Chapter 2

OUTLINE OF VARIOUS ADAPTIVE STRUCTURE AND LEARNING ALGORITHMS EMPLOYED FOR ADAPTIVE MODEL DEVELOPMENT
OUTLINE OF VARIOUS ADAPTIVE STRUCTURE AND LEARNING ALGORITHMS EMPLOYED FOR ADAPTIVE MODEL DEVELOPMENT

2.1 Introduction

The main motive of the research work carried out in this thesis is to develop elegant and efficient forecasting model for prediction of nonlinear financial time series. All these adaptive models inherently need suitable adaptive structures and appropriate learning rules to train the parameters of these models. In this chapter, we briefly deal with some selected adaptive architectures such as adaptive auto regressive moving average (ARMA), functional link artificial neural network (FLANN), multilayer artificial neural network (MLANN) and radial basis function neural network (RBFNN). In addition we provide details of some widely used population based evolutionary and swarm intelligence based derivative free techniques such as genetic algorithm (GA), differential evolution (DE), particle swarm optimization (PSO), bacterial foraging optimization (BFO) and cat swarm optimization (CSO) for training the parameters or coefficients of the adaptive structures. These new population based cooperative learning rules have been developed to achieve optimal or near optimal coefficients of the models and used as the basic backbones of prediction models employed in various chapters of the thesis.

2.2 Auto Regressive Moving Average (ARMA) Prediction Model

The ARMA prediction model essentially consists of a feed forward and feedback linear combiners. The feed forward portion acts as moving average (MA) or all-zero network where as the feedback portion functions as an autoregressive (AR) or all-pole network. Thus the ARMA model [17] contains both feed forward and feedback coefficients which need to be properly trained using appropriate learning algorithm. The amounts of delays on the feedback side are suitably selected so as to provide best possible prediction performance. This model is an adaptive pole-zero structure and is described by the recursive difference equation given in (2.1)

\[ y(n) = \sum_{m=1}^{N-1} a_m(n)y(n-m) + \sum_{m=0}^{M-1} b_m(n)x(n,m) \]  

(2.1)
where \( x(n) \) and \( y(n) \) represent the \( n \)th input pattern and output of the model respectively. The current estimated output \( y(n) \) depends on the past estimated output samples \( y(n-m), m = 1, 2, \ldots, N - 1 \) and the features \( x(n, m) \) of the current financial input. The coefficients \( \{a_m(n), b_m(n)\} \) are adjusted using some learning rules until the appropriate model is developed. \( d(n) \) is the desired or target financial value. The pole and zero parameters of the ARMA model are \( a_m \) and \( b_m \) respectively.

![Fig. 2.1 Adaptive ARMA based prediction model](image)

The output error is computed as \( e(n) = d(n) - y(n) \) and is generated by subtracting the model output in (2.1) from the true or desired value, \( d(n) \). The weights of the ARMA model are updated iteratively using some learning algorithm to minimize the mean squared error value. The minimization process leads to optimum weights of the ARMA based prediction model. The feed forward and feedback weights of the ARMA model are usually updated by the forward backward least mean square (FBLMS) algorithm given by (2.8). The aggregate coefficient vector of the model is given as

\[
W(n) = [b_0(n), \ldots, b_{M-1}(n), a_1(n), \ldots, a_{N-1}(n)]^T
\]  

(2.2)

The corresponding data vector is represented as

\[
S(n) = [x(n,0), \ldots, x(n,M - 1), y(n - l), \ldots, y(n - N + 1)]^T
\]  

(2.3)

The output of the ARMA model at the nth iteration is obtained as
\[ y(n) = W^T(n) \times S(n) \quad (2.4) \]

The estimated gradient vector is given by

\[ \nabla(n) = -2(d(n) - y(n)) \times [\alpha_1(n) \cdots \alpha_{N-1}(n) \beta_0(n) \cdots \beta_{M-1}(n)] \quad (2.5) \]

where

\[ \alpha_k(n) = \frac{\partial y(n)}{\partial a_k(n)} = y(n-k) + \sum_{m=1}^{N-1} a_m(n) \frac{\partial y(n-m)}{\partial a_k(n-m)} \]

\[ \alpha_k(n) = y(n-k) + \sum_{m=1}^{N-1} a_m(n) \alpha_k(n-m) \quad (2.6) \]

and

\[ \beta_k(n) = \frac{\partial y(n)}{\partial b_k(n)} = x(n,k) + \sum_{m=0}^{M-1} a_k(n) \frac{\partial y(n-m)}{\partial b_k(n-m)} \]

\[ \beta_k(n) = x(n,k) + \sum_{m=0}^{M-1} a_k(n) \beta_k(n-m) \quad (2.7) \]

Finally the FBLMS update algorithm is given by

\[ W(n+1) = W(n) - \mu \times \nabla(n) \quad (2.8) \]

where \( \nabla(n) \) is computed using (2.5), (2.6) and (2.7).

