Chapter 3
Soft Computing Methodologies

The primary theme of this chapter is to present the elemental aspects of SC methodologies, with special emphasis on the data-driven models and evolutionary methods. Various issues in relation to SC model development that have been employed in the present study for estimation of scour depth around different types of abutments are introduced. The theoretical aspects of data-driven models and evolutionary methods are presented in the chapter together with the learning process, model architecture, training/evolving and testing algorithm.

3.1 Introduction

Soft computing (SC) is a set of computing techniques, such as ANNs, fuzzy logic (FL), genetic algorithms (GAs), genetic programming (GP) and gene expression programming (GEP) to construct computationally intelligent systems that are capable of imitating the human reasoning process as well as handling quantitative and qualitative knowledge. It is well known that the intelligent systems, which can provide human like expertise such as domain knowledge, uncertain reasoning, and adaptation to a noisy and time-varying environment, are important in tackling practical computing problems. In contrast with hard computing methods, which only deal with precision, certainty and rigor, SC is effective in acquiring imprecise or sub-optimal but economical and competitive solutions. The guiding principle of SC is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, better rapport with reality (Zadeh, 1997). As there is no proper general mathematical model for predicting scour depth and imprecision and uncertainty are generally inherent in the scouring phenomenon, SC models can be adopted as a better alternative.

ANNs can map any random input pattern to an output pattern through training and accepted as universal function approximator (Haykin, 1999). These models do not require specifying functional relationship between the input and output parameters and thus they are more useful in solving complex hydraulic and water resource engineering problems that are often poorly defined. In recent years, it has emerged as
a viable alternative to the classical statistical methods such as, discriminant analysis, logistic regression, multiple regression models and gained a great deal of interest from researchers in many fields of study and has proved to be a more powerful and self adaptive predictive model (Lee et al., 2007, Ramírez et al., 2005). ASCE Task Committee (2000a, 2000b) summarized the application of ANN in various fields of hydrology and water resources and concluded that ANN can be used as alternative modelling tool subject to further exploration. Applications of ANN in hydraulic engineering have been presented by many researchers including Azamathulla et al. (2007a, 2008), Lee et al. (2007), Shin et al. (2010).

Fuzzy logic (Bellman et al., 1970) provides an inference morphology that enables approximate human reasoning capabilities to be applied to knowledge based systems. The integration of the techniques of fuzzy systems and ANNs suggests the novel idea of transforming the burden of designing fuzzy systems to the training and learning of the neural networks. The ANNs provide connectionist structures (fault tolerance and distributed properties) and learning abilities to the fuzzy systems whereas the fuzzy systems offer ANN a structured framework with high level IF–THEN rule thinking and reasoning. Several paradigms of neural fuzzy modelling are available in the literature such as fuzzy inference networks, fuzzy aggregation networks, fuzzy modelling networks, neural network-driven fuzzy reasoning, adaptive neuro-fuzzy inference system (ANFIS) and fuzzy associative memory systems (Rajasekaran et al., 2014). Among the neuro-fuzzy models, ANFIS has been mostly used for modelling hydraulic engineering problems. The hybrid system can very well handle imprecision and uncertainty which are generally inherent in the hydraulic engineering problems. Thus, ANFIS has attracted the attention of researchers and proved to be a more powerful method compared to the traditional methods in various fields including scour depth estimation around different types of obstructions (Azamathullah et al. 2009, 2011; Muzzammil et al., 2010; Muzzammil et al., 2011).

GAs (Holland, 1975) are computerized search and optimization algorithms based on mechanics of natural genetics and natural selection. GA is able to determine connect weights of ANN, resulting to an optimal solution the problem (Rajasekaran et al., 1996). GP (Koza, 1992) is a domain-independent problem-solving approach in which computer programs are evolved to solve or approximately solve problems based on the Darwinian principle. GP creates computer programs that consist of variables and several mathematical operators (function) sets as the solution. GEP is a
relatively new evolutionary computing technique proposed by Ferreira (2001a, 2001b) is an extension of the GP. It is a full-fledged genotype/phenotype system with chromosomes consists of one or more genes and each gene encoded as a sub-program. GEP combines the advantages of both its predecessors, i.e., GA and GP and removes some of their limitations. GEP is able to generate mathematical relationship between independent and dependent variables and thus applicable to the problems including scour depth estimation where relationship between independent and dependent parameters are poorly understood.

In brief, neural networks have the capability of adaptive learning and nonlinear function approximation; fuzzy logic deals with imprecision and approximate reasoning; and genetic algorithms are used for systematic random search.

The main goal of this research is to develop a hybrid data-driven scour depth prediction model at bridge abutments and to compare the performance of the developed model with the traditional empirical equations. The study also investigates to enhance the generalization performance of the developed hybrid model for predicting scour depth around different types of abutments. Thus, the theory and methods in relation to artificial neural networks and evolutionary methods have been discussed to develop a better understanding of this research for prediction of scour depth at abutment. Besides that, the general framework of neural networks model has been investigated with special emphasis on multilayer feedforward networks and backpropagation (BP) learning algorithm.

3.2 Artificial Neural Networks

The primary advantage of ANNs is that they can solve problems that are too complex for conventional technologies; problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be defined. This section presents the elemental aspects of ANNs and the various issues in relation to the considered ANN model development for the prediction of scour around different types of abutments. The theoretical aspects of ANNs relating to the structure of biological neuron, model of an artificial neuron, activation functions, learning rules, ANN architecture, training and testing procedures are presented in the chapter. Certain important issues in relation to network generalization are also discussed in the subsequent sections.
3.2.1 Basic Concepts and Theoretical Background

ANNs are massively parallel distributed processor made up of simple processing units, which is capable of storing experiential knowledge gained by the process of learning and making it available for future use (Haykin, 1999). ANNs use mathematical formulations to model the operations of the nervous system. Although significant advance has been achieved in the area of conventional expert systems for mimicking human intelligence, there is still a long way to go for the current computational techniques before realizing the capability of carrying out certain man-dependent tasks. Nowadays, when silicon chips are $10^6$ times faster than the operation speed of biological cells, a typical human activity, e.g., face recognition with a noisy background, which only requires the order of 100 ms to be accomplished in the brain will still run on a serial-programming computer for days. The reason for this can be explained by several factors: the massively connected synapses among the biological neurons, the highly parallel computing structure, and the imprecise information processing capability of the brain (Russel, 1979). Therefore, neural networks are constructed as simplified mathematical models, which can resemble the organizational principles of the human brain on the level of microscopic biological models. By taking advantage of these principles, neural networks are considered as a kind of new generation intelligent information processing systems. They have the advantageous capabilities of learning from training data, recalling memorized information, and generalizing to the unseen patterns. On the other hand, a neural network may be considered as an adaptive system that progressively self-organizes in order to approximate the solution, making the problem solver free from the need to accurately and unambiguously specify the steps towards the solution. Moreover, ANNs have the ability to progressively improve their performance on a given task by learning. These capabilities do show great potential in such application areas as signal processing (Amari et al., 1998), pattern recognition (Yegnanarayana, 1994), classification (Reby et al., 1997) and prediction (Faraway et al., 1998; Azamathullah et al., 2008). The majority of the ANN model used in the available literatures for prediction is a supervised multi-layered feedforward network using BP learning algorithm.
3.2.2 Structure of a Biological Neuron

