Chapter 15

Genetic Algorithms

"One general law leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die."
— **Charles Darwin, The Origin of Species**

Genetic algorithms are computational techniques to solve optimization problems. It can be viewed as a technique for learning based on the concept of simulation, which is based on evolution strategy.

Learning Objectives

- Provide an overview of genetic algorithms
- Get a comprehensive background of genetic algorithms
- Introduce genetic operators
- Learn the pseudocode of genetic algorithms
- Explain future selection algorithm based on genetic algorithms
- Understand genetic algorithm-based classifiers

15.1 OVERVIEW OF GENETIC ALGORITHMS

'Artificial Intelligence' is an important area. Evolutionary computing is a branch of artificial intelligence that focuses on solving optimization problems using biological principles. It is a problem-solving technique that is based on the concept of Charles Darwin's principle of natural evolution. The theory of evolution is based on the principle of variation (i.e., individuals of a population vary and therefore are not exact copies), the principle of inheritance (i.e., individuals inherit certain traits from parents) and the principle of selection (i.e., species that are better adapted to the environment would survive and get a chance to reproduce and produce offspring). In short, organisms that can survive, adapt, and prosper continue their ability to maintain their offspring based on fitness. Here, the notion of 'fitness' is defined as how well an organism is able to adapt to the environment and survive.

Genetic algorithm (GA) is one of the important branches of evolutionary computation. It is a family of search-based algorithms that uses the concept of natural selection and principle of evolution. Genetic algorithms too are based on the Charles Darwin principle of natural selection.

The idea of evolutionary computation was introduced by Reichenberg in his work titled 'Evolution Strategies'. This idea was used by John Holland and his students to develop the concept of Genetic algorithms. In 1992, John Kaza developed the concept of genetic programming.

All living beings have cells. Every cell has a set of chromosomes, which represent DNA as a set of strings. Every character of a chromosome is called a gene. A gene encodes a trait. For example, a child is born with a brown eye. This is called a trait. Each gene has a position in a chromosome called locus. The possible settings for a trait are called alleles. A complete set of chromosomes is called genomes. Chromosomes are called genotypes. A set of chromosomes is called a population. Phenotypes represent the physical and mental characteristics such as eye colour, height, etc. The evaluation of the population is called a 'fitness function'. A chromosome that satisfies the fitness function proceeds to next generation leading to a new generation that is comparatively better than the previous generation. A chromosome that fails to satisfy a fitness function perishes and will not be carried to the next generation.

One may be wondering how these biological concepts are required for computer science professionals. The reason is that genetic algorithms simulate the theory of natural selection in computers. Table 15.1 indicates the concepts of theory of natural selection and its equivalent in genetic algorithms.

Table 15.1: Natural Selection and its Equivalents

Concepts in Natural Selections	Equivalent in Genetic Algorithms
Chromosome	String of binary bits or digits or character strings to represent candidate solutions
Population	Set of strings
Fitness of survival	Fitness function
Reproduction and mating	Application of genetic operations
Gene	Parts of the solution
Locus	Position in the string
Allele	Positional value
Genotype	Encoded solution
Phenotype	Decoded solution

Genetic algorithms thus mimic the natural selection process. GA can model the solution as a set of individuals. A solution is called a candidate solution. A set of solutions is called a population. GA provides a framework to find the 'best' solution among the candidate solutions. Thus, genetic algorithms are different from conventional methods like hill climbing, simulated annealing and Tabu search in the following way:

1. GA is a population-based technique where the traditional mathematics such as derivatives is used. Also, it is different from a single solution based technique. GA finds many solutions at a time.
2. GA is based on the concept of fitness, which are not used by traditional methods.
3. GA is based on probabilistic concepts and hence all its operations are probability based. So, there is no guarantee that GA will produce answers. This contrasts with traditional methods that are deterministic in nature.

If so, why is GA popular? The reason is that genetic algorithms are useful for solving optimization problems that are considered difficult, called as non-polynomial deterministic algorithms.

Often, these problems lack solution in polynomial time and cannot be solved. So, GA forms an important category of algorithms called approximation algorithms that are suitable for solving hard problems by giving near optimal solutions. This is different from exact solutions but acceptable as the problems otherwise cannot be solved using conventional methods. Many such hard problems are encountered in many domains such as machine learning, neural networks, deep learning, parallel algorithms, and image processing.

Therefore, one can conclude that genetic algorithm is one of the most important methods of evolutionary computation and is useful for solving difficult real-world problems. Let us discuss the mechanism of GA in the next section.

15.2 OPTIMIZATION PROBLEMS AND SEARCH SPACES

Genetic algorithms always provide suboptimal and approximate solutions, and faster for hard optimization problems. Before proceeding further, let us take an example to illustrate the optimization problem. Consider minimizing a function $f(x) = x^3$, where $0 \le x \le 15$. This is an optimization problem as the purpose is to find at what value of x, the function is maximized. The problem can be extended for many values such as $f(x_1, x_2, \cdots, x_n)$, where many variables are involved. These variables are called decision variables. The function $f(x)$ is called an objective function. An optimization problem is one that is associated with an objective function in the form of a maximizing or minimizing condition. Minimum and maximum can be termed as extremum.