### 2.3 Artificial Neural Network (ANN)

This section provides the outlines of various ANN architectures used in the development of the prediction models. This includes the MLANN, RBFNN and the FLANN. The artificial neural network (ANN) takes its name from the network of nerve cells in the brain. Recently, the ANN has proved to be an important nonlinear technique for classification and optimization problems [14, 18, 19]. McCulloch and Pitts [14] have developed the neural networks for different computing machines. There are extensive applications of various types of ANNs in the field of communication, control,
instrumentation and forecasting. The ANN is capable of performing nonlinear mapping between the input and output space due to its large parallel interconnection between different layers and the nonlinear processing characteristics. An artificial neuron consists of a computing element that performs the weighted sum of the input signal and the connecting weights. The sum is then added with the bias or threshold and the resultant signal is then passed through a nonlinear function of sigmoid or hyperbolic tangent type. Each neuron is associated with three parameters whose learning can be adjusted. These are the connecting weights, the bias and the slope of the nonlinear function. For the structural point of view, an ANN may be of single or multi layers. In multilayered structure, there are one or many artificial neurons in each layer. Each neuron of the one layer is connected to each neuron of the next layer through connecting weights.

2.3.1 Structure of an Artificial Neuron

The structure of a single artificial neuron is shown in Fig. 2.2.

![Structure of a single neuron](image)

Fig. 2.2 Structure of a single neuron

The operation in a neuron involves the computation of the weighted sum of inputs and threshold [14,18,19]. The resultant signal is then passed through a nonlinear activation function. This is also called as a perceptron, which is built around a nonlinear neuron. The output of the neuron is represented as,

\[
y(n) = \varphi \left[ \sum_{j=1}^{N} W_j(n) X_j(n) + \alpha(n) \right] \tag{2.9}
\]
where $\alpha (n)$ is the threshold to the neurons at the first layer, $w_j(n)$ is the weight associated with the $j^{th}$ input, $N$ is the number of inputs to the neuron and $\phi (\cdot)$ is the nonlinear activation function. Different types of nonlinear functions commonly employed are shown in Fig. 2.3.

![Diagram showing different types of nonlinear activation functions](image)

**Fig. 2.3 Different types of nonlinear activation functions, (a) Signum function or hard limiter, (b) Threshold function, (c) Sigmoid function, (d) Piecewise Linear**

**Signum Function:** This type of activation function is defined as

$$
\varphi(v) = \begin{cases} 
1 & \text{if } v > 0 \\
0 & \text{if } v = 0 \\
-1 & \text{if } v < 0 
\end{cases}
$$

**(2.10)**

**Threshold Function:** This function is represented as,

$$
\varphi(v) = \begin{cases} 
1 & \text{if } v \geq 0 \\
0 & \text{if } v < 0 
\end{cases}
$$

**(2.11)**

**Sigmoid Function:** This function is S-shaped and is the most common form of the activation function used in artificial neural network. It is a function that exhibits a graceful balance between linear and nonlinear behavior.

$$
\varphi(v) = \frac{1}{1 + e^{-av}}
$$

**(2.12)**

where, $v$ is the input to the sigmoid function and $a$ is the slope of the sigmoid function.

For the steady convergence a proper choice of $a$ is required.
**Piecewise-Linear Function:** This function is defined as

\[
\varphi(v) = \begin{cases} 
1, & v \geq +\frac{1}{2} \\
, & +\frac{1}{2} > v > -\frac{1}{2} \\
0, & v \leq -\frac{1}{2} 
\end{cases}
\]  

(2.13)

where the amplification factor inside the linear region of operation is assumed to be unity. Out of these nonlinear functions the sigmoid activation function is extensively used in ANN structure.

### 2.3.2 Multilayer Artificial Neural Network (MLANN)

In the multilayer artificial neural network or multilayer perceptron (MLP), the input signal propagates through the network in a forward direction, on a layer-by-layer basis. This network has been applied successfully to solve some difficult and diverse problems by training the parameters of the network in a supervised manner with a highly popular algorithm known as the back-propagation algorithm [14, 18]. The scheme of a MLANN architecture using four layers is shown in Fig. 2.4. In this figure \( x_j(n) \) represents the input to the network, \( f_j \) and \( f_k \) represent the output of the two hidden layers and \( y_i(n) \) represents the output of the final layer of the neural network. The connecting weights between the input to the first hidden layer, first to second hidden layer and the second hidden layer to the output layer are represented by \( w_{ij}, w_{jk}, w_{kl} \) respectively.

![Fig. 2.4 A MLANN Architecture](image_url)
If \( P \) is the number of neurons in the first hidden layer, then the output of the \( j^{th} \) neuron of first hidden layer is calculated as,

\[
f_j = \varphi_j \left[ \sum_{i=1}^{N} W_{ij} x_i(n) + \alpha_j \right], \quad i = 1, 2, 3, ..., N , \quad j = 1, 2, 3, ..., P_1
\]

(2.14)

where \( \alpha_j \) is the threshold to the neurons of the first hidden layer, \( N \) is the number of inputs and \( \varphi (\cdot) \) is the nonlinear activation function of the neurons of the first hidden layer which is defined in (2.11). The time index \( n \) has been dropped to make the equations simpler. Let \( P_2 \) be the number of neurons in the second hidden layer. The output of this layer is represented as, \( f_k \) and may be written as

\[
f_k = \varphi_k \left[ \sum_{j=1}^{P_1} W_{kj} f_j + \alpha_k \right], \quad k = 1, 2, 3, ..., P_2
\]

(2.15)

where, \( \alpha_k \) is the threshold to the neurons of the second hidden layer. The output of the final output layer can be calculated as

\[
y_l(n) = \varphi_l \left[ \sum_{k=1}^{P_2} w_{kl} f_k + \alpha_l \right], \quad l = 1, 2, 3, ..., P_3
\]

(2.16)

where, \( \alpha_l \) is the threshold to the neuron of the final layer and \( P_3 \) is the number of neurons in the output layer. The output of the MLP may be expressed as

\[
y_l(n) = \varphi_n \left[ \sum_{i=1}^{P_3} w_{li} \varphi_l \left( \sum_{j=1}^{P_2} w_{kj} \varphi_j \left( \sum_{i=1}^{N} w_{ij} x_i(n) + \alpha_j \right) + \alpha_k \right) + \alpha_l \right]
\]

(2.17)

The details of BP algorithm used to train the weights of various layers of the ANN are discussed in Section 2.4.1.1.

