parents and generates two offspring for the next population.

➜ **3. Evaluation**

The fitness of the new chromosomes is evaluated.

➜ **4. Replacement**

Individual from the old population are replaced by new ones.

The algorithm /*Simple GA*/

```
BEGIN
Generate initial population
Compute fitness of each individual
WHILE NOT stop
LOOP
  BEGIN
      Select individuals from current population for mating;
    Create offspring by applying recombination/mutation to the selected individuals;
     Compute fitness of the new individuals;
```

```
        Replace old individulas by new ones to generate new population
IF new population has converged
THEN stop = TRUE
END
END
```

Syllabus Topic : General Genetic Algorithm

5.10 General Genetic Algorithm

- The first step in applying GA to a problem is the encoding scheme.GA requires that the actual parameter space be converted into genetic space. In general, the GA evolves a multi-set of chromosomes.
- The chromosome is usually expressed as a string of variables, each element of which is called a gene. The variable can be represented as binary, real number, or other forms and its range is usually defined by the problem specified.

The steps involved in general GA are as follows.

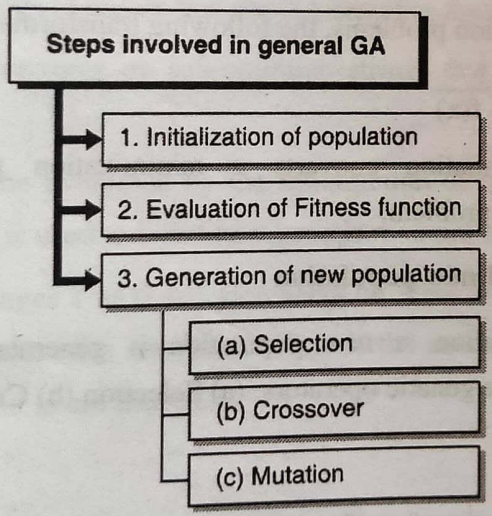


Fig. 5.10.1

1. Initialization of population

- GA works on the population of chromosomes. So, the first step is to select the initial population.
- There are two parameters that need to be decided for initial population: size of the population and the method of selecting initial population.

- In general, the size of 20 to 50 is preferable. There are two ways to generate the initial population, namely, the random initialization and heuristic initialization.

2. Evaluation of Fitness function

- A **fitness function** in a genetic algorithm is a function which is used to measure the "quality" or "fitness" of a given chromosome.
- Depending on the "fitness", chromosome either survives or dies in the process of evolution in a genetic algorithm. This is in sync with the Darwinian theory of survival of the fittest.
- If a particular chromosome has a higher fitness value, it is more likely that it will survive and pass on to the next generation. On the other hand, chromosomes with lower fitness values have very low chances of survival.
- GAs, when used to solve optimization problems, derive the fitness function from the objective function. In general, the fitness function is always problem dependent.
- For maximization problems, the fitness function 'F(x)' can be considered to be the same as the objective function 'f(x)' and hence F(x) = f(x).
- For minimization problems, the following transformation is often used.

$$F(x) = \frac{1}{1 + f(x)}$$

- This transformation converts a minimization problem to an equivalent maximization problem.

3. Generation of new population

After fitness evaluation, a new population is generated. This new population is generated using three genetic operators: (a) Selection (b) Crossover and (c) Mutation.

(a) Selection

- Selection is a mechanism for selecting individuals (strings) from a population according to their fitness (objective function value).
- The highest rank chromosome will have more possibility of selection and the worst will be eliminated.

(b) Crossover

- Once the selection process is over, the crossover operator is applied next. Crossover is a recombination operator that combines subparts of two parent chromosomes to produce offspring that contains some parts of both parents' genetic material.
- In the crossover, highly fit individuals are given opportunities to reproduce by exchanging pieces of their genetic information with other highly fit individuals.
- This produces new "offspring" solutions, which share some good characteristics taken from both parents. The various types of crossover methods have been discussed in section 5.6.

(c) Mutation

- The selection and crossover operators will generate a large amount of different off springs. However, there are two main problems with this.
  (1) Depending upon the initial population chosen, there may not be enough diversity in the initial strings to ensure that the GA searches the entire problem space and
  (2) The GA may converge on sub-optimum strings due to a bad choice of initial population.
- These problems may be overcome by the introduction of mutation operator into GA. The mutation operator is used to inject new genetic material into the genetic population.
- Mutation operator changes 1 as 0 and vice versa bit wise. Bitwise mutation is done bit by bit by flipping a coin with low probability. If the outcome is true, then the bit is altered; otherwise the bit is not altered.

5.10.1 Next Generation

If the new generation contains a solution that produces an output that is close enough or equal to the desired answer then the problem has been solved. Otherwise the new generation will go through the same process of fitness evaluation and reproduction.

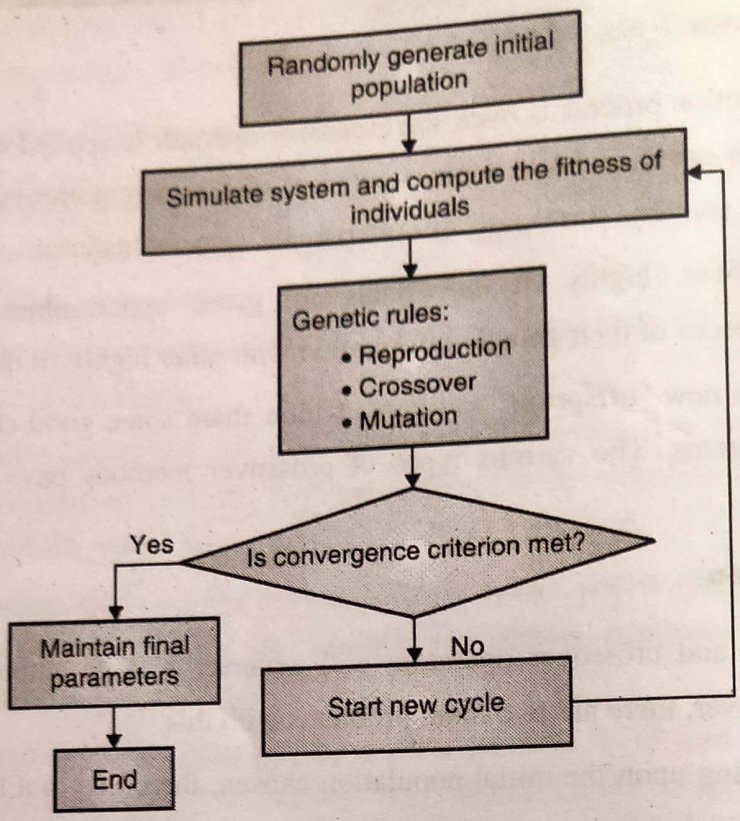


Fig: 5.10.2 : Flowchart of General GA

Syllabus Topic : Schema Theorem

5.11 Schema Theorem

Schema theorem serves as the analysis tool for the GA process.

- Applicable to a canonical GA
- binary representation
- fixed length individuals
- fitness proportional selection
- single point crossover
- gene-wise mutation

☞ Schema

A schema is a set of binary strings that match the template for schema H.