Neurons are the elementary information processing units of the nervous system. All the neurons possess’ identical structural organization irrelevant of their sizes. The schematic structure of a biological neuron is depicted in Figure 3.1. The basic components of biological neurons are cell body (or soma), nucleus, dendrites, axon and synaptic terminals. The dendrites are fibers which originate from cell body and receive signals from outside or other neurons. The cell body is the place for message processing which generate brief pulses that propagate rapidly via axon to the next neurons, or to the outside. Nucleus contains the genetic material in the form of DNA. Synapses are the junctions that allow signal transmission between the axons and dendrites.

![Fig 3.1 Biological Neuron](image)

The working process of a biological neuron is to collect signals from the outside and send them to the information processing area, when sufficient input is received (i.e., a threshold is exceeded), the neuron generate an action potential or spike and transmits to other neurons. Billions of those kinds of neurons construct a powerful brain. In terms of the rules of biologic neurons, a heuristic model of artificial neural networks operates based on the following assumptions (Fausett, 1994):

(i) Information is processed at neurons.

(ii) Signals are passed between neurons in different layers over connection links (paths).

(iii) Each connection link has a weight, which multiplies the signal transmitted.

(iv) There is a transfer function in each neuron to determine the value of the outgoing signal.
Thus, a neural network is characterized by: (a) a pattern of connections between the neurons that link specific inputs to outputs (the architecture), (b) a procedure on creating the “weights” (its training/learning algorithm), and (c) an activation function (also called transfer function). Although all neural networks share these characteristics, there are many different training algorithms developed for estimating the weights.

### 3.2.3 Artificial Neuron Modeling

An artificial neuron is an information-processing unit that is fundamental to the operation of an artificial neural network. Figure 3.2 shows the block diagram of a neuron. It consists of a set of synapses or connecting links, each of which is characterized by a weight or strength. The synaptic weights may be positive or negative. A negative weight value will inhibit the activity of the connected-to unit, whilst a positive weight value will excite the connected-to unit. A signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{jk}$. The weights $w_{jk}$ can be interpreted as the importance of the input $x_j$ to neuron $k$.

Mathematically, a neuron $k$ may be represented by the following pair of equations:

$$z_k = \sum_{j=1}^{n} w_{jk} x_j$$  \hspace{1cm} (3.1)

and

$$y_k = \phi(z_k + b_k)$$  \hspace{1cm} (3.2)
where, $x_1, x_2, \ldots, x_n$ are the input signals, $w_{1k}, w_{2k}, \ldots, w_{nk}$ are the synaptic weights of neuron $k$; $z_k$ is the linear combiner output due to the input signals; $b_k$ is the bias; $\phi(\cdot)$ is the activation function and $y_k$ is the output signal of the neuron. The bias is an external parameter of an artificial neuron. It refers to the portion in the neuron model where input is a constant. The bias has the effect of increasing or lowering the net input of the activation function.

The computing ability of network units or neurons are typically restricted to a rule for combining input signals and an activation rule that takes the combined input to calculate an output signal. Output signals may be sent to other units along weighted connections. The weights associated to a connection, usually excite or inhibit the signal that is being communicated. The pattern of connectivity refers to the way in which the units are connected. In a type of network, each unit may be connected to all other units; in another network model, units may be arranged into an ordered hierarchy of layers where connections are only allowed between units in immediately adjacent layers. Again, other networks allow feedback connections between adjacent layers, or within a layer, or for units to send signals back to themselves.

### 3.2.4 Activation Function

The activation function or transfer function is used for limiting the amplitude of the output of a neuron. It transforms input data from any domain of values into finite range of output values. The activation function may be linear or non linear. Linear, logistic sigmoid, hyperbolic tangent sigmoid, ramp and step function are the examples of activation function.

Linear transfer function passes activation level directly as the output. It is used when the input-output relationship is linear. The logistic sigmoid function is the most commonly used activation function. This function is especially advantageous to use in neural networks trained by back-propagation algorithms because of the differentiable property. It maps the input from the range $-\infty$ to $+\infty$ onto the output in the interval $[0, 1]$. The hyperbolic tangent sigmoid function is similar to the sigmoid function but produces output values in the range of $[-1, 1]$.

### 3.2.5 Learning in Neural Networks

A key feature of neural networks is an iterative learning process in which data cases are presented to the network and the weights of its component connections are
adjusted in each iteration so as to bring the output of the network closer to a desired output or problem solution. There are two fundamental learning paradigms of neural networks, which are supervised learning and unsupervised learning method (Haykin, 1999).

In supervised learning, network is defined with a dataset of input/output pairs. The learning process updates the weights at each training step so that, for a given input, an error measure between the network’s output and the desired known target value is reduced. In contrast, unsupervised learning rules do not use target values, and even the target values are not available. They use the inputs information to categorize the input patterns.

A third commonly used form of training makes use of the concept of reinforcement learning. In this case the value of training input/output pairs is not a measure of the difference between the desired and the obtained value, as in supervised learning, but rather, an evaluation of the result as ‘wrong’ or ‘right’ result.

Learning rules are mathematical algorithms that dictate how the connection weights of a neural network are altered during learning. This is an approximation of biological function as our knowledge of biological learning systems is incomplete. There are a number of learning rules for neural networks. Some of the important learning rules (Haykin, 1999; Lamba, 2008) are discussed next.

(i) **Hebbian Learning**: It is the first learning rule employed in neural networks that was introduced by Donald Hebb (1949). According to Hebb’s rule, if two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously) then the strength of that synapse is selectively increased. On the other hand, if two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated. Hebbian learning is an unsupervised learning algorithm.

(ii) **Competitive Learning**: In competitive learning, those neurons which respond strongly to input stimuli have their weight updated. When an input pattern is presented, all the output neurons of a neural network compete among themselves to become active and the winning neuron undergoes weight adjustment. Thus, a single output neuron is active at any one time.