In computer science, we encounter many problems where we may have to maximize a function such as increasing profit or minimize a function such as minimizing effort. Hence, the objective function of an optimization problem should be defined first. This is also known as cost function. The cost function indicates the performance or summary of the problem. Optimization involves the improvement of the cost function by selecting from available alternative cost functions to look for 'best solutions'.

If the problem's decision variables have no conditions or constraints, it is called an unconstrained optimization problem. For example, in the problem $f(x) = x^3$, if there is no restriction on the value of x, then it is an unconstrained optimization problem. But, when one puts a restriction such as, $0 \le x \le 15$, it becomes a constrained optimization problem. A constrained optimization problem is one that has an objective function and a set of constraints. Any solution that achieves the objective function while fulfilling the constraints is called a constrained optimal solution.

Traditionally, these problems are solved in calculus using derivatives or methods like simulated annealing, hill climbing, or Tabu search. Genetic algorithms can also be useful to solve unconstrained as well as constrained optimization problems.

Some of the advantages and disadvantages of genetic algorithms are listed below.

Advantages of GA

1. GAs are population-based searches and hence search many points rather than a single point in search space.
2. Can be executed in parallel and hence can deal large problems
3. Can handle continuous as well as discrete optimization problems
4. GA is a metaheuristic algorithm. So, it is a problem independent strategy for solving problems.

5. Fast
6. These algorithms find better solutions by avoiding local minima and always get the global maxima.
7. It is useful for solving optimization problems.

Disadvantages of GA

1. GA operates in a different space than the given problem space and hence tuning of hyper-parameters is must.
2. It can handle a limited domain of problems. For example, one cannot use GA for all generic problems apart from optimization problems.
3. Identification of the fitness function in GA is difficult as it depends on the problem.
4. The selection of suitable genetic operators for a given problem is difficult.
5. There is a limit on the number of iterations and population size due to computational requirement.
6. GAs are computationally very intensive programs and require more resources.
7. May terminate prematurely
8. These are probabilistic algorithms and hence there is no guarantee of solutions.

Scan for *'Examples of Optimization Problems and Search Spaces'*

Genetic algorithms function through genetic algorithm operators. Let us discuss about it now.

15.3 GENERAL STRUCTURE OF A GENETIC ALGORITHM

These are the steps required to implement genetic algorithms. Initially, a set of candidate solutions of the given problem is identified. All the individuals are candidate solutions. It is encoded as a chromosome. Depending on the problem, encoding of the chromosomes is done. Normally, a chromosome is represented as a binary string. The set of chromosomes is called population.

Population size is one of the important parameters in genetic algorithms. Size is the number of chromosomes in a population. It should be a trade-off between the number of chromosomes and the performance of the algorithm. More chromosomes lead to slowing down of the algorithm. If the number is small, then GA will have few possibilities as only a fraction of the search space is explored. The population size typically varies between 20 and 1000. Once the population is finalized, an evaluation function that checks the fitness of the population is designed. It is called as fitness function. It computes the fitness of the population and can be done concurrently. The candidate solution with a higher score is a better solution compared to the chromosome with a lesser score.

Selection operator is an important operator that selects the individuals from the current generation to create an offspring in the next generation.

Once the new population is decided, genetic operations like crossover and mutation are applied. Crossover creates an offspring from individuals by interchanging parts of chromosomes. Crossover probability is the probability of frequency of crossover performed. If the crossover probability is 100%, then all springs are created by the crossover operation, but if it is 0%, all the chromosomes are copied from the parent generation directly. The crossover is expected to create the new population that is an improvement of the previous generation.

The mutation operator randomly changes a gene to create a new individual. This occurs with a less probability. Mutation probability describes how often the mutation is performed. If it is 100%, then all the genes of the chromosomes are muted but if it is 0%, nothing is changed.

GA would terminate if:

1. There is no improvement of population even after large iterations
2. There are absolute number of iterations
3. The fitness function has reached a predefined value

The outline of genetic algorithms is given below.

Algorithm 15.1: Genetic Algorithms

1. Create an initial population randomly based on the parameter, population size.
2. Apply fitness function to evaluate population fitness as the ratio of given chromosome and fitness of the population.
3. While (! Termination condition) do.
   Select the parents randomly as per parameter population size.
4. Create a new population by mating the parents through crossover with parameter crossover probability and mutation with parameter mutation rate.
5. Replace the least fit chromosomes with chromosomes of high fitness value and update the population.
6. Evaluate new fitness value of the population for convergence.