A template is made up of 1s, 0s, and *s where* is the 'don't care' symbol that matches either 0 or 1.

Schema Examples

The schema H = 10*1* represents the set of binary strings.

10010, 10011, 10110, 10111

The string '10' of length $l = 2$ belongs to $2^l = 2^2$ different schemas

**, *0, 1*, 10

Schema : Order o(*H*)

The order of a schema is the number of its fixed bits, i.e. the number of bits that are not '*' in the schema H

Example : if H = 10*1* then o(H) = 3

Defining Length δ (*H*)

The defining length is the distance between its first and the last fixed bits

Example : if H = *1*01 then $\delta(H) = 5 - 2 = 3$

Example : if H = 0**** then $\delta(H) = 1 - 1 = 0$

Count

Suppose x is an individual that belongs to the schema H, then we say that **x is an instance of H ($x \in H$).**

$m(H, k)$ denotes the number of instances of H in the k th generation.

Fitness

$f(x)$ denotes fitness value of x.

$f(H, k)$ denotes **average fitness of *H*** in the k-th generation.

$$f(H, k) = \frac{\sum_{x \in H} f(x)}{m(H, k)}$$

Effect of GA on a Schema

Effect of Selection = Effect of Crossover + Effect of Mutation= Schema Theorem

Effect of Selection on Schema

Assume a GA with proportional selection and arbitrary but fixed fitness.

Selection probability for the individual x *is given as,*

$$p_s(x) = \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}$$

Where, the N is the total number of individuals.

Net Effect of selection

The expected number of instances of H in the mating pool $M(H,k)$ is,

$$M(H, k) = \frac{\sum_{x \in H} f(x)}{\bar{f}} = m(H, k) \frac{f(H, k)}{\bar{f}}$$

"Schemas with fitness greater than the population average are likely to appear more in the next generation"

Effect of Crossover on Schema

Assume a single-point crossover.

Schema H survives crossover operation if,

- one of the parents is an instance of the schema H **AND**
- one of the offspring is an instance of the schema H

Crossover Survival examples

Consider H = *10**

$P_1 = 1\ 1\ 0 \mid 1\ 0 \in H$ $\quad \Rightarrow \quad$ $S_1 = 1\ 1\ 0\ 1\ 1 \in H$ Schema H

$P_2 = 1\ 0\ 1 \mid 1\ 1 \notin H$ $\quad\quad\quad$ $S_2 = 1\ 0\ 1\ 1\ 0 \notin H$ Survived

$P_1 = 1\ 1 \mid 0\ 1\ 0 \in H$ $\quad \Rightarrow \quad$ $S_1 = 1\ 1\ 1\ 1\ 1 \notin H$ Schema H

$P_2 = 1\ 0 \mid 1\ 1\ 1 \notin H$ $\quad\quad\quad$ $S_2 = 1\ 0\ 0\ 1\ 0 \notin H$ destroyed

Crossover Operation

- Suppose a parent is an instance of a schema H. When the crossover is occurred within the bits of the defining length, it is destroyed unless the other parent repairs the destroyed portion.
- Given a string with length l and a schema H with the defining length $\delta(H)$, the probability that the crossover occurs within the bits of the defining length is $\delta(H)/(l-1)$

Crossover Probability Example

Suppose H = *1**0

We gave,

$$l = 5$$

$$\delta(H) = 5 - 2 = 3$$

Thus, the probability that the crossover occurs within the defining length is ¾.

Crossover Operation

The upper bound of the probability that the schema *H* being destroyed is,

$$D_c(H) \leq p_c \frac{\delta(H)}{l-1}$$

Where, p_c is the crossover probability.

Net Effect of Crossover

The lower bound on the probability $S_c(H)$ that H survives is,

$$S_c(H) = 1 - D_c(H) \geq 1 - p_c \frac{\delta(H)}{l-1}$$

"Schemas with low order are more likely to survive"

Mutation Operation

Assumption : mutation is applied gene by gene.

For a schema *H* to survive, all fixed bits must remain unchanged.

Probability of a gene not being changed is $(1 - p_m)$.

Where , p_m is the mutation probability of a gene.

Net Effect of Mutation

The probability a schema *H* survives under Mutation

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

"Schemas with low order are more likely to survive"

Putting it all together,

Schema Theorem

Expected number of Schema H in next generation >=

$$E[m(H, k+1)] \geq m(H, k) \frac{f(H, k)}{\bar{f}} \left(1 - p_c \frac{\delta(H)}{l-1}\right) (1 - p_m)^{o(H)}$$

"The schema theorem states that the schema with *above average fitness*, *short defining length* and *lower order* is more likely to survive"

Syllabus Topic : Classification of GA

5.12 Classification of GA

Genetic algorithms can be broadly classified as sequential GA and parallel GA.

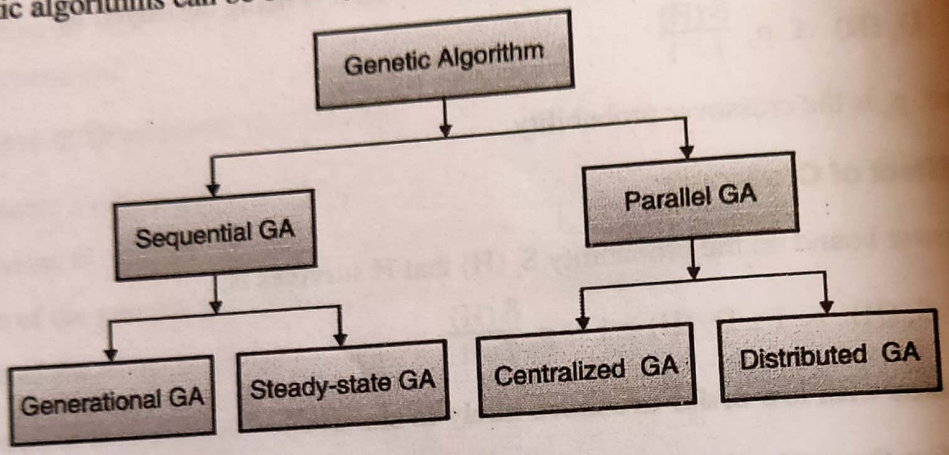


Fig. 5.12.1 : Classification of Genetic algorithms

Messy Genetic Algorithms

- In simple GA, genes are encoded as a fixed length strings. The meaning of a single gene is determined by its position inside the string.
- Messy GA uses variable - length, position-independent coding. Each gene has associated index with it that allows to identify its position. Thus a gene in Messy GA is no more represents the single allele value and a fixed position. It represents the pair of index and an allele. Fig 5.12.2 shows working of this Messy GA.