### 2.3.3 Functional Link Artificial Neural Network (FLANN)

The FLANN [20] is a novel single layer ANN structure capable of forming arbitrarily complex decision regions [13]. In this structure, the initial representation of a pattern is enhanced by using nonlinear function and thus the pattern dimension space is increased. The functional link acts on an element of a pattern or entire pattern itself by generating a set of linearly independent function and then evaluates these functions with the pattern as the
argument. Hence separation of the patterns becomes possible in the enhanced space. The use of FLANN not only increases the learning rate but also has less computational complexity [22]. Pao et al. [21] have investigated the learning and generalization characteristics of a random vector FLANN and compared with those attainable with MLP structure trained with back propagation algorithm by taking few functional approximation problems. A FLANN structure with two inputs is shown in Fig. 2.5.

Let \( \mathbf{X} \) is the input vector of size \( N \times 1 \) which represents \( N \) number of elements; the \( n^{th} \) element is given by:

\[
X(n) = x_n, \quad 1 \leq n \leq N
\]  

(2.18)

Each element undergoes nonlinear expansion to form \( M \) elements such that the resultant matrix has the dimension of \( N \times M \).

The functional expansion of the element \( x_n \) by power series expansion is carried out using (2.18).

\[
s_i = \begin{cases} 
  x_n & \text{for } i = 1 \\
  x_n^i & \text{for } i = 2, 3, 4, \ldots, M 
\end{cases}
\]

(2.19)

where \( l = 1, 2, \ldots, M \)

\[
s_i = \begin{cases} 
  x_n & \text{for } i = 1 \\
  \sin(l \Pi x_n) & \text{for } i = 2, 4, \ldots, M \\
  \cos(l \Pi x_n) & \text{for } i = 3, 5, \ldots, M + 1 
\end{cases}
\]

(2.20)

where \( l = 1, 2, \ldots, \frac{M}{2} \)

In matrix notation the expanded elements of the input vector is denoted by \( \mathbf{S} \) of size \( N \times (M+1) \).

The bias input is unity. So an extra unity value is padded with the \( \mathbf{S} \) matrix and the dimension of the \( \mathbf{S} \) matrix becomes \( N \times Q \), where \( Q = (M + 2) \). Let the weight vector is represented as \( \mathbf{W} \) having \( Q \) elements. The output \( y \) is given as
\[ y = \sum_{i=1}^{\varphi} s_i w_i \]  \hfill (2.21)

In matrix notation the output is written as

\[ Y = S \cdot W^T \]  \hfill (2.22)

Fig. 2.5 Structure of the FLANN prediction model

2.3.4 Radial Basis Function Neural Network (RBFNN)

An RBF network can be viewed as a special two layer network which is linear in the parameters by fixing all RBF centers and non-linearity in the hidden layer [15]. A schematic diagram of an RBF network to be used as a financial series predictor with M inputs and one output is depicted in Fig. 2.6. The performance of an RBF network depends on many factors including number of centers. Out of many basis functions the Gaussian one is more effective and is used in the proposed RBF network predictor.

Fig. 2.6 Schematic diagram of Radial basis function neural network
The output, Y of the network is given as

\[ Y(t) = w_0 + \sum_{j=1}^{N} w_j \phi \left( \| x - c_j \| \right) \tag{2.23} \]

where \( w_j, 0 \leq j \leq N \) are weights of the output layer,

\[ \phi \left( \| x, c_j \| \right) = \exp \left( -\frac{m}{d_{\text{max}}^2} \| x - c_j \|^2 \right), \quad j = 1, 2, \ldots, m \tag{2.24} \]

where \( d_{\text{max}} \) is the maximum distance between these selected centers, \( c_j \). \( \| \bullet \| \) denotes the Euclidean distance and \( m \) is the number of centers. The standard deviation or width of all the Gaussian radial basis functions is fixed at

\[ \sigma = \frac{d_{\text{max}}}{\sqrt{2m}} \tag{2.25} \]

By providing a set of the input \( x(t) \) and the corresponding desired value \( d(t), t = 1, 2, \ldots, n \), the weights \( w_j \) are determined using the linear least square (LS) method. The weight vector is updated using the pseudo inverse method [23] as

\[ w = \phi^+ d \tag{2.26} \]

where \( d \) is the desired response vector in the training set. The matrix \( \phi^+ \) is the pseudo inverse of matrix \( \phi \) and is defined as

\[ \phi^+ = \left( \phi^T \phi \right)^{-1} \phi^T \tag{2.27} \]

### 2.4 Learning Algorithms

There are many learning algorithms which are employed to train various adaptive models. The performance of these models depends on rate of convergence, training time, computational complexity and minimum mean square error achieved after training. The learning algorithms may be broadly classified into two categories: (a) derivative based (b) derivative free. The derivative based algorithms include least means square (LMS), recursive least square (RLS), forward-backward least mean square (FBLMS) and back propagation (BP). As derivative free algorithms, the genetic algorithm (GA), the particle swarm optimization (PSO), the bacterial foraging optimization (BFO), the differential
evolution (DE) and the cat swarm optimization (CSO) have been employed. In this section the details of these algorithms are outlined in sequel.