Sometimes, a population may be dominated by similar solutions. It is called crowding. It arises when dominant chromosomes mate and then in a few generations, a possibility of similar solutions occurs. There are numerous ways to solve this problem, such as mutation (to keep population diversity), rank selection or tournament selection and fitness sharing. Fitness sharing is a process where the fitness score is reduced if there are many similar solutions in the population.

In genetic algorithm, only the information contained in the individual genotype is transmitted to the next generation. There are other types of models such as Lamarckian model and Baldwin model. In Lamarckian model, the basic assumption of the model is that an individual acquires many traits in their lifetime. It notes that an entire trait is passed on to the successive offspring. In this model, the neighbours are locally searched and if a better chromosome is present, it becomes the offspring.

James Mark Baldwin (1896) is a model where the tendency to acquire traits is encoded rather than the traits itself. In this, if the neighbours are better, then a high fitness score (to indicate the ability to acquire the trait) is provided instead of modifying the chromosome itself, thereby ensuring no direct transmission to the offspring.

Scan for the *'Flowchart of a Typical GA Algorithm'*

15.4 GENETIC ALGORITHM COMPONENTS

Some of the genetic algorithm operators are selection, crossover and mutation. The first step of genetic algorithm is encoding a solution as a chromosome. The decision variables of the optimization problem should be encoded first suitably. Similarly, decoding means retrieving the value of the decision variable from a chromosome. The following sections discuss the ways of doing it.

15.4.1 Encoding Methods

The first stage of the genetic algorithm is encoding or representation. GA encodes each candidate solution. For example, for the problem $f(x) = x^3$, where x ranges from 0 to 15, any value of x, say 3 or 5 or 9, can be a solution as this remains unknown in the beginning of the problem. These are called potential solutions and hence called candidate solutions. Candidate solutions are different from best solutions, as the best solution maximizes or minimizes the objective function subjected to the constrains and is known only after the problem is solved.

The representation of candidate solution into a suitable form is called encoding. Genetic algorithms do not work directly on decision variables, for example, in the problem, $f(x) = x^3$, GA does not directly work with the decision variable x. Instead, it works only with the encoding of x and is dependent on the given problem. There are different encoding techniques available, listed below:

1. Binary encoding
2. Permutation encoding
3. Value encoding as real number, Integer/Literal
4. Gray encoding

Binary Encoding

Binary encoding is one of the commonest methods of encoding and is useful for majority of the genetic algorithms. Here, a chromosome is represented as a string of 1's and 0's. For example chromosomes A and B can be encoded as follows:

Chromosome A:

0	1	1	0	0	1	0	1

Chromosome B:

1	1	1	0	0	0	0	1

The number of bits of a chromosome is called length. The choice of the length of the vector depends on precision. The length is determined as follows:

$$\frac{b-a}{2^m - 1} \leq precision$$

Here, a and b are parameters that indicate the range and m is the bits. For the problem, $f(x) = x^2$, when x ranges from 0 to 15, the values of a and b are 0 and 15, respectively. Let us assume that the precision required is 1, then m can be calculated as:

$$\frac{15 - 0}{2^m - 1} \leq 1$$

This implies that:

$$2^m - 1 \geq 15,$$

Therefore, m should be 4 bits. Therefore, 0 to 15 can be encoded as 0000 to 1111. If more precision is required, then the length can be changed accordingly.

The opposite concept is called decoding. Decoding converts the value of a binary string to a real value of the decision variable.

The advantage of binary encoding is that it is very common and can be represented using minimum alleles. The negative point of binary coding is that it is not natural for all the problems and corrections must be done after genetic algorithms.

Permutation Encoding

Permutation encoding is useful for order problems where the order of processing is particularly important. Each chromosome is represented as a sequence of numbers. For example, in travelling salesperson game (TSP), a salesperson starts from a city, say 1, and visits all other cities, say 2, 3, 4, 5 only once and returns to the original city 1.

For this problem, binary encoding is not a good choice. Instead, the permutation encoding technique can be applied where different ways of visiting a chromosome are chosen. For example, chromosomes A and B can be represented as follows:

Chromosome A:

1	2	3	4	5

Chromosome B:

1	5	4	3	2

Chromosome A represents a route 1, 2, 3, 4, 5 and chromosome B represents another route 1, 5, 4, 3, 2.

Value Encoding

Here, a chromosome is a set of values. The values can be real numbers, alphabets, or sets. These are dependent on problems, as problems may require this kind of encoding. The values can be real numbers, numbers, characters, or complex objects. For example, the following chromosome A is encoded as shown below.

Chromosome A:

A	G	T	A	A	A

(Character sequence)

Similarly, the chromosome can be encoded as a real number as shown below:

Chromosome B:

12.2	3.56	23.67	31	43

(Real numbers)

Gray Encoding

Gray encoding is also one of the popular methods of encoding a chromosome like binary encoding. It was designed by Frank gray which is like binary coding but with a specific property that the two successive numbers would differ only in one bit. Like binary codes, gray codes are also used in genetic algorithms.