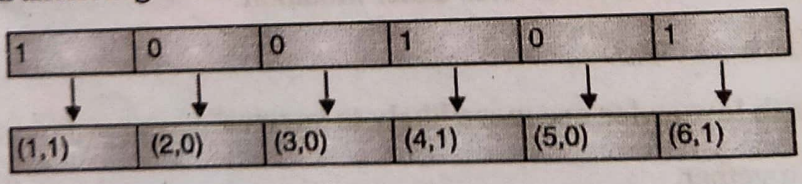


Fig. 5.12.2(A) : Messy Coding

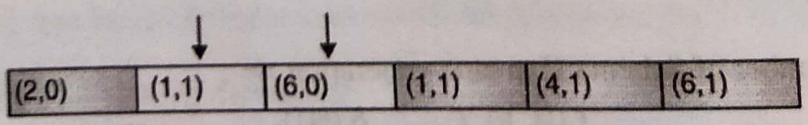


Fig. 5.12.2(B) : Positional Preference; Genes with indices 1 and 6 occur twice, the first occurrences are used

- Since it is possible to identify the genes uniquely with the help of the index, genes may be swapped randomly without changing the meaning of the string.
- The free arrangements of genes and the variable length encoding may pose some of the challenges in Messy GA.
- It can happen that there are two entries in a string, which corresponds to the same index, but have conflicting alleles. This situation is called 'over-specification'.
- The solution to this problem is to use positional preferences- The first entry, which refer to a gene, is taken.
- There could be a problem of 'under specification' meaning some of the genes may not occur in the encoded string at all. In Fig 5.12.2(B) the genes with index 3 and 5 do not occur at all in the example.
- One possible solution to this problem is to check all possible combinations and to take the best one.
- The mutation operator used in Messy GA is same as mutation operator used in simple GA, but crossover operator is replaced by a more general cut and splice operators. These operators allow to mate parents with different length.

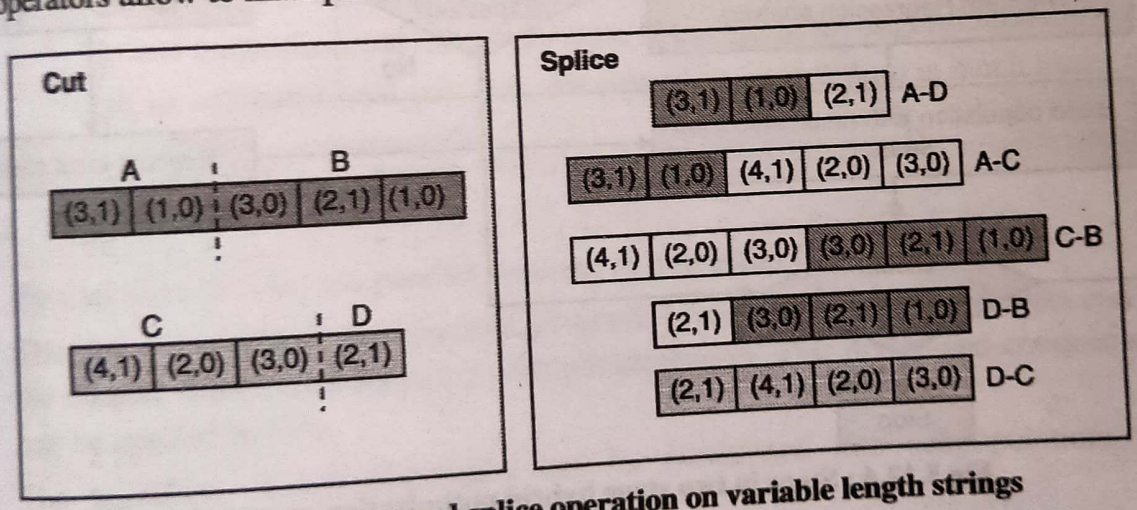


Fig 5.12.3 : The cut and splice operation on variable length strings

Adaptive Genetic Algorithms

In adaptive GA, the parameters such as population size, cross over probability, mutation probability etc. are varied at run time. One such example is, one can change the mutation rate according to the changes in the population. Longer the population does not improve, the higher the mutation rate is chosen.

Hybrid genetic algorithm

- Hybrid genetic algorithm combines the GA and conventional methods for optimization task.

- Here, the optimization task is divided into two parts.
- The GA does the course, global optimization and local refinement is done by the conventional methods (such as gradient descent, hill climbing, simulate annealing etc.).
- A number of variants are possible :
  1. GA performs coarse search first. After the GA completes, local refinement is done using conventional methods.
  2. The local method is integrated in GA itself.
  3. Both methods run in parallel.

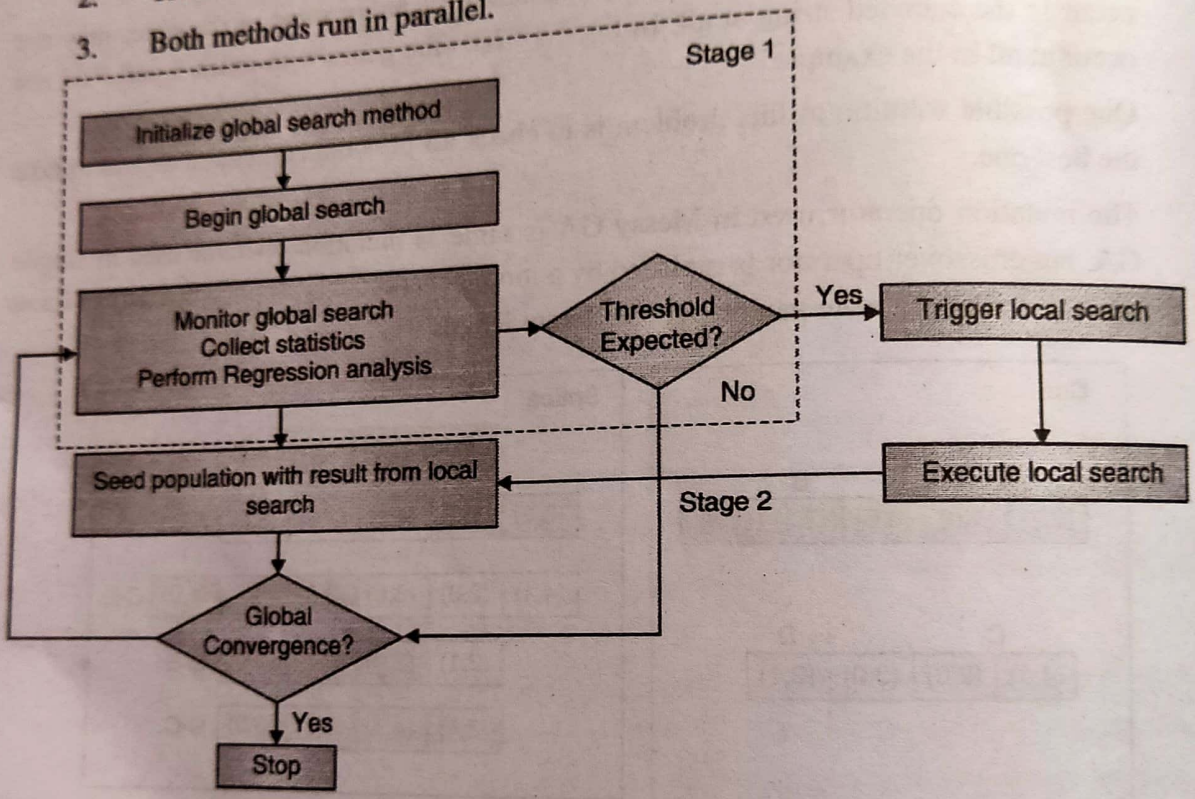


Fig 5.12.4 : Steps in two stage hybrid optimization approach

Fig 5.12.4 illustrates the stages in the algorithm.