2.4.1 Derivative Based Algorithms

These algorithms are gradient search in nature and have been derived by taking derivative of the squared error as the cost function. During the process of training these algorithms tend to drive the weights of the model to local minima. This leads to premature termination of the learning algorithm. As a result the mean square error does not attain the least possible value and hence the accuracy of prediction becomes inferior. However these learning algorithms are simple to implement and can be expressed in close form equations. A brief description of each of them is presented below.

2.4.1.1 Back propagation (BP) algorithm

The generalized structure of MLANN is shown in Fig. 2.4. To derive the BP algorithm a simplified neural network with two inputs and 2-3-2-1 neurons (2, 3, 2 and 1 denote the number of neurons in the input layer, the first hidden layer, the second hidden layer and the output layer respectively) is depicted in Fig. 2.7. The parameters of the neural network can be updated in both sequential and batch mode of operation. In BP algorithm, the weights and the thresholds are initialized as very small random values. The intermediate and the final outputs of the MLANN are calculated by using (2.14), (2.15), and (2.16) respectively.

![Fig. 2.7 Neural network using BP algorithm](image-url)
The final output \( y_i(n) \) at the output of neuron \( i \), is compared with the desired output \( d(n) \) and the resulting error signal \( e_i(n) \) is obtained as
\[
e_i(n) = d(n) - y_i(n)
\] (2.28)

The instantaneous value of the total error energy is obtained by summing all errors squared over all neurons in the output layer, that is
\[
\xi(n) = \frac{1}{2} \sum_{i=1}^{P_3} e_i^2(n)
\] (2.29)

where \( P_3 \) is the no. of neurons in the output layer.

This error signal is used to update the weights and thresholds of the hidden layers as well as the output layer. The reflected error components at each of the hidden layers is computed using the errors of the last layer and the connecting weights between the hidden and the last layer and error obtained at this stage is used to update the weights between the input and the hidden layer. The thresholds are also updated in a similar manner as that of the corresponding connecting weights. The weights and the thresholds are updated in an iterative method until the error signal becomes minimized. The weights are updated according to the following equations
\[
\begin{align*}
w_{kl}(n+1) &= w_{kl}(n) + \Delta w_{kl}(n) \quad (2.30) \\
w_{jk}(n+1) &= w_{jk}(n) + \Delta w_{jk}(n) \quad (2.31) \\
w_{ij}(n+1) &= w_{ij}(n) + \Delta w_{ij}(n) \quad (2.32)
\end{align*}
\]

where, \( \Delta w_{kl}, \Delta w_{jk}, \Delta w_{ij} \) are the change in weights of the second hidden layer-to-output layer, first hidden layer-to-second hidden layer and input layer-to-first hidden layer respectively. The terms \( \Delta w_{kl}(n) \) can be computed as
\[
\Delta w_{kl}(n) = -2\mu \frac{d\xi(n)}{dw_{kl}(n)} = 2\mu e(n) \frac{dy_i(n)}{dw_{kl}(n)}
\] (2.33)
\[
= 2\mu e(n)\phi'[\sum w_{kl}f_k + \alpha_i]f_k
\]

where, \( \mu \) is the convergence coefficient \((0 \leq \mu \leq 1)\). Similarly \( \Delta w_{jk}(n) \) and \( \Delta w_{ij}(n) \) can also be computed in the similar manner.

The threshold or bias term of each layer can be updated in a similar manner using equations
\[ \alpha_i(n+1) = \alpha_i(n) + \Delta \alpha_i(n) \]  
(2.34)

\[ \alpha_k(n+1) = \alpha_k(n) + \Delta \alpha_k(n) \]  
(2.35)

\[ \alpha_j(n+1) = \alpha_j(n) + \Delta \alpha_j(n) \]  
(2.36)

where, \( \Delta \alpha_i(n) \), \( \Delta \alpha_k(n) \), \( \Delta \alpha_j(n) \) are the change in thresholds of the output, hidden and input layer respectively. The change in the threshold is represented as,

\[ \Delta \alpha_i(n) = -2\mu \frac{d\xi(n)}{d\alpha_i(n)} = 2\mu e(n) \frac{dy_i(n)}{d\alpha_i(n)} \]

\[ = 2\mu e(n)\phi_i \left[ \sum_{k=1}^{n_i} w_{ki} f_k + \alpha_i \right] \]  
(2.37)

Similarly \( \Delta \alpha_k(n) \) and \( \Delta \alpha_j(n) \) can also be computed as (2.37).