Once encoding is done, a population is created in genetic algorithms. Let us discuss it now.

15.4.2 Population Initialization

A population is a set of chromosomes. How initial population is constructed? There are two types of initialization population:

1. Random initialization
2. Heuristic initialization

In random initialization, the population is initialized with random solution. This is the best way for most of the problems. Random initialization improves diversity. So, there is a trade-off between randomness and diversity. In some cases, the initial population can be initialized using a known heuristic of the problem. It is often useful to hybridize GA with local search. Here, local search refers to checking the neighbours in the population looking for better objective values. But, the problem with this method is that there may not be much diversity in the population.

15.4.3 Fitness Functions

Once the population is initialized, there is a need for an evaluation function. This evaluation function is called a fitness function that should be designed for the given problem. This function is equivalent to objective function of an optimization problem. The role of fitness function is very important for the success of genetic algorithms.

What is a fitness function? It is a function that is used to determine the fitness of the individuals and is the measure of the given problem. In many problems, the objective and fitness functions are the same in most of the genetic algorithms. In some genetic algorithm problems, a problem may have more than one fitness function. In that case, if $f(x)$ is the objective function, then the fitness is given as follows:

$$F(x_i) = \frac{\text{Fitness of the } i^{th} \text{ chromosome}(f(x_i))}{\text{Fitness of the population}\left(\sum_{i=1}^{\text{population size}} f(x_i)\right)}$$

The objective of the fitness function is to quantify the fitness of the chromosome as a number. Some of the important characteristics of fitness functions are given below:

1. It should be possible to compute fitness functions faster.
2. It is the quantitative measure of how fit the solution is and should be positive.
3. It should be exact. If exact fitness cannot be determined, then the fitness function should approximate it well.

If the fitness value is not positive, then it can be scaled as:

$$F(x) = a \times f(x) + b$$

Where, a and b are variables that are user controlled. The parameter a is a positive number for maximization problems and the parameter b ensures that the fitness value is positive. This process is called fitness scaling.

15.4.4 Selection Methods

Once the suitable fitness values are found, a new generation is generated from the old one. There are two ways in which it can be done. They are called steady state and generational models as shown in Figure 15.1.

Population Models

Steady state models are called incremental models where only a few chromosomes are generated while much of the chromosomes are retained. The advantage of this method is that they maintain population diversity. In generational model, N offspring are generated, and the entire population is replaced. The choice of the model also influences the performance of the genetic algorithm.

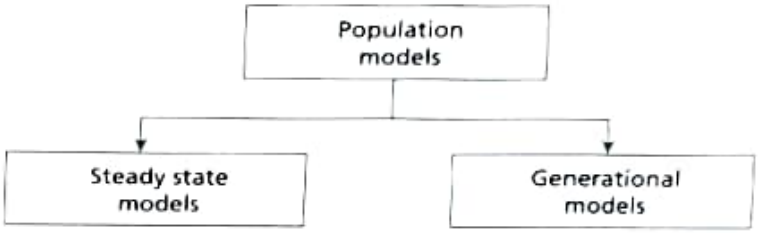

Figure 15.1: Types of Population Models

Once the population is designed, genetic operators are applied to form a new generation. The genetic operators used are selection, crossover, and mutation. The solutions that have high fitness value can survive the new generation. The application of genetic operators continues till the 'best solution' is obtained.

Selection operation is one of the important GA operations. The aim of the selection operator is to choose individuals based on the fitness function. Selection determines the individuals that get a chance to reproduce and create an offspring. Therefore, chromosomes from the population should be selected for the parent chromosomes to mate and produce an offspring. This is based entirely on the fitness score.

Some of the selection methods used in genetic algorithms are fitness- and age-based methods. In age-based methods, there is no concept of fitness. The chromosomes can proceed till a certain generation and then are eliminated irrespective of the fitness value. Fitness-based methods, on the other hand, allow the chromosomes to proceed to next generation only if its fitness value is high.

Fitness-based methods are popular and are performed in the following way.

Algorithm 15.2: Fitness-based Methods

Step 1: Calculate the fitness value of each chromosome.

Step 2: Find the total fitness of the population.

Step 3: Normalize the value by diving the fitness value of each chromosome by the total fitness of the population.

Step 4: Find the cumulative probability of the chromosome.

Some of the fitness-based methods are as follows:

1. Roulette Wheel Selection
2. Rank Selection
3. Steady-State Selection
4. Stochastic-State Sampling
5. Tournament Selection

Let us discuss about these methods.

Roulette Wheel Selection

This method of selecting the parent chromosomes for mating is popular in genetic algorithms. The basic idea is to find the selection of parent chromosomes in proportion to the fitness value. Then, these values are modelled as a roulette wheel. It should be noted that all chromosomes are allotted space in proportion to the fitness value. The spin of roulette wheel is then carried out as per the population size to select chromosomes for mating randomly. Roulette wheel can be simulated using random numbers as shown in Figure 15.2. It can be observed that the space in the wheel is in proportion to the fitness percentage.