- The algorithm switch repeatedly between global search (stage 1) and the local search (stage 2) during execution.
- In stage 1, the global search is initialized and monitored.
- In stage 2, the local search is executed when the threshold levels are expected. Then this solution is passed back to the global search.
- The algorithm is stopped when convergence is achieved for the global optimization algorithm.

5.12.1 Sequential GA

- In sequential genetic algorithms, we proceed in an iterative manner, where, in each iteration, we generate a new population of strings from the old ones.
- Every string is the encoded version of a tentative solution.
- Every string's suitability to the problem is evaluated based on its fitness measure.
- In order to compute a whole generation of new strings, the algorithm applies stochastic operators such as selection, crossover and mutation on an initial random population.
- In sequential GA there are again two variants: steady state GA and generational GA.
  - The steady state GA is a simpler version of the generational GA.
  - In steady state GA, the entire current population is not replaced, rather, two best parents are selected from the population and crossed obtaining two offspring that are then mutated and inserted in the current population.
  - On the other hand, in the generational version, a large portion of the population is selected and crossed (typically half the individuals), the resulting offspring is mutated and inserted into the population, thus replacing the old individuals.
  - Steady state GA is much simpler. In terms of computation, both versions are more or less equivalent, but the steady state GA requires more generations to converge, but its computational cost (i.e., the cost of every iteration) is much lower than the cost of the generational GA

5.12.2 Parallel GA

- Parallel GAs (PGAs) are parallel versions of sequential GAs.
- The basic idea behind most parallel programs is to divide a task into chunks and to solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be applied to GAs.
- The members of the population can be partitioned into subpopulations that are distributed among different processors.
- There are two classes of Parallel GAs: Centralised GA and distributed GA:
- Centralised GAs are also called **Global GA**. There is a single population (just as in simple GA), but the evaluation of fitness is distributed among several processors.
- Since, in this type of GA selection and crossover consider the entire population, it is also called global parallel GA.
- The algorithm can be executed on a shared memory multiprocessor, where the chromosomes are stored in the shared memory and each processor only develops certain chromosomes.

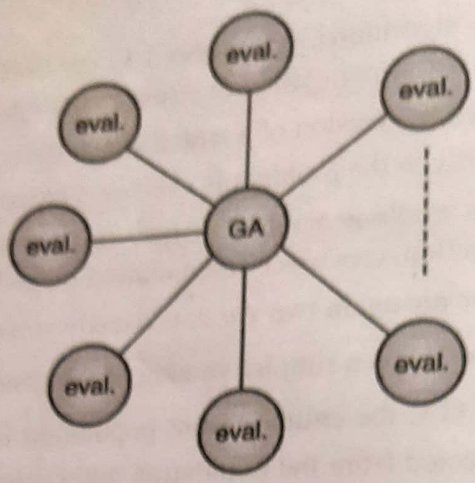


Fig. 5.12.5 : Schematic of centralised GA

- With distributed GA, each parallel component has its own copy of the reproduction. Distributed GAs can be further classified as :
  - Single - population Fine-grained
  - Multiple-population Coarse-grained.
- **Fine-grained parallel GAs** are suitable for massively parallel computers and consist of one spatially structured population. Each solution has a spatial location and interacts only with some neighbourhood, termed as *deme*.
- Demes are a special class of operators that indicate a pattern of neighbors for each location in the population structure.
- Selection and mating are restricted to small neighbourhood, but neighbourhood overlap permits some interaction among all the individuals.

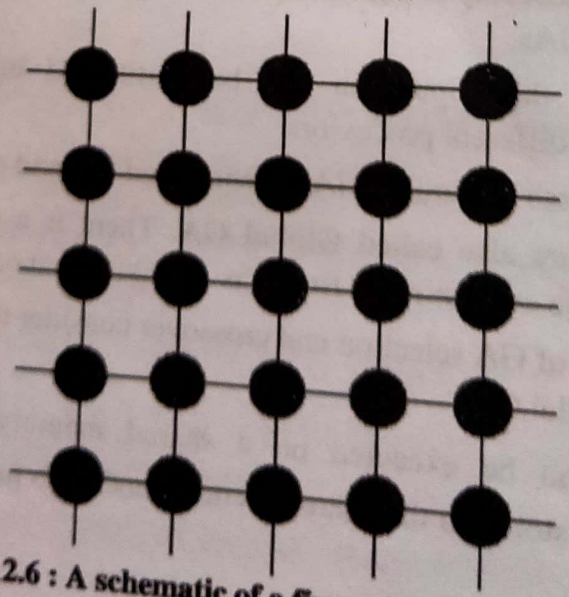


Fig. 5.12.6 : A schematic of a fine grained parallel GA

- In multiple population **coarse-grained GA,** genetic algorithm distributes the entire population to several sub-populations.
- Each sub-population is considered as a whole and separate offspring and cross-over (reproduction) area, which is controlled by its own genetic algorithm.
- In order to enable a good genetic material transfer from one sub-population into another and consequently a faster improvement of the entire population, we occasionally allow an individual chromosome migration between sub-populations.

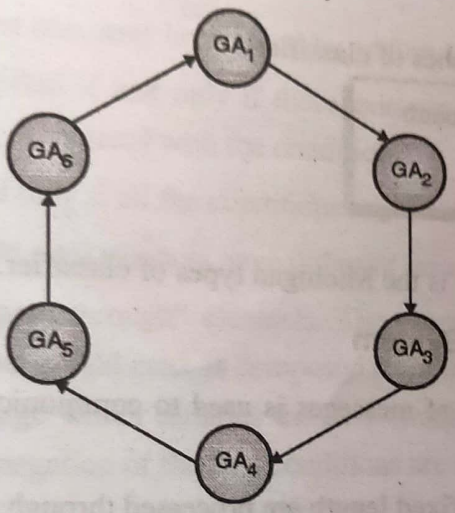


Fig. 5.12.7 : A schematic of multiple population parallel GA: Each process is a simple GA, and there is communication between the population

Advantages of using PGA

- Works on a coding of the problem (least restrictive)
- Basically independent of the problem (robustness)
- Can yield alternative solutions to the problem.
- Parallel search from multiple points in the space.
- Easy parallelization as islands or neighbourhoods.
- Better search, even if no parallel hardware is used.
- Higher efficiency and efficacy than sequential GAs.
- Easy cooperation with other search procedures.

Syllabus Topic : Holland Classifier System

5.13 Holland Classifier System

- A classifier system is a machine learning system that learns syntactically simple string rules, called classifiers.
- A classifier system is much more than a simple expert system that can learn from experience.
- There are two approaches of classifier

1. Michigan Approach
2. Pitt approach

- The Holland classifier is the Michigan types of classifier.

5.13.1 The Production System

- An arbitrary long list of messages is used to communicate the production system with the environment.
- Binary messages of a fixed length are processed through a rule base.
- The rules of the rule base are adapted according to response of the environment.
- The detectors detect the responses from the environment and translate them into binary messages. These translated messages are then placed onto the message list which is then scanned and changes by the rule base.
- Finally, the effectors translate output messages into actions on the environment, such as forces or movements.
- In Fig. 5.13.1, detectors, effectors and classifiers population blocks are part of the rule and message sub system.