(iii) **Delta Rule** (Widrow-Hoff Learning Rule): The Delta rule is a variation of Hebb's Rule. This rule is commonly referred to Widrow-Hoff learning rule or least mean
square (LMS) learning rule. This rule works only for continuous activation functions and in the supervised training mode. It is one of the most commonly used learning rules in the field of neural networks. This rule is based on the simple idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output. The training process is to change the network weights so as to minimize the mean squared error for the network. In this rule, the adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse. The delta error in the output layer is transformed by the derivative of the transfer function and is then backpropagated into the previous neural layers one at a time to adjust input connection weights. The delta rule can be applied for a single output unit or several output units. For single output unit, the weight correction according to delta rule given as

$$\Delta w_{jk} = \eta (t_j - y_j)x_j$$

where, $\eta$ is the learning rate parameter, $t_j$ and $y_j$ are the target and predicted values for the input, $x_j$.

(iv) Gradient Descent Rule: This rule is similar to the Delta Rule because the gradient of the error is used to update the connection weights. In gradient descent method, weights are updated using the following expression:

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

where, $\eta$ is the learning rate parameter and $\frac{\partial E}{\partial w_{jk}}$ is the error gradient with reference to the weight $w_{jk}$. If all the weights are updated using this same formula, then this amounts to moving in the direction of steepest descent along the error surface - hence the name, gradient descent. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight.

3.2.6 Neural Network Architecture

There are many classes of neural network architectures, classified according to the learning mechanisms and structural organizations, however, mainly three
fundamentally different classes of networks architectures are identified and discussed in this section.

(i) Single Layer Feedforward Network

This type of network comprises of two layers, namely the input layer and the output layer, as shown in Figure 3.3.

![Single Layer Feedforward Neural Network](image)

The neurons in the input layer receive the input signals and the output layer neurons produce the output values. The information flows from the input layer neurons to the output layer neurons but not vice-versa. Such type of network is called feedforward. Although there are two layers, the input layer neurons just transmits the signals to the output layer and do not perform any computation. The computation is performed by the output layer nodes only. Thus, the network is named as single layer feedforward network (Haykin, 1999).

(ii) Multi Layer Feedforward Network

Multi layer feedforward network, as shown in Figure 3.4, consists of one input layer, one output layer and one or more intermediary layers called hidden layers. The hidden layers aid in performing useful intermediary computations before directing the input to the output layer. The input layer neurons are connected to the hidden layer neurons and the hidden layer neurons are connected to the output layer neurons. The input layer neurons receive the input signal and transmit to the hidden layer nodes, which perform intermediary computations and transmit the output signal to the output layer nodes. The hidden layer enables to extract higher order statistics (Haykin, 1999).
Recurrent Network

Recurrent neural networks (RNNs) differ from feedforward network architectures in the sense that there is at least one feedback loop as shown in Figure 3.5. There could also be neurons with self-feedback loop, i.e. the output of a neuron is fed back into itself as input. In RNNs, the directed cycle creates an internal state of the network that allows the network to exhibit dynamic temporal behavior (Haykin, 1999). A RNN can use its internal memory to process an arbitrary input sequences. The concept of memory by feedback was first described by Jeffrey Elman (Elman, 1990). Elman recurrent network is a simple recurrent network consisting of an input layer, one hidden layer and an output layer and an additional feedback or recurrent connection from the hidden neurons to a layer of context units consisting of unit delays. The input is propagated in a standard feed-forward fashion in the network at each time step and then a learning rule is applied. During a forward pass the context units are activated by the hidden neurons. Since the information propagate over the connections before the learning rule is applied the context units always maintain a copy of the previous neural state for one time step and then feed them back to hidden layer. The hidden neurons are activated both by the current input from the input layer and feedback (previous neural states of the hidden neuron) from the context units.
Among the different types of neural network architecture, multilayer feedforward type of networks are mostly used. Multilayer perceptron (MLP) network is the most commonly used multilayer feedforward type of network. Radial basis function (RBF) network is also a feedforward type of network. Both these networks are very popular and widely used network. MLP and RBF have been employed in the present investigation for prediction of scour depth around abutment; therefore a discussion on these two is presented in the following subsections.

### 3.2.7 Multilayer Perceptron Network

A multilayer perceptron network is also called feedforward neural network or back-propagation network is an important class of neural networks due to its simple topology and powerful approximation capability. Besides the input and output layer, there is at least one hidden layer. Each node in the input layer consists of linear transfer functions which just act as a buffer. The hidden and output layer neurons generally consist of sigmoid transfer functions. Many capabilities of the network, such as nonlinear function approximation, learning, generalization are due to
nonlinear activation function of neurons (Rajasekaran et al., 2014). An MLP network with one hidden layer is as shown in Figure 3.4.

Backpropagation (BP) is the most commonly used method for training MLP network. BP is a form of supervised learning where the error rate is back propagated from output to hidden, and then hidden to input layer links to update the weights to decrease error (BP algorithm is described in section 3.2.9). It has been proven that a MLP network with sufficient hidden nodes can approximate any nonlinear function to arbitrary accuracy (Hornik, 1989).

3.2.8 Radial Basis Function Network

A radial basis function neural network (Broomhead and Lowe, 1988) consists of only one hidden layer. The neurons in the hidden layer contain radial basis transfer functions. The output units of RBF network form a linear combination of the basis (kernel) functions computed by the hidden units. The RBF network training involves the determination of data clustering centers, $\mu_i$ ($i = 1, 2, \ldots, H$), where $H$ is the number of the centers/hidden nodes, and network connection weights, $w_{jk}$ (weight between $j^{th}$ hidden unit to $k^{th}$ output unit) from the hidden layer to the output layer. A schematic diagram of RBF network is shown in Fig. 3.6. Here, $x_1, x_2, \ldots, x_n$ represents the input parameter and $y_1, y_2$ are the two outputs. Similar to BP training, the RBF training establishes the map between process inputs and desired outputs by minimizing the errors between the desired outputs and the calculated outputs driven from the inputs and network learning.

![Fig. 3.6 Radial Basis Function Network Structure](image)
A RBF is any function $R$ that satisfies $R(x - \mu_i) = R(||x - \mu_i||)$, where $\mu_i$ represent centers. The norm $||.||$ is usually Euclidean distance. Centers can be selected by picking input vector randomly or by clustering algorithms such as $k$-means clustering (Hartigan et al., 1979) or fuzzy $c$-means (Bezdek, 1981). There are choices of RBF functions, such as Gaussian function, multiquadric function, inverse multiquadric function and thin plate spline.