### 2.4.2 Derivative Free Algorithms / Evolutionary Computing Based Algorithms

#### 2.4.2.1 Genetic Algorithm (GA)

Genetic algorithms are a class of evolutionary computing techniques, which has been a rapidly growing area of artificial intelligence. Genetic algorithms are inspired by Darwin's theory of evolution. Simply said, problems are solved by an evolutionary process resulting in the best (fittest) solution (survivor) - in other words, the solution is evolved.

Evolutionary computing was introduced in the 1960s by Rechenberg in his work "Evolution strategies" (Evolutions strategies in original). His idea was then developed by other researchers. The Genetic Algorithm (GA) was invented by John Holland and his student and colleague [7]. This led to the book "Adaption in Natural and Artificial Systems" published in 1975. The algorithm begins with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are then selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are, the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.
2.4.2.1 Outline of the basic genetic algorithm

1. [Start] Generate random population of $n$ chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome $x$ in the population
3. [New population] Create a new population by repeating following steps until the new population is complete
   a  [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   b  [Crossover] With a crossover probability cross over the parents to form new offspring (children). If no crossover is performed, offspring is the exact copy of parents.
   c  [Mutation] With a mutation probability mutate new offspring at each location (position in chromosome).
   d  [Accepting] Place new offspring in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

The outline of the Basic GA provided above is very general. There are many parameters and settings that can be implemented differently in various problems.

Elitism is often used as a method of selection. This means, that at least one of best solution of a generation is copied without change to a new population, so the best solution can survive to the succeeding generation.

2.4.2.1.2 Operators of GA

Overview:

The crossover and mutation are the most important parts of the genetic algorithm. The performance is influenced mainly by these two operators.
**Encoding of a Chromosome**

A chromosome should in some way contain information about solution that it represents. Very often the chromosome represented by binary string. Then typical chromosomes have patterns as provided in Fig. 2.8.

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>1101100100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>1101111000011110</td>
</tr>
</tbody>
</table>

Fig. 2.8 Representation of typical chromosomes

Each chromosome is represented by a binary string. Each bit in the string can represent some characteristics of the solution. There are many other ways of encoding. The encoding depends mainly on the problem to be solved. For example, one can encode directly integer or real numbers; sometimes it is useful to encode some permutations and so on.

**Crossover**

Crossover operates on selected genes from parent chromosomes and creates new offspring. The simplest way is to choose randomly some crossover point and copy everything before this point from the first parent and then copy everything after the crossover point from the other parent. There are other ways to make crossover. For example more crossover points may be chosen. A one point crossover result is depicted in Fig. 2.9.

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>11011</th>
<th>00100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>11011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>11011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 2</td>
<td>11011</td>
<td>00100110110</td>
</tr>
</tbody>
</table>

Fig. 2.9 Offspring formation due to single point crossover
Mutation

Mutation is intended to prevent falling of all solutions in the population into a local optimum of the solved problem. Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding a few randomly chosen bits are changed from 1 to 0 or from 0 to 1. The mutation process is illustrated Fig. 2.10.

<table>
<thead>
<tr>
<th>Original offspring 1</th>
<th>1101111000011110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original offspring 2</td>
<td>1101100100110110</td>
</tr>
<tr>
<td>Mutated offspring 1</td>
<td>1100111000011110</td>
</tr>
<tr>
<td>Mutated offspring 2</td>
<td>1101101100110110</td>
</tr>
</tbody>
</table>

Fig. 2.10 An illustration of mutation operation

The technique of mutation (as well as crossover) depends mainly on the encoding of chromosomes. For example, when permutations are encoded the mutation can be performed as an exchange of two genes.

2.4.2.1.3 Parameters of GA

Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability

It indicates the frequency of performance of crossover. If there is no crossover, the offspring are exact copies of the parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, the whole new generation is made from exact copies of
chromosomes from old population. The crossover operation is made such that new chromosomes contain good parts of old chromosomes and therefore the new chromosomes would be better. However, it is good that some parts of old population survive to the next generation.

**Mutation probability**

This signifies how often parts of chromosome are mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, the whole chromosome is changed, if it is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, otherwise the GA changes to a random search technique.

**Other Parameters**

There are also some other parameters of GA such as the population size.

**Population size**

It signifies how many chromosomes are present in population (in one generation). If there are too few chromosomes, then GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, then the GA slows down.

2.4.2.1.4 Selection

**Introduction**

The chromosomes are selected from the population to act as parents for crossover operation. The problem is the manner selection of these chromosomes. According to Darwin's theory of evolution, the best ones survive to create new offspring. There are many methods in selecting the best chromosomes. Examples are roulette wheel selection, Boltzman selection, tournament selection, rank selection and steady state selection. In this thesis only the tournament selection is used as it performs better than the others.

**Tournament Selection**

A selection strategy in GA is simply a process that favors the selection of better individuals in the population for the mating pool. There are two important issues in the evolution
process of genetic search, population diversity and selective pressure. Population diversity means that the genes from the already discovered good individuals are exploited while promising the new areas of the search space continue to be explored. Selective pressure is the degree to which the better individuals are favored. The tournament selection strategy provides selective pressure by holding a tournament competition among individuals [24].