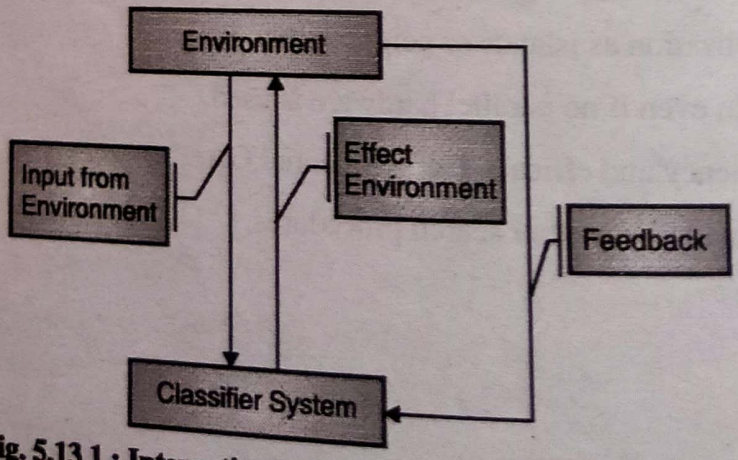


Fig. 5.13.1 : Interaction between classifier and environment

- Messages are binary strings of the same length k. That is, a message belongs to $\{0,1\}^k$.
- The rule base consists of a fixed number (m) of rules. Each rule consist of a fixed number (r) of conditions and actions.
- Both conditions and actions are strings of length k over the alphabet {0,1,*}.
- The * is a wild card., a don't care symbol.
- A condition is matched if and only if there is message in the list which matches the condition in all non- wild card positions.
- Condition, Except the first one, may be negated by adding '-' prefix.
- Such conditions are satisfied if and only if there exist no message in the message list which matches the string associated with the condition.
- Finally, a rule fires if and only if all the conditions are satisfied.
- In the action part, the wild card symbols have different meaning.

They take the role of "pass through" elements. The output messages of a firing rule, whose action part contains a wild card, is composed from the non - wildcard positions of the action and the message which satisfies the first condition of the classifier. This is actually the reason why negation of the first conditions are not allowed.

- More formally the outgoing message is defined as :

$$m = \begin{cases} a[i] & , \text{ if } a[i] \neq * \\ m[i] & , \text{ if } a[i] = * \end{cases} \qquad i = 1, \ldots k$$

Where ***a*** is the action part of the classifier and ***m*** is the message which matches the first condition.

Formally, a classifier is a string of the form,

$Condition_1$, l'-'l $Condtion_2$,.....l'-'$Condition_r$ /Action

- The messages that are not interpreted by the output interface may be preserved for the next step by using the tagging.
- Tagging allows transferring information about the current step into the next step simply by placing tagged messages on the list.
- The following steps depict the simple execution of the production system.

1. Messages from the environment are appended to the message list.
2. Set of firing rules are obtained by checking all the conditions of all classifiers against the message list.
3. The message list is deleted.

4. The firing classifiers compete to place their message on the list
5. The winning classifiers place their action on the list.
6. The messages directed to the effectors are executed.

The above steps are repeated iteratively.

5.13.2 The Bucket Brigade Algorithm

- Bucket brigade algorithm evaluates rules and decides among competing alternatives.
- It Use a Genetic Algorithm (GA) to generate new rules.
- A classifier's strength U is used as its fitness.
- In each time step t, we assign a strength value ($G_{i,t}$) to each classifier R_i.
- This strength value represents the importance of the classifier.
- The strength value of each classifier is adapted depending upon the feedback from the environment (called payoff).
- Each matching classifier computes a **'bid'** by multiplying its specificity (the number of non-don't cares in its condition part) with the product of its strength and a learning parameter.
- The bid of the classifier R_i at t is defined as :

$$Bi, t = C_L G_{i,t} S_i$$

Where , C_L is a learning parameter, S_i is the specificity, $G_{i,t}$ is the strength of the classifier.

- For small value of C_L, the system adapts slowly. If C_L high the strength tends to oscillate chaotically.
- Then the rules have to compete for the right for placing their output messages on the list. This can be done by random experiment like the selection in GA.
- For each biding classifier it is decided randomly if it wins or not, where the probability that it wins is proportional to its bid. In this approach, more than one classifiers can win.
- Another approach is **highest bidding** agent wins. This method resolves the conflict between two winning classifiers.
- **Payoff :** each classifier receives a portion of Payoff from the environment and accordingly the strength of the classifier is adapted.
- Let us denote the set of classifiers, which have supplied the winning agent R_i, in step t with $S_{i,t}$. Then the new strength of a winning agent is reduced by its bid and increased by its portion of the payoff P_t received from the environment.

$$U_{i,t} + 1 = U_{i,t} + P_t / W_t - Bi, t$$

- Where W_t is the number of winning agents in the actual time step. The winning agent pays its bid to the supplier which share the bid among each other equally.
- If the winning agent is also active in the previous step and supplies another winning agent, the value above is additionally increased by one portion of the bid the consumer offers.
- The strength of all other classifiers R_m, which are neither winning agents nor suppliers of winning agents are reduced by a certain factor.

$$U_{m,t+1} = U_{m,t}(1-T)$$

T is small value lying in the interval [0,1].

- Thus we punish that classifier which never contributes anything in the output of the system.
- The central idea is that classifiers which are not active when the environment gives payoff but which had an important role for setting the stage for directly rewarded classifiers can earn credit by participating in 'bucket brigade chains'.

Syllabus Topic : Genetic Programming

5.14 Genetic Programming

- Genetic programming typically starts with a population of randomly generated computer programs composed of the available programmatic ingredients.
- All programs in the initial population are syntactically valid and executable programs.
- Genetic programming iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations.
- These operations are applied to individual(s) selected from the population.
- The individuals are probabilistically selected to participate in the genetic operations based on their fitness.
- The iterative transformation of the population is executed inside the main generational loop of the run of genetic programming.
- The executional steps of genetic programming are as follows :

1. Randomly create an initial population (generation 0) of individual computer programs composed of the available functions and terminals.

2. Iteratively perform the following sub-steps (called a generation) on the population until the termination criterion is satisfied :
   (a) Execute each program in the population and ascertain its fitness (explicitly or implicitly) using the problem's fitness measure.
   (b) Select one or two individual program(s) from the population with a probability based on fitness (with reselection allowed) to participate in the genetic operations in (c).
   (c) Create new individual program(s) for the population by applying the following genetic operations with specified probabilities :
      (i) **Reproduction :** Copy the selected individual program to the new population.
      (ii) **Crossover :** Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.
      (iii) **Mutation :** Create one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program.
      (iv) **Architecture-altering operations:** Choose an architecture-altering operation from the available repertoire of such operations and create one new offspring program for the new population by applying the chosen architecture-altering operation to one selected program.
3. After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is harvested and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

- Genetic programming is problem-independent in the sense that the flowchart specifying the basic sequence of executional steps is not modified for each new run or each new problem.
- There is usually no discretionary human intervention or interaction during a run of genetic programming (although a human user may exercise judgment as to whether to terminate a run).
- Fig. 5.14.1 is a flowchart showing the executional steps of a run of genetic programming. The flowchart shows the genetic operations of crossover, reproduction, and mutation as well as the architecture-altering operations. This flowchart shows a two-offspring version of the crossover operation.