3.2.9 MLP Training Algorithm

The first and most popular method for performing supervised learning is the BP algorithm (also known as the generalized delta rule). BP has emerged as the standard algorithm for the training of MLP networks. Gradient of the error with respect to weights for a given input propagates backwards through the network and weights updation is done. The back propagation algorithm was developed by Paul Werbos (1974) and rediscovered independently by Rumelhart et al. (1986a) and Parker (1985).

BP is essentially a search procedure that attempts to minimize the network error function, i.e., the error between target and predicted values. It minimizes the error by recursively adjusting the complex weights of the MLP using gradient descent technique. The gradient vector is calculated in the direction opposite to the flow of the output of the each node. Thus, BP algorithm determines the weights for an MLP network. It has been shown that a two-layer feedforward neural network with a sufficient number of hidden units can approximate any continuous function to any degree of accuracy (Cybenko 1989). This makes multilayered feedforward neural networks a powerful modeling tool. In a feedforward network, given a set of input patterns (observations) with associated known outputs (responses), the objective is to train the network, using supervised learning, to estimate the functional relationship between the inputs and outputs. The network can then be used to model or predict a response corresponding to a new input pattern. This is similar to the regression problem where there is a set of independent variables (inputs) and dependent variables (output), and the relationship between the two is found.

To accomplish the learning, some form of an objective function or performance metric is required. The goal of the objective function is to optimize the weights. The most common objective function used in neural networks is the sum of squared
differences between the predicted values and observed values defined by equation (3.3).

\[
E = \frac{1}{2} \sum_{p=1}^{N} \sum_{k=1}^{M} (t_{pk} - y_{pk})^2
\]  

(3.3)

where, subscript \( p \) refers to patterns with a total of \( N \) observations, subscript \( k \) refers to output unit with a total of \( M \) units, and \( t \) and \( y \) are observed and estimated responses, respectively. When the number of output parameter, \( k=1 \), as in the present case, the equation (3.3) becomes:

\[
E = \frac{1}{2} \sum_{p=1}^{N} (t_{p} - y_{p})^2
\]  

(3.4)

The BP training algorithm involves four stages, \( \text{viz.} \)
- Initialization of weights
- Feed forward
- Back propagation of errors
- Updating weights and biases

The various parameters used in this algorithm are as follows:
- \( x \): input training vector ; \( x = (x_1, \ldots, x_i, \ldots, x_n) \)
- \( t \): output target vector ; \( t = (t_1, \ldots, t_k, \ldots, t_m) \)
- \( \partial_k \) = error at output unit \( y_k \); \( \partial_j \) = error at hidden unit \( z_j \); \( \eta \) = learning rate ;
- \( v_{ij} \) = weights between \( i^{th} \) input unit and \( j^{th} \) hidden unit ; \( v_{0j} \) = bias on the hidden unit \( j \);
- \( w_{jk} \) = weights between \( j^{th} \) hidden unit and \( k^{th} \) output unit; \( w_{0k} \) = bias on output unit \( k \);
- \( z_j \) = output at hidden unit \( j \); \( y_k \) = output at output unit \( k \)

**Initialization of Weights**
Step 1: Initialize weights to small random values.
Step 2: While stopping condition is false, do steps 3-10
Step 3: For each training pair do steps 4-9

**Feed Forward**
Step 4: Each input unit \( (x_i, i=1, \ldots, n) \) receives the input signal \( x_i \) and transmits this signal to all units in the layer above i.e., hidden units.
Step 5: Each hidden unit \( (z_j, j=1, \ldots, H) \) sums its weighted input signals
applying activation function

\[ z_j = f(z_{inj}) \]

and sends this signal to all units in the layer above i.e., output units.

Step 6: Each output unit \((y_k, k=1, \ldots, M)\) sums its weighted input signals

\[ y_{ink} = w_{0k} + \sum_{j=1}^{P} z_j w_{jk} \]

and applies its activation function to calculate the output signals

\[ y_k = f(y_{ink}) \]

**Back Propagation of Errors**

Step 7: Each output unit \((y_k, k=1, \ldots, M)\) receives a target pattern corresponding to an input pattern, error information term is calculated as

\[ \delta_k = (t_k - y_k)f'(y_{ink}) \]

Step 8: Each hidden unit \((z_j, j=1, \ldots, H)\) sums its delta inputs from units in the layer above

\[ \delta_{inj} = \sum_{k=1}^{M} \delta_j w_{jk} \]

The error information term is calculated as

\[ \delta_j = \delta_{inj} f'(z_{inj}) \]

**Updation of Weight and Biases**

Step 9: Each output unit \((y_k, k=1, \ldots, M)\) updates its bias and weights \((j=0, \ldots, H)\)

The weight correction term is given by

\[ \Delta w_{jk} = \eta \delta_k z_j \]

And the bias correction term is given by

\[ \Delta w_{0k} = \eta \delta_k \]
Therefore, \( w_{jk} (\text{new}) = w_{jk} (\text{old}) + \Delta w_{jk} \), \( w_{0k} (\text{new}) = w_{0k} (\text{old}) + \Delta w_{0k} \)

Each hidden unit \((z_j, j=1, \ldots, H)\) updates its bias and weights \((i=0, \ldots, n)\)

The weight correction term
\[
\Delta v_{ij} = \eta \delta_j x_i
\]

The bias correction term
\[
\Delta v_{0j} = \eta \delta_j
\]

Therefore, \( v_{ij} (\text{new}) = v_{ij} (\text{old}) + \Delta v_{ij}, \quad v_{0j} (\text{new}) = v_{0j} (\text{old}) + \Delta v_{0j} \)

Step 10: Test the stopping condition. The stopping condition may be minimum error goal or number of epochs etc.

### 3.2.10 MLP Testing Algorithm

The testing algorithm for the network is given as follows:

Step 1: Initialize weights (from training algorithm).

Step 2: For each test input vector do steps 3-5

Step 3: For \(i=1, \ldots, n\) : set the activation of input unit, \(x_i\);

Step 4: For \(j=1, \ldots, H\):
\[
z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}
\]

Step 5: For \(k=1, \ldots, M\)
\[
y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}
\]
\[
y_k = f(y_{ink})
\]

### 3.2.11 Training RBF Networks

The learning algorithm of the RBF network employed in the present study works as follows:

Step 1: Input data are fed through the input layer nodes which act as a buffer.

Step 2: Centers are chosen by fixed centre selection at random, \(k\)-means and fuzzy \(c\)-means clustering algorithms. In fixed centre selection method, centers are
sub-sampled from the set of input data, i.e. they are restricted to the input data points. Whereas, in clustering methods, each centre is a representative of a group of data points.

Step 3: Width ($\sigma$) of the Gaussian function is determined by

$$\sigma = \frac{d_{\text{max}}}{\sqrt{2H}}$$

where, $H$ is the number of centers, $d_{\text{max}}$ is the maximum distance between the centers.