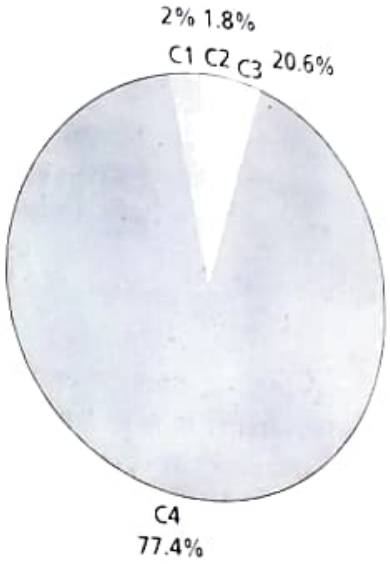

Figure 15.2: Roulette Wheel Selection

Example 15.1: Consider the problem of $f(x) = x^3$, where x ranges from 0 to 15. Let us assume that the decision variable is encoded as a binary string. The fitness value is the decoded value of binary string. Given the fitness value, construct the roulette wheel.

Solution: Let us randomly pick some chromosomes and find fitness value and fitness proportional as shown in the Table 15.2.

Table 15.2: Fitness Table

Chromosomes	Code (randomly chosen)	x	$f(x) = x^3$	$\frac{f(x)}{\sum f(x)}$	$\frac{f(x)}{\text{average} \sum f(x)}$
1.	0010	2	8	0.002	0.009
2.	0100	4	64	0.018	0.0722
3.	1001	9	729	0.206	0.0823
4.	1110	14	2744	0.774	3.096
Sum			3545		
Average			3545/4 = 886.25		
Maximum			2744		

It can be observed that for each value of x, x^3 itself is taken as fitness value. The value is normalized by diving each chromosome fitness value with the fitness of the population that is obtained by summing the fitness values of all chromosomes, and in this problem it is 3545.

It can be observed from the Figure 15.2 that the largest slice of the roulette wheel is occupied by the chromosome with the highest fitness value.

Practically, roulette wheel spin can be done as follows:

- Generate a random number, s.
- Select the first chromosome whose cumulative value crosses s.
- Repeat this for population size times to select all chromosomes.

Rank Selection

This technique ranks the chromosomes first and then associates rank 1 with the worst chromosome, rank 2 with the next worst and so on till it reaches the best chromosome, which is assigned rank N. The same roulette wheel algorithm is used. The advantage of this method is that unlike roulette wheel that allows greater space for the best one and chooses that chromosome for selection, ranking provides an equal opportunity for selection. But the disadvantage is that it leads to slower convergence of genetic algorithms.

Stochastic Universal Sampling

This method is also same as the roulette wheel selection method. But instead of one fixed point, many fixed points are present. So only one spin is required to select multiple parents. The only necessary condition for this technique to be effective is that the fitness value should be positive.

Tournament Selection

In this method, k individuals are selected from the population in random and the winner is determined. This process is repeated till all parent chromosomes are selected. This is like traditional cricket tournament, where the teams selected arbitrarily play and then the winner is selected. Tournament size is an important parameter. If it is large, then weak individuals are removed because of competition. If the tournament size is low, then more chromosomes are selected for the next generation.

Steady-State Selection

This selection method chooses the best chromosome with the largest fitness function. Most importantly, this method rejects the chromosomes with the least fitness value. Rest of the chromosomes can survive to the next generation. This is helpful in keeping the diversity of the population.

Elitism

Elitism is a method that retains the best chromosome to the new population while the rest are chosen through conventional methods. This increases performance of the genetic algorithms as it prevents the loss of good chromosomes.

15.4.5 Crossover Methods

The crossover operator models the reproduction process by combining the given set of individuals (called parents) to produce new individuals or an offspring. A crossover operation involves atleast two chromosomes to produce two child chromosomes by exchanging the parts of the chromosome based on crossover point. Crossover depends on the important parameter called probability of crossover. If the probability of crossover is 1, the entire population is subjected to crossover, and if it is 0.5, then only half of the population is subjected to crossover.

The outline for crossover algorithm is given below.

Algorithm 15.3: Crossover

Step 1: Generate a random number in the range [0 – 1].

Step 2: Pick a parameter called probability of crossover.

Step 3: Compare the random number with the probability of crossover and select chromosome pairs as parents for mating using one-point, two-point or in general, *k*-point crossover.

The description of one-point and two-point crossover is given below.

One-point Crossover

Once the crossover point of a parent chromosome is selected, the beginning of the chromosome to the crossover point is copied to the new chromosome and appended with the ones from another parent. The crossover point can be chosen randomly. For example, consider the following example of two chromosomes – 1 1 0 1 0 1 and 1 0 0 1 1 1. The crossover point is shown as a line, which yields two children chromosomes as shown in Figure 15.3.