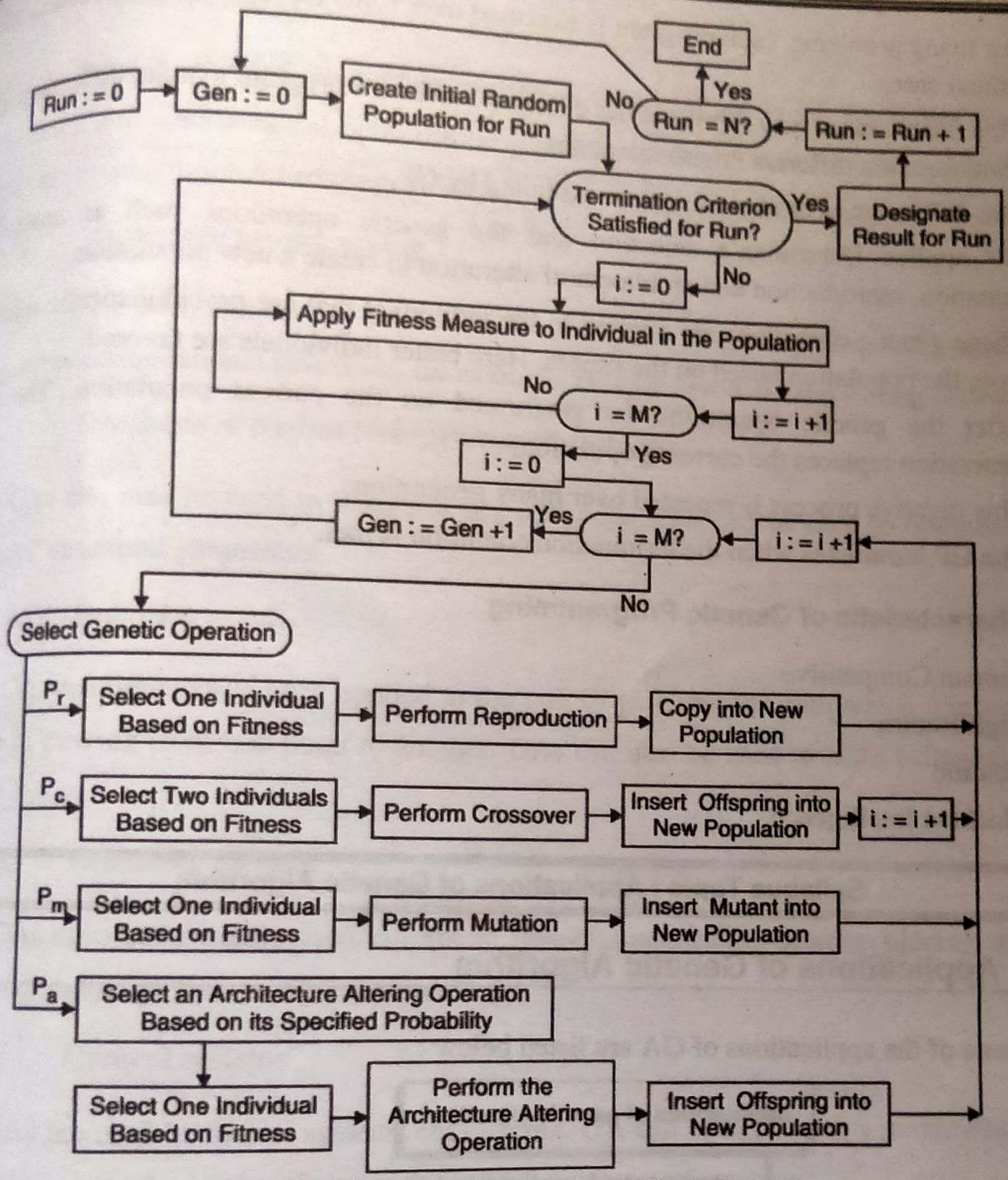


Fig. 5.14.1 : The flow chart of Genetic Programming

Explanation of Flowchart

- GP starts with an initial population of Computer programs composed of terminals appropriate to the problem.
- The individual program in the population is executed.
- Each individual program's performance (also called fitness) is compared.
- The fitness of a computer program may be measured in many ways for example the time required to bring the system in desired target state, number of errors between the actual output and desired output, the accuracy of the program in recognizing a pattern etc.

- For many problems, each program is executed over some representative test cases called fitness cases.
- This fitness cases may represent the different values for program's input, different initial condition or a different environment etc.,
- The difference in the fitness is then exploited by GP.
- GP applies Darwinian's selection and the genetic operations such as crossover, mutation, reproduction and architectural alteration to create a new population.
- These genetic operations are applied to the individual that are probabilistically selected from the population based on the fitness. Here better individuals are favored.
- After the genetic operations are performed on the current population. The new generation replaces the current population.
- This iterative process is repeated over many generations.
- The GP terminates when the termination condition is met.

Characteristic of Genetic Programming

1. Human Competitive
2. High Return
3. Routine
4. Machine Intelligence

Syllabus Topic : Applications of Genetic Algorithm

5.15 Applications of Genetic Algorithm

Some of the applications of GA are listed below :

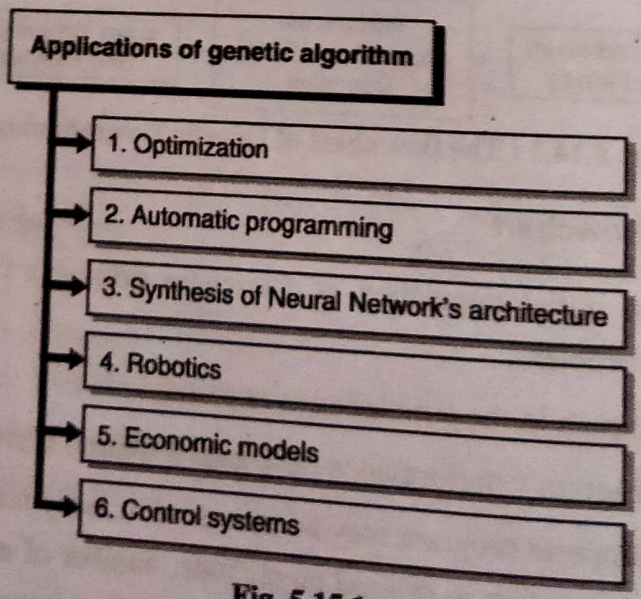


Fig. 5.15.1

1. Optimization

GAs have been used in a wide variety of optimization tasks, including numerical optimization and combinatorial optimization problems such as Travelling Salesman Problem (TSP), circuit design, job scheduling, video and sound quality optimization etc.

2. Automatic programming

GAs have been extensively used to evolve computer programs for specific tasks and to design other computational structures, for example, cellular automata and sorting networks.