Step 4: The output of the hidden layer neurons with Gaussian transfer functions is computed as follows.

$$G(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

where, $r = ||x - \mu_i||$ ($x$ denotes the input data, $\mu_i$ denotes the centre of the function and $\sigma$ denotes its width)

Step 5: Weights between the hidden and output layer is derived by the following equation:

$$w = G^+d$$

where, $G^+$ is the pseudoinverse of matrix $G$, i.e.,

$$G^+ = (G^T G)^{-1} G^T$$

Step 6: Calculate the output of the neural network.

$$y_{\text{net}} = \sum_{i=1}^{H} w_i G(||x - \mu_i||)$$

where,

- $H$ = number of hidden nodes
- $y_{\text{net}}$ = predicted output of the network
- $w_i$ = weight between $i^{th}$ hidden unit and output node
- $||.||$ = Euclidean norm
- $G$ = radial basis function
3.2.12 RBF Testing Algorithm

The testing algorithm for a network is given as follows:

Step 1: Using the weight matrix obtained in training phase, the output of the network is calculated for a given input as follows:

\[ y_{net} = \sum_{i=1}^{N} w_i G(\|x - \mu_i\|) \]

Step2: Check whether the calculated output converges to the desired output.

3.2.13 ANN Design Issues

The design of the ANN plays an important role for successful application of the neural network. There are number of parameters to be set that can affect the performance of the network. Some of these parameters are the type of network that has to be considered, the number of layers and nodes in each layer and the connection weights. Other network design decisions include the selection of activation functions of the hidden and output nodes, the training algorithm, initial weights, data normalization method, training and test data sets and performance measures. In this section, the modeling issues of a neural network predictive model pertaining to fully connected feedforward networks are presented.

(i) **ANN Architecture**: The architecture refers to the organization of neurons and their connections. The architecture design is crucial in the successful application of ANNs because of its significant impact on a network information processing capabilities. Given a learning task, an ANN with only a few connections and linear nodes may not be able to perform the task at all due to its limited capability, while an ANN with a large number of connections and nonlinear nodes may overfit noise in the training data and fail to have good generalization ability. The main problem is that there is no systematic way to design a near-optimal architecture for a given task automatically.

An ANN is generally composed of layers of nodes. In the popular MLP, all the input nodes are in one input layer, all the output nodes are in one output layer and the hidden nodes are distributed into one or more hidden layers in between. In designing an MLP following variables must be decided on:
• the number of input nodes
• the number of hidden layers and hidden nodes
• the number of output nodes

The selection of these parameters is basically problem-dependent. The number of input nodes corresponds to the number of variables in the input vector. For causal forecasting, the number of inputs is usually transparent and relatively easy to choose. In a time series forecasting problem the number of input nodes corresponds to the number of lagged observations used to discover the underlying pattern in a time series and to make forecasts for future values. However, currently there is no suggested systematic way to determine this number. The number of output nodes is relatively simple to decide as it is directly related to the problem under study.

The hidden layer and nodes play very important roles for many successful applications of neural networks. Hidden layer allows the networks to detect the feature, to perform complicated nonlinear mapping between input and output variables. A single hidden layer is sufficient for ANNs to approximate any complex nonlinear function with any desired accuracy (Cybenko, 1989). Two hidden layer networks may provide more benefits for some type of problems (Barron, 1994). Several authors address this problem and consider more than one hidden layer (usually two hidden layers) in their network design processes. Srinivasan et al. (1994) used two hidden layers and this resulted in a more compact architecture which achieved a higher efficiency in the training process than one hidden layer networks.

There is no specific way to determine the best number of hidden nodes without training several networks and estimating the generalization error of each. If there are a few hidden units, then the training error and generalization error are very high due to underfitting and there occurs high statistical bias. If there are too many hidden units, then the training error is too low but still has high generalization error due to overfitting and high variance. Several rules have been proposed to avoid the over fitting problem, by restricting the number of hidden nodes. Lachtermacher et al. (1995) give a heuristic constraint on the number of hidden nodes. In the case of the popular one hidden layer networks, several practical guidelines exist. These include using ‘‘2n +1’’ (Lippmann, 1987; Hecht-
Nielsen, 1990), ‘‘2n’’ (Wong, 1991), ‘‘n’’ (Tang and Fishwick, 1993), ‘‘n/2’’ (Kang, 1991), where \( n \) is the number of input nodes. However none of these heuristic choices works well for all problems.

Although there exists many different approaches such as polynomial time algorithm (Roy et al., 1993), canonical decomposition technique (Wang et al., 1994) and pruning algorithm (Cottrell et al., 1995) for finding the optimal architecture of an ANN, these methods are usually complex in nature and are difficult to implement. Further, none of these methods can guarantee the optimal solution for all real life problems. The methods are either heuristic or based on simulations derived from limited experiments. There are two comparatively simple methods for designing ANN architectures: Incremental algorithms and decremental algorithms. In incremental algorithms, hidden layers and hidden neurons are added to a network of minimum size until the required precision is reached. This algorithm starts with a minimal number of hidden layers, nodes and connections and adds new layers, nodes and connections when necessary, during training, until the error is sufficiently small. Decremental algorithms start with a large network and eliminate unnecessary layers, nodes, and connections during training. The problem with these algorithms is that they start with excessively large networks, which slows down training. Furthermore, the units to eliminate and their elimination order have to be guessed correctly.

(ii) **Transfer Function:** The activation function or transfer function squashes the weighted sum of the input of a node to a specific range of output values. In general, the activation function introduces a degree of nonlinearity that is important to handle non-linear problems. Chen et al. (1995) identified general conditions for a continuous function to qualify as an activation function. Generally, any differentiable function can qualify as an activation function in theory. In practice, only a small number of ‘‘well behaved’’ (bounded, monotonically increasing, and differentiable) activation functions are used.

There are some heuristic rules for the selection of the activation function. For example, Klimasauskas (1991) suggested logistic activation functions for classification problems which involve learning about average behavior, and to use the hyperbolic tangent functions if the problem involves learning about deviations from the average such as the forecasting problem.
Generally, a network may have different activation functions for different nodes in the same or different layers (Schoneburg, 1990 and Wong, 1991). Yet, almost all the networks use the same activation functions particularly for the nodes in the same layer. While the majority of researchers use logistic activation functions for hidden nodes, there is no consensus on which activation function should be used for output nodes. Following the convention, a number of authors simply use the logistic activation functions for all hidden and output nodes (Chakraborty et al., 1992; Sharda et al., 1992).