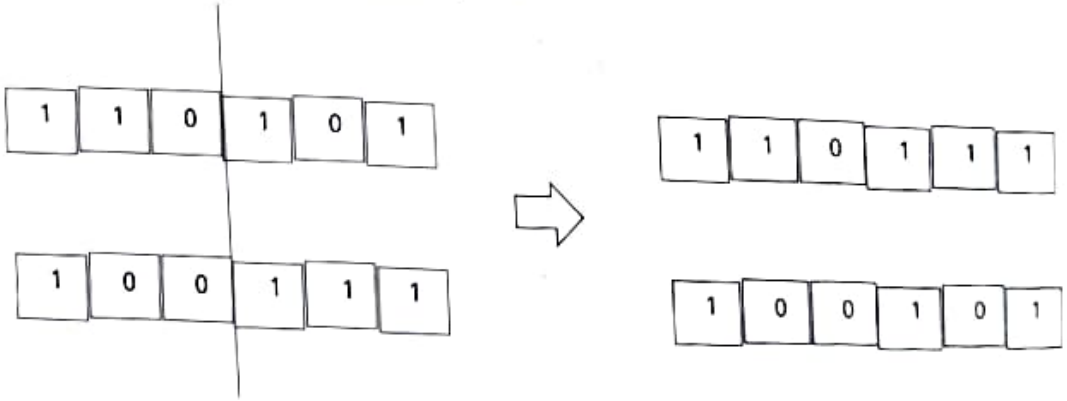

Figure 15.3: One-point Crossover

Two-point Crossover

Here, there are two crossover points. In this operation, the beginning to 1st crossover of 1st parent chromosome is copied to the resultant chromosome and the part from the first to the second crossover point of the second chromosome. The rest are from the first chromosome. For example, the two-point crossover of two chromosomes, 1 1 0 1 1 1 and 1 0 0 1 1 1 is shown through lines as in Figure 15.4.

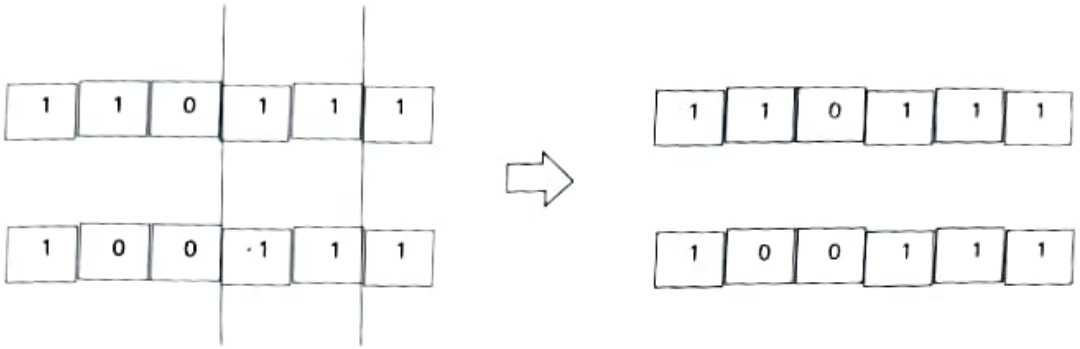

Figure 15.4: Two-point Crossover

This logic can be extended to *k*-point crossover too with *k* crossover points.

Uniform Crossover

This is another useful type of crossover where the bits are randomly copied from two parent chromosomes. A coin may be flipped for every bit and based on the outcome either the bit is included or excluded. The coin can be a biased one to ensure that the child chromosome is a better one with more randomness.

Arithmetic Crossover

useful for integer representations

This type of crossover applies some operations like OR or AND to perform crossover. The following example shows an AND operation as an arithmetic crossover. Another useful arithmetic combination is as shown below:

$$\text{Child 1: } \alpha x + (1 - \alpha)y$$

$$\text{Child 2: } \alpha x + (1 - \alpha)y$$

If $\alpha = 0.5$, then both children are identical.

Permutation Crossover

It can be recollected that the chromosomes are represented as a number in order. In the one-point permutation crossover, one crossover point is selected. The first position to the crossover point is copied to the new chromosome and rest from the second chromosome if it is not the offspring of the resultant.

15.4.6 Mutation Methods

A mutation operator is used to prevent genetic algorithms from falling into local optima. It can be noted that children do not have exact copies of the parent chromosome. There are some variations. The mutation operator tends to mimic this operation. It is controlled by a parameter called mutation rate. Its value should be low, otherwise the child turns out to be radically different from the parent.

In binary encoding, the mutation is done by selecting the bits and by inverting it using NOT operation (In binary coding, 1 becomes 0 and 0 becomes 1) as shown in Figure 15.5.

This operator is useful when no further progress is possible. Another example is that for a string, 1 0 0 1 1 1, the third bit can be mutated to give a new string 1 0 1 1 1 1.

Different kinds of mutation are discussed below.