2.4.3 Particle Swarm Optimization (PSO)

2.4.3.1 Basic method

Natural creatures sometimes behave as a swarm. One of the main streams of artificial life researches is to examine how natural creatures behave as a swarm and reconfigure the swarm models inside a computer. Swarm behavior can be modeled with a few simple rules. School of fishes and swarm of birds can be modeled with such simple models. In 2001 the PSO algorithm as simulation of bird flocking in two-dimensional space was developed [8]. The position of each agent is represented by XY axes position and also the velocity is expressed by \( v_x \) (the velocity of X axis) and \( v_y \) (the velocity of Y axis). Modification of the agent position is realized by the position and velocity information. Bird flocking optimizes a certain objective function. Each agent knows its best value so far (pbest) and its XY position. This information is analogy of personal experiences of each agent. Moreover, each agent knows the best value so far in the group (gbest) among pbests. This information is analogous to the knowledge of how the other agents around them have performed. Namely, each agent tries to modify its position using the following information:

1. the current positions \((x, y)\)
2. the current velocities \((v_x, v_y)\)
3. to go to the center of the swarm
4. the distance between the current position and the pbest
5. the distance between the current position and the gbest

2.4.3.2 Particle swarm optimization (PSO) algorithm

This modification can be represented by the concept of velocity. The velocity of each agent [8] can be modified by the following equation:

\[
V_i^{k+1} = wV_i^{k} + c_1 \times rand_1 \times (pbest_i - s_i^k) + c_2 \times rand_2 \times (gbest_i - s_i^k)
\]  

(2.38)
where

\[ V_i^k : \text{velocity of agent } i \text{ at iteration } k \]
\[ w : \text{weighting function} \]
\[ c_j : \text{weighting factor} \]
\[ \text{rand} : \text{random number between 0 and 1} \]
\[ s_i^k : \text{current position of agent } i \text{ at iteration } k \]
\[ pbest_i : \text{personal best of agent } i \]
\[ gbest : \text{global best of the group} \]

The weighting function defined in (2.39) is utilized in (2.38)

\[ w = w_{\max} - \frac{w_{\max} - w_{\min}}{\text{iter}_{\max}} \times \text{iter} \tag{2.39} \]

where

\[ w_{\max} : \text{initial weight} \]
\[ w_{\min} : \text{final weight} \]
\[ \text{iter}_{\max} : \text{maximum iteration number} \]
\[ \text{iter} : \text{current iteration number} \]

Using (2.39) an initial velocity becomes close to the pbest and the gbest can be calculated. The current position (searching point in the solution space) is modified as in (2.40)

\[ s_i^{k+1} = s_i^k + V_i^{k+1} \tag{2.40} \]

The general flow chart of PSO is shown in Fig. 2.11 and its step wise procedure has been detailed.

**Step 1:** Generation of initial condition of each agent

Initial searching points \( s_i^0 \) and velocities \( v_i^0 \) of each agent are usually generated randomly within the allowable range. The current searching point is set to pbest for each agent. The best evaluated value of pbest is set to gbest and the agent number with the best value is stored.

**Step 2:** Evaluation of searching point of each agent

The objective function value is calculated for each agent. If the value is better than the current pbest of the agent, the pbest value is replaced by the current value. If the best
value of pbest is better than the current gbest, the gbest is replaced by the best value and the agent number with the best value is stored.

**Step 3:** Modification of each searching point

The current search point of each agent is changed using (2.38), (2.39) and (2.40).

**Step 4:** Checking the exit condition

If the current iteration number reaches the predetermined maximum iteration number, then exit. Otherwise, go to step 2.

![Flowchart](chart.png)

Fig. 2.11 General flow chart of the PSO
The features of the searching procedure of PSO can be summarized as follows:

1. As shown in (2.38), (2.39) and (2.40), the PSO can essentially handle continuous optimization problem.

2. The PSO utilizes several search points like the genetic algorithm and the search points gradually get close to the optimal point using their pbest and the gbest information.

3. The first term of the right-hand side (RHS) of (2.38) corresponds to the diversification in the search procedure. The second and third terms of that are corresponding to intensification in the search procedure. This method has a well-balanced mechanism to utilize diversification and intensification in the search procedure efficiently.

4. The PSO can handle continuous optimization problems with continuous state variables in an n-dimension solution space.

The RHS of (2.38) consists of three terms. The first term is the previous velocity of the agent. The second and third terms are utilized to change the velocity of the agent. Without the second and third terms, the agent keeps on flying in the same direction until it hits the boundary. Namely, it tries to explore new areas and, therefore, the first term is corresponding to diversification in the search procedure. On the other hand, without the first term, the velocity of the "flying" agent is only determined by using its current position and its best positions in history. Namely, the agents will tries to converge to their pbests and/or gbest and, therefore, the terms are corresponding to intensification in the search procedure.

2.4.4 Bacterial Foraging Optimization (BFO)

2.4.4.1 Introduction

Natural selection tends to eliminate animals with poor "foraging strategies" (methods for locating, handling, and ingesting food) and favor the propagation of genes of those animals that have successful foraging strategies since they are more likely to enjoy reproductive success (they obtain enough food to enable them to reproduce). After many generations, poor foraging strategies are either eliminated or shaped into good ones (redesigned).
Logically, such evolutionary principles have led the scientists to the field of "foraging theory" to hypothesize that it is appropriate to model the activity of foraging as an optimization process: A foraging animal takes actions to maximize the energy obtained per unit time, in the face of constraints presented by its own physiology and environment.