3. Synthesis of Neural Network's architecture

GAs can also be used to design neural networks. GAs can be used to optimize neural networks' structural parameters.

4. Robotics

GA techniques have been applied to the task of planning the path which a robot arm is to take in moving from one point to another. GAs can also be used to learn behaviour of the robot.

5. Economic models

GAs have been widely used to forecast financial markets, to develop bidding strategies and also to model processes of innovation.

6. Control systems

GAs are used in control systems engineering. GA can be applied to a number of control methodologies for the improvement of the overall system performance.

Syllabus Topic : Convergence of GA

5.16 Convergence of GA

- In GAs, as we proceed with more and more generations, there may not be much improvement in the population fitness and the best individual chromosome may not change for subsequent generations.

- With higher generations, 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 that of the best individuals.
- A GA is usually said to converge when there is no significant improvement in the values of fitness of the population from one generation to the next.
- Examples of stopping criteria can be :

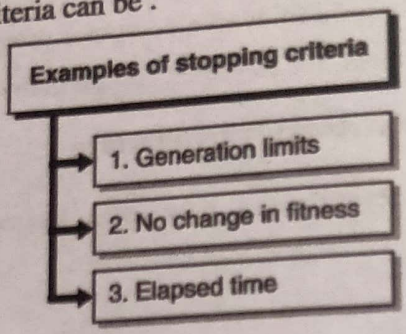


Fig. 5.16.1

→ 1. **Generation limits :** GA stops when the specified number of generations has evolved. Algorithm finds a suitably high fitness individual higher than the specified fitness threshold.

→ 2. **No change in fitness :** The genetic algorithm stops when there is no change in the population's best fitness.

→ 3. **Elapsed time :** GA will end when specified time has elapsed.

- The termination or convergence criterion finally brings the search to halt. The following are few methods of termination.
  1. Best Individual
  2. Worst Individual
  3. Sum of Fitness' Median Fitness.

☞ **Advantages of GA**

1. GAs are easy to understand since they do not demand the knowledge of complex mathematics.
2. Good for noisy environment.
3. Easy to discover global optimum.
4. Can work for wider solution space.

5. They can solve multimodal, non-differentiable, non-continuous or even NP-complete problems.
6. GAs are inherently parallel and distributed.
7. Flexible in forming building blocks for hybrid applications.
8. Have substantial history and range of use.

Limitations of GA

1. Not all optimization problems can be solved by means of a GA. Such problems are called variant problems. In some of the problems the fitness function which generates a block of chromosome is not known or poorly known blocks.
2. GA doesn't always guarantee to find a global optimum.
3. GAs are not suitable for real-time control systems since the difference between the shortest and longest optimization response time is much larger than with conventional gradient methods. This property of GA limits the GA's use in real-time systems.
4. Choosing various parameters such as the size of population, mutation rate, crossover rate, the selection method and deciding a fitness function has to be done meticulously.
5. The algorithm may prematurely converge.

Syllabus Topic : Advances in Genetic Algorithms

5.17 Advances in Genetic Algorithms

Many advancements in the field of genetic algorithm have been seen since its inception. Recent advancement of GA includes,

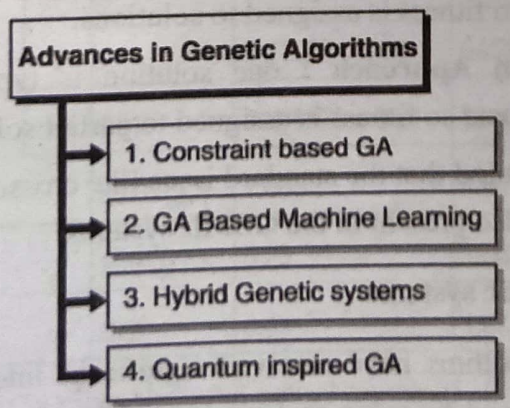


Fig. 5.17.1

1. Constraint based GA

- Constrained Optimization Problems are those optimization problems in which we have to maximize or minimize a given objective function value that is subject to certain constraints. Therefore, not all results in the solution space are feasible.
- In such a scenario, crossover and mutation operators might give us solutions which are infeasible. Therefore, additional mechanisms have to be employed in the GA when dealing with constrained Optimization Problems.
- Some of the most common methods are :
  - Using penalty functions which reduces the fitness of infeasible solutions, preferably so that the fitness is reduced in proportion with the number of constraints violated or the distance from the feasible region.
  - Using repair functions which take an infeasible solution and modify it so that the violated constraints get satisfied.
  - Not allowing infeasible solutions to enter into the population at all.
  - Use a special representation or decoder functions that ensures feasibility of the solutions.

2. GA Based Machine Learning

- Genetic Algorithms also find application in Machine Learning. Classifier systems are a form of genetics-based machine learning (GBML) system that are frequently used in the field of machine learning. GBML methods are a niche approach to machine learning.
- There are two categories of GBML systems :
  - **The Pittsburg Approach :** In this approach, one chromosome encoded one solution, and so fitness is assigned to solutions.
  - **The Michigan Approach :** one solution is typically represented by many chromosomes and so fitness is assigned to partial solutions.
- It should be kept in mind that the standard issue like crossover, mutation, Lamarckian or Darwinian, etc. are also present in the GBML systems.

3. Hybrid Genetic systems

Hybrid genetic algorithms have received significant interest in recent years and are being increasingly used to solve real-world problems. A genetic algorithm is able to incorporate other techniques within its framework to produce a hybrid that reaps the best from the combination. Genetic algorithms can be combined with other technologies like neural networks, fuzzy logic or any other intelligent systems.

4. Quantum inspired GA

A novel evolutionary computing method called quantum genetic algorithms has emerged where principles of quantum mechanics are used to inspire more efficient evolutionary computing methods.

5.18 Problem Solving using GA

Ex. 5.18.1 : Maximize the function $f(x) = x^2$ when $x \in [0, 31]$. Show computation of minimum two generations.

Soln. :

Here we use binary unsigned integer of length 5 to represent variable x.

Let us start with initial population size of 4.

Here initial population P_1 is chosen randomly.

For each string in initial population we compute fitness value $f(x) = x^2$.

Now perform selection on initial population P_1. We have chosen Roulette-wheel selection as a reproduction operator. The selection phase will decide which of the strings in initial population will take part in generation of new offspring.

The process is shown in the Table P. 5.18.1.

Table P. 5.18.1 : Initial population and selection phase

String No.	Initial population p_1	X	f(x)	Fi/∑f	Expected count fi/f	Actual count
1	01101	13	169	169/1170 = 0.14	169/293 = 0.58	1
2	11000	24	576	576/1170 = 0.49	576/293 = 1.97	2
3	01000	8	64	64/1170 = 0.06	64/293 = 0.22	0
4	10011	19	361	361/1170 = 0.31	361/293 = 1.23	1
Sum			1170 (∑f)	1.00	4.00	
Average			293 (f)	0.25	1.00	
Maximum			576	0.49	1.07	

From the above table, it is clear that, the actual count of string 1 and 4 is 1. Hence one copy of both the strings will be taken in the mating pool. For string 2, actual count is 2, Hence two copies of string 2 will be taken in mating pool. Since string 3 has actual count 0, it is eliminated and will not participate in further population generation process.