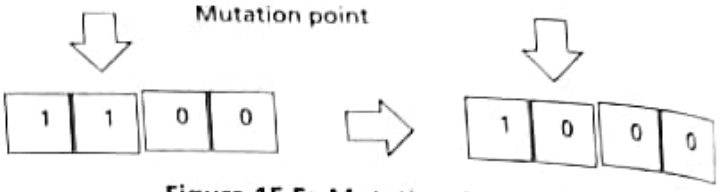

Figure 15.5: Mutation Operation

Swap Mutation

Select two positions at random in a chromosome and swap the values as shown in Figure 15.6. The resultant of this swap operation is 1 4 3 2 5. This is used in permutation encoders. This is suitable for problems like traveling salesperson problem.

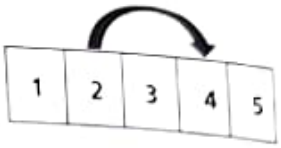

Figure 15.6: Swap Mutation

Scramble Mutation

In scramble mutation, a subset of chromosomes is selected and the values are scrambled or shuffled randomly.

Inversion Mutation

In inversion mutation, the operator takes a set of bits and reverses it. For the given example, 1 0 1 0 0, the first three bits can be reversed to yield a new string 0 1 1 1 0 0 as shown in Figure 15.7.

Figure 15.7: Inversion Mutation

Scan for *'Additional Examples and Information on Topics Discussed in Section 15.4'*

15.5 CASE STUDIES IN GENETIC ALGORITHMS

Let us discuss some case studies in genetic algorithms. One of the easiest problems that can illustrate the genetic algorithm is an optimization of a function. This is discussed in the subsequent section.

15.5.1 Maximization of a Function

Let us consider a simple problem of maximization of a function. Say $f(x) = x^3$, and $0 \leq x \leq 15$. Though this is a simple problem, it can be a foundation for the latter case studies – feature selection

and classification. Let x be an integer that ranges from $0 - 15$. What would its genetic algorithm be like?

The first step is deciding the chromosome. Binary encoding can be used to encode x. Since the maximum value of x is 15, four bits are required to encode it. The binary string 0000 is for integer 0 and 1111 is for 15. Initially, a population of $8 - 10$ binary strings can be generated randomly.

The fitness function of $f(x)$ can be value of the function itself. For example, the fitness value of 2 will be $f(x) = 2^3 = 8$, and for 5, it will be $f(x) = 5^3 = 125$. Based on the fitness value, the parent chromosomes can be selected. Then, genetic crossover can be applied to get new chromosomes and mutation operator to get population diversity. The process is repeated till new generation is generated.

This process is repeated till convergence. The convergence can be of fixed iteration or till the population is stable. The stability of the population implies that the new population is not different from older generation much. The convergence of population implies that the evolved generation is dominated by a better chromosome and it is the potential solution of the problem. The steps of the pseudocode are given below.

Algorithm 15.4: Maximization of a Function

Step 1: Form a population with population size 4. The length of the chromosome is 4.

Step 2: Let the fitness function be $f(x) = x^3$. If $x = 4$. The fitness function is $f(x) = 4^3 = 64$. Compute the fitness function for the population.

Step 3: Select operation: Select the parent chromosomes from the population if its fitness value is high.

Step 4: Apply genetic operations:

Step 4(a) Apply crossover.

Step 4(b) Apply mutation.

Step 5: Allow chromosomes of step 4 along with the parent chromosomes to form new population.

Step 6: Repeat the steps 2-5 till convergence is achieved. If convergence is achieved, then exit; else go to step 2.

The minimization of the function is also same as the above, the only difference is that sign of the objective function should be made as $z = -f(x) = -x^3$.

Example 15.2: Solve a problem $f(x) = x^3$, where $0 \leq x \leq 15$. Find some stages of application of genetic algorithms.

Solution: The first step is deciding the chromosome. Binary encoding can be used to encode x. Since the maximum value of x is 15, four bits are required to encode it. Some random sequences are chosen for the first iteration randomly and shown in Table 15.3.

Table 15.3: First Iteration

Chromosomes	Code (randomly chosen)	x	$f(x) = x^3$	$\frac{f(x)}{\sum f(x)}$	$\frac{f(x)}{\text{average} \sum f(x)}$	Truncated to
1.	0010	2	8	0.002	0.009	0
2.	0100	4	64	0.018	0.0722	0
3.	1001	9	729	0.206	0.0823	1
4.	1110	14	2744	0.774	3.096	3
Sum			3545			
Average			886.25			
Max			2749			

It can be observed from the Table 15.3, that the unproductive chromosomes like 0010 and 0100 are eliminated. The other chromosomes such as chromosome 1110 (three times) and 1001 (one time) are taken to the next generation. These chromosomes are selected for the genetic operation crossover. The crossover point is selected randomly and shown in Table 15.4.