(iii) Training Algorithm: The ANN training is an unconstrained nonlinear minimization problem in which weights of a network are iteratively adjusted to minimize the overall error between the predicted and desired outputs. There is no algorithm currently available to guarantee the global optimal solution for a general nonlinear optimization problem in a reasonable amount of time. There are many optimization methods, all of which suffer from the local optima problems and at most they can provide the best local optima if the true global solution is not available (Fletcher, 1987).

The most popularly used training method is the BP algorithm which is a gradient steepest descent method. For the gradient descent algorithm, a step size, which is called the learning rate, must be specified. The learning rate is a key parameter in BP training as it determines the magnitude of weight changes. It is well known that the steepest descent suffers the problems of slow convergence, inefficiency, and lack of robustness. Furthermore, it can be very sensitive to the choice of the learning rate. Smaller learning rates tend to slow the learning process while the network may oscillate in the weight space for larger learning rates.

One way to improve the original gradient descent method is to include an additional momentum parameter faster convergence and overcoming from small local minima (Rumelhart et al., 1986b). The standard BP algorithm with momentum is employed by most of the researchers. Since there are few systematic ways of selecting the learning rate and momentum simultaneously, the “best” values of these learning parameters are usually chosen through experimentation. As the learning rate and the momentum can take on any value between 0 and 1, it is actually impossible to do an exhaustive search to find the best combinations of these training parameters. Only selected values are considered by the researchers.
For example, Sharda and Patil (1992), tried nine combinations of three learning rates (0.1, 0.5, 0.9) and three momentum values (0.1, 0.5, 0.9).

(iv) **Data Rescaling/Normalization**: In the context of neural networks, data rescaling is also referred to as “data normalization” or “data standardization”. Data normalization often refers to rescaling input data to make all the elements lie in a specific range (e.g., between 0 and 1). According to Sarle (2002), gradient-descent methods such as standard BP are very sensitive to scaling. Normalization may have some benefits for the network training. The reasons being the following:

a) Rescaling of an input parameters can make the surface of the performance index smoother, make training faster and reduce the chances of getting stuck in local optima.

b) Rescaling of inputs into uniform range initially equalizes the importance of input variables. For example, if one input has an order of magnitude of 1,000 and the other is approximately 0.0001, so the weights associated with the first input variable are small but very large for the second input. In some algorithms, the contribution of second input to the network may be swamped by the contribution of the first one. So it is essential to rescale the inputs so that their variability reflects their importance.

Although it is not necessary to rescaling the target values for the network with an unbounded transfer function in the output layer, in some cases such as network with bounded transfer function in the output layer, target variable also need be re-scaled.

(v) **Selecting Data**: One of the most important factors for training neural networks is the availability and the integrity of data. They should represent all possible states of the problem considered, and they should have enough patterns for validating the performance of the network. The training and testing data must be representative of the problem.

The issue here is the division of the data into the training and test sets. Although there is no general solution to this problem, several factors such as the problem characteristics, the data type and the size of the available data should be considered in making the decision. The literature offers little guidance in selecting the training and the test sample. Most authors select them based on the rule of 90% vs. 10%, 80% vs. 20% or 70% vs. 30%, etc.
(vi) Performance Metrics: Although there can be many performance measures for an ANN predictor like the modelling time and training time, the ultimate and the most important measure of performance is the predictive accuracy it can achieve for unseen data. An accuracy measure is often defined in terms of the predicting error which is the difference between the actual (desired) and the predicted value. There are a number of error measures in literature and each has advantages and limitations (Makridakis et al., 1983). The measures can be placed in three categories: (a) absolute errors, (b) squared errors and (c) directional errors.

Both absolute error and squared error measures treat the positive and negative errors the same. The absolute error method gives same weights for all errors whether it is small or large error. The squared error method gives more weight to the large errors, and the less weight to the small errors. Because of this, the measures obtained from the predictive techniques cannot always be the same. Directional errors occur when the sign of predicted values is not consistent with the corresponding sign of the target values. A measure of directional error is the number of pairs in which the predicted values and target values have different signs. Since each category of measure emphasizes a specific property of errors, different measures of accuracy could lead different conclusions.

The most frequently used are:

- Mean absolute error (MAE) \[ \frac{1}{N} \sum |t_p - y_p| \]
- Mean squared error (MSE) \[ \frac{\sum (t_p-y_p)^2}{N} \]
- Root mean squared error (RMSE) \[ \sqrt{\frac{\sum (t_p-y_p)^2}{N}} \]
- Mean absolute percentage error (MAPE) \[ \frac{1}{N} \sum \left| \frac{t_p-y_p}{y_p} \right| \times 100 \]

where, \( t_p \) is the target value, \( y_p \) is the actual value and \( N \) is the number of patterns.

Coefficient of determination, \( R^2 \): It is another most commonly used criterion of model performance. It gives the proportion of the variance of one variable that is predictable from the other variable. It is a measure that allows us to determine how certain one can be in making predictions from a certain model. The coefficient of determination represents the percent of data that is closest to the line of best fit.
**Correlation coefficient, R:** Correlation is a measure of the strength and direction (i.e., when the actual value increases, does the predicted value increase and vice versa) between two variables. The values for $R$ range from zero to one. The closer the correlation value is to one, the more correlated the actual and predicted values.

(x) **Sensitivity Analysis:** Sensitivity analysis provides the importance or relevance of each input parameter to the output. In models involving many input variables, sensitivity analysis is an essential ingredient of model building and quality assurance. Sensitivity refers to how a neural network output is influenced by its input and/or weight perturbations. A large sensitivity to a parameter implies that the system’s performance can drastically change with small variation in the parameter. Similarly, a small sensitivity implies slight change in the performance (Shojaeefard et al. 2013). Some of the sensitivity tests methods of ANN model are: “partial derivatives” method, “weights” method, “Profile” method and “backward stepwise” method. In “partial derivatives” method, the partial derivatives of the output is calculated according to the input variables (Dimopoulos et al., 1995). “Weights” method is a computation using the connection weights (Garson, 1991). In “Profile” method, an input variable is successively varied while the others are kept constant at a fixed value (Lek et al., 1996). The “backward stepwise” method works by eliminating one input variable at a time (Gevrey et al., 2003). Backward stepwise method is the most commonly used method which is employed in the present study.