2.4.4.2 Bacterial Foraging

Bacteria have the tendency to gather to the nutrient-rich areas by an activity called chemotaxis. It is known that bacteria swim by rotating whip like flagella driven by a reversible motor embedded in the cell wall. A typical E. coli has 8-10 flagella placed randomly on a cell body. When all flagella rotate counterclockwise, they form a compact, helically propelling the cell along a helical trajectory, which is called run. When the flagella rotate clockwise, they pull on the bacterium in different directions, which causes the bacteria to tumble. The four steps involved in bacterial foraging optimization process are briefly outlined next.

![Fig. 2.12 (a) Swimming, (b) Tumbling and (c) Chemotactic behavior of Ecoli](image)

(1) **Chemotaxis:** An E. coli bacterium can move in two different ways; it can run (swim for a period of time) or it can tumble, and alternate between these two modes of operation in the entire lifetime. In the BFO, a unit walks with random direction represents a tumble and a unit walk with the same direction in the last step indicates a run. After one step move, the position of the $i^{th}$ bacterium can be presented as [10]

$$
\theta^i (j +1, k, l) = \theta^i (j, k, l) + C(i)\phi(j)
$$

(2.41)
where $\theta'(j,k,l)$ represents the $i^{th}$ bacterium at $j^{th}$ chemotactic, $k^{th}$ reproductive and $l^{th}$ elimination and dispersal step. $C(i)$ is the length of unit walk in the random direction. It is assumed to be constant and $\phi(j)$ is the direction angle of the $j^{th}$ step. During run operation, $\phi(j)$ is same as $\phi(j-1)$, otherwise, $\phi(j)$ is a random angle directed within a range of $[0,2\pi]$.

If the cost at $\theta'(j+1,k,l)$ is better than the cost at $\theta'(j,k,l)$ then the bacterium takes another step of size $C(i)$ in the same direction otherwise it tumbles. This swim process continues as long as the cost function reduces, but only to a maximum number of steps, $N_r$.

(2) Swarming: The bacteria in times of stresses release attractants to signal bacteria to swarm together. It however also releases a repellant to signal others to be at a minimum distance from it. Thus all of them have a cell to cell attraction is attractant and cell to cell repulsion via repellant. The cell to cell signaling in E. coli swarm may be represented as

$$J_{cc}(\theta, P(j,k,l)) = \sum_{i=1}^{S} J_{cc}(\theta, \theta'(j,k,l)) = \sum_{i=1}^{S} [d_a \exp(-w_a \sum_{m=1}^{p} (\theta_m - \theta'_m)^2) + \sum_{i=1}^{S} h_r \exp(-w_r \sum_{m=1}^{p} (\theta_m - \theta'_m)^2)] +$$

where $J_{cc}(\theta, P(j,k,l))$ represents the objective function value to be added to the actual objective function, $S$ is the total number of bacteria, $p$ is the number of variables to be optimized, $\theta = [\theta_1, \theta_2, \cdots, \theta_p]^T$ is a point in the $p$-dimensional search domain, $d_a = \text{depth of the attractant}$, $w_a = \text{width of the attractant}$, $h_r = \text{height of the repellant}$ and $w_r = \text{width of the repellant}$.

(3) Reproduction: A reproduction step is taken after $N_r$ chemotactic steps. Let $S_r = S_b / 2$ be the number of population members who have had sufficient nutrients so that they reproduce i.e. split into two. For reproduction, the population is stored in order of ascending fitness function. The $S_r$ least healthy bacteria die and each of the rest $S_r$ bacteria each split into two identical ones which occupy the same positions in the environment. This method keeps the population size constant.
(4) **Elimination and Dispersal**: Since bacteria may be stuck around the initial positions or local optima, it is possible for the diversity of BFO to change either gradually or suddenly to eliminate the possibility of being trapped into local minima. The dispersion vent happens after a certain number of reproduction processes. A bacterium is chosen, according to a preset probability $P_{ed}$, to be dispersed and moved to another position within the environment. These events may prevent the local minima trapping effectively, but unexpectedly disturb the optimization process. The mathematical treatment of this new concept is presented in [10].

**2.4.5 Differential Evolution (DE)**

Differential Evolution (DE) is a population based stochastic meta-heuristic global optimization tool in continuous domains, introduced by Storn and Price in 1995 [9]. Due to its simplicity, effectiveness and robustness, the DE has been successfully applied for solving complex optimization problems arising in different practical applications. A population in DE consists of $P$ vectors is represented as $\bar{x}_{i,G}, i = 1,2,\ldots,P$, where $G$ is the number of generations. To keep the population within some bounds it is randomly initialized from a uniform distribution between the lower and the upper bounds defined for respective variables. These bounds are problem depended. The possible solutions known as target vectors are represented with $D$-dimensional vectors as $\bar{x}_{i,G} = (x_{i,1,G}, x_{i,2,G}, \ldots, x_{i,D,G})$ (2.43)

The initial populations are changed in each generation using sub-processes such as mutation, crossover and selection operators. In a simple DE algorithm the mutant vector $\bar{v}$ for every target vector $\bar{x}_{i,G}$ is computed as

$$\bar{v}_{i,G} = \bar{x}_{i,G} + F(\bar{x}_{r_2,G} - \bar{x}_{r_1,G}), r_1 \neq r_2 \neq r_3$$ (2.44)

where $F$ is a mutation control parameter with its value between 0 to 2 and $r_1, r_2$ and $r_3$ are randomly chosen numbers within the population size. After mutation, the crossover operator generates a trial vector, $\bar{u}_{i,G}$ using (2.45)