Table 15.4: Crossover

Chromosomes Selected	Crossover Point (randomly chosen)	Offspring
1110 1110	3	1110 1110
1110 1001	3	1111 1000

The offspring constitutes a new generation as shown in Table 15.5.

Table 15.5: Second Iteration

Chromosomes	Code (randomly chosen)	x	$f(x) = x^3$	$\frac{f(x)}{\sum f(x)}$	$\frac{f(x)}{\text{average} \sum f(x)}$	Truncated to
1.	1110	14	2744	0.2927	1.17	1
2.	1110	14	2744	0.2927	1.17	1
3.	1111	15	3375	0.36	1.44	1
4.	1000	8	512	0.055	0.218	0
Sum				9375		
Average				2343.75		
Max				3375		

One can also observe that the sum and average of the population are higher than the first generation. Thus, the fitness of the population is increased.

It can also be observed that the unproductive chromosome 1000 gets eliminated and the chromosome 1111 dominates, that is the solution. Thus, genetic algorithm terminates here. But in real-world problems, many iterations need to be carried out for getting a final solution.

The initial temperature is set to 1 and the minimum temperature is often in the order of 10^{-4}. One of the major problems of simulated annealing is that the algorithm needs to be provided with an initial solution. This is done either with the prior knowledge of the problem or randomly. The temperature is reduced by multiplying the temperature by a fraction, called α, till it reaches the minimum temperature. At every temperature, the optimization task is performed. The optimization task involves finding the neighbour solution and accepting it if the difference between the current solution and the neighbouring solution is with a probability of $e^{|f(c)-f(n)|}$, where c is the current solution and n is the neighbouring solution. This probability is inverse to the increasing cost. The neighbouring solution is obtained by perturbing the current solution a little. This includes techniques like moving the bits in a random direction, shifting the bits randomly, permutating and swapping the elements randomly. The key idea is that by accepting the less optimal solution than the current one with a probability, the algorithm is more likely to converge to global optimum.

15.6.2 Genetic Programming

Genetic Programming is an extension of Genetic algorithms. It is an important branch of Evolutionary computing. John koza invented the concept of genetic programming. While the algorithmic structure of genetic algorithms and genetic programming is the same, genetic programming differs from genetic algorithms in terms of encoding and the approach of optimizing the executable code itself instead of parameters. John Koza's used LISP programs to perform various tasks. LISP is a part of automatic programming where programs are written to generate programs. These programs can be represented as a tree called parse tree. Tree encoding is employed by genetic programming where the encoding is done in the form of a tree instead of simple binary encoding techniques of genetic algorithms.

The outline of a genetic programming is given below.

Algorithm 15.5: Genetic Programming

Step 1: Choose possible functions and terminals.
Step 2: Generate an initial population of random trees using the set of functions and terminal. A random tree can be of any size.
Step 3: Calculate fitness function of the generated trees.
Step 4: Apply crossover and mutation.
Step 5: Repeat till convergence is achieved.

John Koza used genetic programming successfully for solving block world problems. The goal of the block world problem is forming a word 'UNIVERSAL' by arranging the block in order. This requires movement of a block from a table to stack or a stack to a table by the agent and should ensure that the required order is maintained. The agent can be a human or robot. The block world problem is shown in Figure 15.8.

The operations required by the problem are listed below:

1. CS — Current stack returns top block of stock, if not NIL

2. TB — Top of the block and returns the name of the topmost block and all. Blocks below it in correct order, if not NIL.
3. NN — Next block needed immediately above top of the block (TB)
4. MOVE (x) — Move to stack, if x is on table
5. MT (x) — Move to table

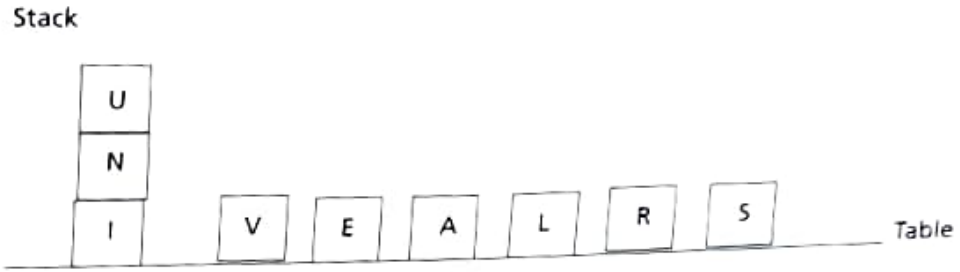

Figure 15.8: A Block World

6. DU (expr1, expr2) — Evaluates two expressions, exp1, exp2 till it is true
7. NOT (expr) — True, if expression expr is NIL
8. EQ (expr1, expr2) — True, if expressions are same, expr1 = expr2

Fitness is the number of sample fitness cases for which the stack is true after the program is run. John koza successfully proved that GP can automatically generate the sequence of operations at generation 10 with a fitness value 166.

Scan for *'Additional Information on Genetic Programming'*