Table P. 5.18.1(a) shows the process of generating next population.

Table P. 5.18.1(a) : New population after cross over on P_2

String No.	Population P_2 after selection (Mating pool)	Mate (selected randomly)	Crossover site(selected randomly)	New population P_3 (after crossover)	x	f(x)
1	011011	2	4	01100	12	144
2	110010	1	4	11001	25	625
3	111000	4	2	11011	27	729
4	101011	3	2	10000	16	256
				Sum		1754
				Average		439
				Maximum		729

- Note that in the Table P. 5.18.1(a), mating pool contains one copy of string 1 and 4 and two copies of string 2 from initial population P_1.
- Now we perform crossover operation on these strings. Crossover points and crossover mates have been selected randomly.
- In this example, string 1 is mated (i.e. crossed over) with string 2. Similarly string 3 is mated with string 4 and vice versa.
- Now take this population P_3 as a current population and process it further for generating next generation.

Table P. 5.18.1(b) : Selection on population P_3

String No.	Population P_3	X	F(x) = x^2	fi/Σf	fi/f expected count	Actual count
1	01100	12	144	144/1754 = 0.08	144/439 = 0.32	0
2	11001	25	625	625/1752 = 0.35	625/439 = 1.42	1
3	11011	27	729	729/1754 = 0.41	729/439 = 1.66	2
4	10000	16	256	256/1754 = 0.15	256/439 = 0.58	1
Sum			1754 (Σf)	1.00	4.00	
Average			439 (f)	0.25	1.00	
Maximum			729	0.41	1.66	

- Note that the fitness value f(x) is increased from 576 to 729.
- Table P. 5.18.1(c) shows process of generating further population.

Table P. 5.18.1(c) : New population after crossover on P_4

String No.	Population P_4 after selection	Mate (random selection)	Crossover points (Random selection)	New population P_5 (after crossover)	x	f(x)
1	11001	3	3	11011	27	729
2	11011	4	2	11010	26	676
3	11011	1	3	11001	25	625
4	10000	2	2	10001	17	289
				Sum →		2319
				Average →		580
				Maximum →		729

Ex. 5.18.2 : Consider the population of string with 10 bits each. The objective function can assume number of 1's in a given string. The fitness function then performs "divide by 10' operation to normalize the objective function. Show computation of minimum of two generations. Assume crossover rate as 0.5 and mutation probability rate as 0.05.

Soln. :

Here we take initial population size as 4.

Objective function = number of 1's in a given string

Fitness function = (number of 1's in a given string)/10

Table P. 5.18.2 : Initial population and selection

String No.	Initial population P_1	Objective function (no. of 1's in given string)	Fitness f	Expected count f/f(av)	Actual count
1	00000 11100	3	0.3	0.3/0.6 = 0.5	0
2	10000 11111	6	0.6	0.6/0.6 = 1.0	1
3	01101 01011	6	0.6	0.6/0.6 = 1.0	1
4	11111 11011	9	0.9	0.9/0.6 = 1.5	2
		Sum	2.4	4.00	
		Average	0.6	1.00	
		Maximum	0.9	1.5	

- Note that, in the Table P. 5.18.2, the actual count of string 1 is 0, hence it is eliminated from the population.

- String 2 and 3 have count 1 each, so one copy of string 2 and string 3 will be taken in mating pool. Since string 4 has actual count 2, two copies of string 4 will be taken in mating pool.
- Table P. 5.18.2 (a) shows the process of generation next population.

Table P. 5.18.2(a) : New population after crossover on P_2

String No.	Population P_2 after selection	Mate	Crossover points		Population P_3 after crossover	f
			CP_1	CP_2		
1	1|10000| 11111	4	1	5	11111 11111	1.0
2	01101 01011	–	–	–	01101 01011	0.6
3	11111 11011	–	–	–	11111 11011	0.9
4	1|11111|1011	1	1	5	10000 11011	0.5
					Sum	3.0
					Average	0.75
					Maximum	1.0

Note that since crossover rate is 0.5 only pair of strings 1 and 4 has been crossed over. Pairs 2 and 3 are left intact.

Table P. 5.18.2(b) : Mutation Operation on population P_3

String No.	Population P_3	Bit mutated	New population P_4 (after mutation)	f
1	11111 11111	1st	01111 11111	0.9
2	01101 01011	6th	01101 11011	0.7
3	11111 11011	–	11111 11011	0.9
4	10000 11011	–	10000 11011	0.5
			Sum	3.0
			Average	0.75
			Maximum	0.9

- Note that out of 40 bits, only two bits have been muted randomly thus representing an effective mutation probability rate of 0.05.

Now, take the population P_4 as current population and process it further for generating next generation.

Table P. 5.18.2(c) : Selection on population P_4

String No.	Population P_4	Fitness f	Expected count f/f(avg)	Actual count
1	01111 11111	0.9	0.9/0.75 = 1.2	1
2	01101 11011	0.7	0.7/0.75 = 0.93	1
3	11111 11011	0.9	0.9/0.75 = 1.2	1
4	10000 11011	0.5	0.5/0.75 = 0.66	1
		Sum	4.00	
		Average	1.00	
		Maximum	1.2	

Table P. 5.18.2(d) : Next population after cross over on P_4

String No.	Population P_4	Mate	Crossover points		New population P_5 (after crossover)	f
			CP_1	CP_2		
1	01111 11111	–	–	–	01111 11111	0.9
2	011|01 11|011	3	3	7	01111 11011	0.8
3	111|11 11|011	2	3	7	11101 11011	0.8
4	10000 11011	–	–	–	10000 11011	0.5

Table P. 5.18.2(e) : Mutation operation on population P_5

String No.	Population P_5	Bit mutated	New population P_6 (after mutation)	f
1	01111 11111	–	01111 11111	0.9
2	01111 11011	–	01111 11011	0.8
3	11101 11011	6^{th}	11101 01011	0.7
4	10000 11011	3^{rd}	10100 11011	0.6

Review Questions

Q. 1 Explain genetic algorithm with the help of example. *(Ans. : Refer Sections 5.2, 5.3 and Ex. 5.18.1)*

Q. 2 What are the various types of Mutation techniques used in GA? *(Ans. : Refer Section 5.6.2)*

Q. 3 Explain the operations of genetic algorithm with help of flowchart. *(Ans. : Refer Sections 5.2 and 5.3)*

Q. 4 Explain Holland's schema Theorem. *(Ans. : Refer Section 5.11)*

Q. 5 Explain the working of Genetic programming with the help of flowchart. *(Ans. : Refer Section 5.14)*

Q. 6 Explain the working of Holland's bucket brigade algorithm. *(Ans. : Refer Section 5.13.2)*

Q. 7 Explain various operators used in genetic algorithm. *(Ans. : Refer Section 5.6)*

Q. 8 How genetic algorithm is different from traditional search algorithms ? Write limitations of GA. *(Ans. : Refer Section 5.8)*

Q. 9 Define crossover rate and mutation rate. *(Ans. : Refer Section 5.6.1 and 5.6.2)*

Q.10 Explain different selection methods used in GA. *(Ans. : Refer Section 5.5)*

❑❑❑

Chapter Ends...