### 3.3 Evolutionary Methods

The idea of evolutionary computing was introduced in 1960 by I. Rechenberg in his work *Evolutionary strategies*. Lawrence J. Fogel and John Henry Holland in the United States (Fogel et al., 1966), and Ingo Rechenberg and Hans-Paul Schwefel in Germany (Rechenberg et al., 1973) laid the theoretical foundations of evolutionary computing. John Holland showed how the evolutionary process can be applied to solve a wide variety of problems using a highly parallel technique that is now called the genetic algorithm (Holland, 1975, 1992). To know the background and differences between the relatively new evolutionary method, i.e., GEP and its predecessor, a brief introduction on GAs and GP are presented in this section.
3.3.1 Genetic Algorithm

GAs are optimization and heuristic search techniques and were inspired by the Darwinian evolution of population and are subject to operators of reproduction, crossover (also called recombination), mutation and inversion in a selective environment where the fittest survive (Holland, 1975). GAs work simultaneously on a set (population) of potential solutions (individuals) to the problem called chromosomes. The chromosomes of GAs are simple replicators, i.e., they function simultaneously as genome and phenome. The fitness of each individual is evaluated based on some fitness (or objective) function. Based on the fitness value the individuals are selected for the next generation. Chromosomes with the higher fitness are given a greater chance to select. Then, the individuals will conduct alterations similar to the natural genetics crossover and mutation. The process is repeated until some predetermined termination criteria is satisfied. The solution is expressed by the best chromosome in the final population. Fig. 3.7 shows a flowchart of operational processes of GAs.

GAs are computationally simple yet powerful enough to provide a robust search for difficult combinational search problems in complex spaces, without being stuck in local extremes (Goldberg, 1989). They have proven useful in areas ranging from stock market analysis to satellite design due to their robustness, speed, efficiency and flexibility.

Therefore, GAs are powerful alternative tools to traditional optimization methods. GAs have been successfully used in many fields, including Mathematics, Engineering, Computer Science, Biology, Finance, Physics, and Economics. Scheduling (Omara and Arafa 2010), function optimization (Goldberg and Samtani, 1986), prediction (Van Batenburg et al., 1995; Wang et al., 2007), designing (Li et al., 2004) are some of the important applications of GAs. These tools have also been used to assist with modeling, classification, control and data clustering.
GA consists of the following steps (Valluru et al., 2011, Rajashekaran et al., 2014):

**Step 1: Random Initialization of Population:** Initialize a population of \( n \) chromosomes. The chromosomes represent a set of possible solutions and are constituted by a group of gene segments. The length of each chromosome is fixed. The choice of population size is important as for too large size of population, the algorithm does not differ much from an exhaustive search; and for too small size, the algorithm may not reach the optimal solution.

**Step 2: Evaluate the Fitness of Individuals in the Population:** Evaluate the fitness \( f_i \) of each chromosome \( i \) in the population. The fitness function \( f \) is derived from the objective function and is used in the next step to create a genetic pool.

![Flowchart of Genetic Algorithm](image)

Fig. 3.7 Flowchart of Genetic Algorithm
Step 3: New Population Generation: The new population is generated with an objective to obtain individuals having high fitness values. Thus, after evaluating the fitness of the individuals of the population, a new population is created. This is accomplished by performing three stages. Initially, a selection stage is applied (reproduction operation) to obtain mating pool (which contains the fittest individuals) and other genetic operators like crossover and mutation are used, which change the population. The working mechanism is briefly explained below:

(i) Selection/Recombination stage: Select two parent chromosomes according to their fitness (better the fitness, bigger the chance to be selected). There are many selection methods viz. Roulette-wheel, Boltzmann, Tournament, Rank selection and Steady state selection. Roulette-wheel selection is the most commonly used selection method, where the probability of an individual to be selected is

$$P_i = \frac{f_i}{\sum_{k=1}^{n} f_k} (k = 1, 2, \ldots, n)$$

where, $f_i$ is the fitness of $i^{th}$ individual, $n$ is the number of individuals in the population.

(ii) Crossover stage: After selecting the chromosomes, genetic crossover operator is applied to the mating pool to generate new individuals (which will retain the good features from the previous generation and which may also outperform their parents). Crossover is applied to the selected parents with a crossover probability. The probability varies from 0 to 1. This is calculated in GA by finding out the ratio of the number of pairs to be crossed to population size. There exist many types of crossover operations in GA such as, one-point, two-point and uniform crossover.

In one-point crossover, a cross-site is selected randomly along the length of the mated strings and the bits next to the cross-sites are interchanged as shown below:
Strings after mating:

| Parent 1: 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| Parent 2: 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

Strings after mating:

| Child 1: 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| Child 2: 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

In two-point crossover, two crossover points are chosen randomly and the contents within the crossover points are interchanged. An example of a two-point crossover operation is given next:

Strings before mating:

| Parent 1: 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| Parent 2: 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |

Strings after mating:

| Child 1: 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| Child 2: 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

The uniform crossover operator allows the parent chromosomes to be mixed at the gene level rather than the segment level. Consider the two parents mentioned in the previous cases are selected for crossover. If the mixing ratio is 0.5, then each child will contain half of the genes from parent 1 and remaining half from parent 2. The possible set of offspring after uniform crossover will be:

| Child 1: 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| Child 2: 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

(iii) *Mutation stage:* Mutation is a genetic operator that introduces auxiliary changes to a chromosome. It maintains genetic diversity from one
generation of a population of chromosomes to the next. Mutation alters one or more gene values in a chromosome. It is implemented with a *mutation rate*, which is used to calculate the number of gene values to be altered. Mutation can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation helps to prevent the search from stagnating into local optimum. The mutation operators are of many types which are flip-bit, boundary, uniform, non-uniform and Gaussian. Flipping bits is the simplest one which is applicable to binary string chromosomes only, where values of chosen bits are inverted, i.e., from 0 to 1 and vice versa.

**Step 4:** Evaluate fitness of the new generated population. If the end condition is satisfied, stop and return the best solution from current population, else GO TO step 2.

The genetic algorithm involves probabilistic steps for at least three points in the algorithm, namely

(i) Creating the initial population,

(ii) Selecting individuals from the population on which to perform each genetic operation (e.g., reproduction, crossover), and

(iii) Choosing a point (i.e., the crossover point or the mutation point) within the selected individual at which to perform the genetic operation.

Due to the probabilistic nature of the GA, it may be required to independently execute the algorithm multiple number of times in order to obtain a satisfactory result for a given problem.

### 3.3.2 Genetic Programming

GP is an extension of the conventional GA in which computer programs are considered as the individuals in the population (Koza, 1992). The search space in GP is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. The functions can be standard arithmetic operations, mathematical functions or domain specific functions.
The GP development process involves the determination of the set of terminals and primitive functions, fitness measure, parameters for controlling the run; and the method for designating a result and the criterion for terminating a run (Koza, 1992).

The first major step in developing GP model is to identify the set of terminals. The terminals are generally the input parameter of the problem.