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & \text{if } rand_j \leq CR \text{ or } j = rn(j) \\ x_{i,j,G}, & \text{otherwise} \end{cases}$$ (2.45)
where \( j. \) (dimension number) = 1, 2, \ldots, D

\[ \text{\( rand_j \) = a random number between 0 to 1} \]

\[ \text{\( rn(j) \) = a randomly chosen index from 1, 2, \ldots, D} \]

\[ \text{\( CR \) = crossover constant between 0 to 1.} \]

The differential evolution uses a greedy selection operator as

\[
\overline{x}_{i,G+1} = \begin{cases} \overline{p}_{i,G} , & \text{if } f(\overline{p}_{i,G}) < f(\overline{x}_{i,G}) \\ \overline{x}_{i,G} , & \text{otherwise} \end{cases}
\]

(2.46)

where \( f(\overline{p}_{i,G}) \) = fitness value of the trial vector

and \( f(\overline{x}_{i,G}) \) = fitness value of the target vector.

The numbers of generations are continued until the cost function almost remains constant and does not decrease further.

**2.4.6 Cat Swarm Optimization (CSO)**

Cat Swarm Optimization is a new optimization algorithm in the field of swarm intelligence [11,25,26]. The CSO algorithm models the behavior of cats into two modes: ‘Seeking mode’ and ‘Tracing mode’. In CSO, every cat has its own position composed of D dimensions, velocities for each dimension, a fitness value, which represents the accommodation of the cat to the fitness function, and a flag to identify whether the cat is in seeking mode or tracing mode. The final solution would be the best position of one of the cats. The CSO keeps the best solution until it reaches the end of the iterations.

Cat Swarm Optimization algorithm has two modes in order to solve the problems which are described below:

**Seeking Mode:**
For modeling the behavior of cats in resting time and being-alert, we use the seeking mode. This mode is a time for thinking and deciding about next move. This mode has four main parameters such as: seeking memory pool (SMP), seeking range of the selected dimension (SRD), counts of dimension to change (CDC) and self-position consideration (SPC) [11, 25, 26].

32
The process of seeking mode is described below:

**Step 1:** Make \( j \) copies of the present position of \( \text{cat}_k \), where \( j = \text{SMP} \). If the value of \( \text{SPC} \) is true, let \( j = (\text{SMP}-1) \), then retain the present position as one of the candidates.

**Step 2:** For each copy, according to \( \text{CDC} \), randomly plus or minus \( \text{SRD} \) percent the present values and replace the old ones.

**Step 3:** Calculate the fitness values (FS) of all candidate points.

**Step 4:** If all FS are not exactly equal, calculate the selecting probability of each candidate point by (2.47), otherwise set all the selecting probability of each candidate point be 1.

**Step 5:** Randomly pick the point to move to from the candidate points, and replace the position of \( \text{cat}_k \).

\[
P_i = \frac{\text{SSE}_i - \text{SSE}_{\text{max}}}{\text{SSE}_{\text{max}} - \text{SSE}_{\text{min}}} \quad (2.47)
\]

If the goal of the fitness function is to find the minimum solution, \( \text{FS}_b = \text{FS}_{\text{max}} \), otherwise \( \text{FS}_b = \text{FS}_{\text{min}} \).

**Tracing Mode:**

Tracing mode is the second mode of algorithm. In this mode, cats desire to trace targets and foods. The process of tracing mode can be described as follow:

**Step 1:** Update the velocities for every dimension according to (2.48).

**Step 2:** Check if the velocities are in the range of maximum velocity. In case the new velocity is over-range, it is set equal to the limit.

\[
V_{k,d} = V_{k,d} + r_1 c_1 (X_{\text{best},d} - X_{k,d}) \quad (2.48)
\]

**Step 3:** Update the position of cat \( k \) according to (2.49).

\[
X_{k,d} = X_{k,d} + V_{k,d} \quad (2.49)
\]

\( X_{\text{best},d} \) is the position of the cat, who has the best fitness value, \( X_{k,d} \) is the position of \( \text{cat}_k \), \( c_1 \) is an acceleration coefficient for extending the velocity of the cat to move in the solution space and usually is equal to 2.05 and \( r_1 \) is a random value uniformly generated in the range of [0,1].
2.4.6.1 Core Description of CSO

In order to decide the proportion of two modes in the algorithm, we define a mixture ratio (MR) which indicates the rate of mixing of seeking mode and tracing mode. This parameter decides how many cats will be moved into seeking mode process. For example, if the population size is 50 and the MR parameter is equal to 0.7, there should be 50×0.7=35 cats move to seeking mode and 15 remaining cats move to tracing mode.

The CSO algorithm is summarized below:

First of all, we create N cats and initialize the positions, velocities and the flags for cats. (*) According to the fitness function, evaluate the fitness value of the each cat and keep the best cat into memory \((X_{\text{best}})\). In next step, according to cat’s flag, apply cat to the seeking mode or tracing mode process. After finishing the related process, re-pick the number of cats and set them into seeking mode or tracing mode according to MR parameter. At the end, check the termination condition, if satisfied, terminate the program, and otherwise go to (*).