The second major step in preparing GP is to identify the set of functions that are to be used to generate the mathematical expression to fit the training sample. Each function should accept arguments of any value and data type that may be returned by other functions in the function set. They should also accept any value and data type that may be assumed by any terminal in the terminal set. The set of terminals and the set of functions are the basic elements with which GP attempts to create a computer program to solve, or approximately solve the problem.

These first two major steps of GP correspond to representation step of the GA. The remaining three major steps correspond to the last three major preparatory steps for the conventional GA.

In summary, GP evolves computer programs to solve problems with the following three steps:

1. Generate an initial random population of computer programs called chromosome composed of the functions and terminals of the problem.

2. Repeat the following sub-steps until the termination criterion is reached:
   a. Execute each computer program in the population and evaluate its fitness value using the fitness measure.
   b. Create a new population of computer program by selecting the fitter programs and applying crossover operations.
      i. Reproduce the fitter individuals by copying it into the new population. The program with higher fitness will have more chance to be selected.
      ii. Create new individuals by exchanging randomly chosen parts of the selected individual using the crossover operation.

3. The best fit computer program evolved in the population during the run is considered as the result of the run of GP. This result may be a solution (or approximate solution) to the problem.
3.3.3 Gene Expression Programming

Gene expression programming is a relatively new evolutionary computing technique developed by Canadida Ferreira in the year 1999. GEP combines the advantages of both of its predecessors, GA and GP, and eliminates some of their shortcomings. GEP works like GA and GP that uses populations of individuals, selects them according to fitness, and introduces genetic variation using genetic operators. The basic differences between these algorithms are the nature of the individuals they deals with. In GAs, the individuals are fixed length symbolic strings (chromosomes); in GP the individuals are non-linear entities of different sizes and shapes (parse trees); and in GEP the individuals are encoded as symbolic strings of fixed length (chromosomes) which are then expressed as non-linear entities of different sizes and shapes (expression trees) (Ferreira, 2001a, 2001b, 2006).

The GEP development process starts with the generation of a random initial population. It consists of individual chromosomes of fixed length that may contain one or more genes. Each individual chromosome of the initial population is then evaluated and their fitness is computed using a fitness function. These chromosomes are then selected based on the fitness value using a selection method with the objective that fitter chromosomes have more chance of selection into the next generation. After selection, they are reproduced with some modifications carried out by genetic operators (e.g., mutation, inversion, transposition and recombination). Mutation is found to be the most effective genetic operator and in most cases is found to be the only operator used to modify the chromosomes (Khan et al., 2012). The new individuals are then undergoes to the same process of modification to create necessary genetic diversion and the process continues until the maximum number of generations is reached or the required accuracy goal is attained (Ferreira, 2001a, 2001b). The flowchart of a gene expression algorithm is shown in Figure 3.8.

The elemental aspects of GEP are discussed next. In GEP, several genetic operators are used for genetic modification of the chromosomes (Ferreira, 2006):

*Structural Organization of Genes: The genes of GEP are composed of a head and a tail. The head contains symbols that represent both functions and terminals, whereas the tail contains only terminals. For each problem, the length of the head ($l_h$) is chosen, whereas the length of the tail ($l_t$) is a function of $l_h$ and the maximum arity ($r$) of function and is evaluated by the equation:*
\[ l_t = l_h (r-1) + 1 \]

**Multigenic Chromosomes and Linking Functions:** The chromosomes of GEP are usually composed of more than one gene of equal length. For each problem, the number of genes and the length of the head are a priori chosen. Each gene is expresses as a sub-expression tree (ET) which interacts with one another with linking function (e.g., +, *) forming a more complex ET, if the number of output is one. For multiple outputs, each sub-ET evolves its respective output.

**Random Numerical Constants:** The initial random constants are generated during the creation of the initial population. In addition to the usual operators of GEP (i.e., mutation, transposition, inversion and recombination), a specific transposition and inversion are used to operate on numeric constants to create diversity of the numeric constants.

**Mutation:** This is the most important and influential of all the operators. In GEP modeling, mutation can take place at any position in a genome. However, the structural organization of the chromosomes must remain the same; that is, in the head of a gene, a function can be replaced by either another function or a terminal but, in the tail of a gene, terminals can only change into other terminals as there is no function in the tail. In this way all new individuals produced by mutation are structurally correct programs.

**Inversion:** In this operator a sequence within the head of a gene is selected and is inverted. It randomly chooses the chromosome, the gene to be modified and the start and terminal points of the portion of head to be inverted.

**Transposition and Insertion Sequence Elements:** The transposable elements of GEP are fragments of the genome that can be activated and jump to another place in the chromosome. In GEP there are three kinds of transposable elements.

(i) Short fragments with a function or terminal in the first position that transpose to the head of genes, except to the root (insertion sequence elements or IS elements).

(ii) Short fragments with a function in the first position that transpose to the root of genes (root IS elements or RIS elements).
(iii) Entire gene that transposes itself to the beginning of chromosomes. In contrast to the other forms of transposition, the transposon (i.e., gene) is deleted at the place of origin.

Figure 3.8 The Flowchart of a Gene Expression Algorithm (Ferreira, 2001a)

*One-point or two point recombination/cross-over*: In one-point recombination, parts of two chromosomes are interchanged over a randomly chosen point forming two new offspring chromosomes. In two-point recombination, two parent chromosomes are paired and two points are randomly chosen as cross-over points. The gene segment
within the cross-over points is then exchanged between the parent chromosomes, forming two new offspring chromosomes.

*Gene recombination:* In gene recombination, entire genes are exchanged between two parent chromosomes, forming two offspring chromosomes containing genes from both parents. The exchanged genes are randomly chosen and occupy exactly the same position in the parent chromosomes.

GEP has been recently applied in many fields including image compression (Gempeler, 2006), stage discharge relationship (Güven et al., 2010), sediment transport (Ghani et al., 2011), hydraulic engineering (Khan et al., 2012; Mohammadpour, 2013; Azamathulla et al., 2013).

### 3.4 Chapter Summary

In recent years, SC techniques have been widely applied in various application domains including water resource engineering. Amongst the soft computing techniques, ANN and GEP are found as the most popular technique for function approximation or prediction problems. An overview of the data-driven and evolutionary modeling was presented in this chapter. Various important issues in relation to ANN modeling for prediction such as number of hidden layers and nodes, the number of input and output nodes, activation function, training algorithm, data normalization, measure of performance and sensitivity analysis in neural networks have been presented. This chapter also provided the background of GEP with a discussion of its predecessors (GA and GP). The basic working principles of the evolutionary computing models are illustrated. In the next chapter, a discussion on the data and the methodological aspects of the present study for estimation of scour at abutments is presented.