Chapter 2: Soft Computing Techniques Employed

2.1 Introduction
Data-driven techniques can be considered as an approach to model the data that focuses on use of the Machine Learning methods for developing models which would complement the ‘knowledge-driven’ models based on extraction of the knowledge or the physical behavior (Solomatine and Ostfeld, 2008).

In the last three decades or so, Artificial Intelligence (AI) has proved to be a promising tool for solving complex scientific problems in various fields. Out of several methods under the domain of AI and data-driven techniques, most frequently used by the scientific community is neural nets popularly known as Artificial Neural Network (ANN). Another exciting and upcoming technique is Genetic Programming (GP), inspired from the natural evolution process and the Darwinian concept of ‘Survival of the Fittest (the Strongest)’. Thus GP is an evolutionary algorithm-based methodology inspired by biological process that evolves computer programs for performing a user-defined task. Use of GP has grown significantly in the last decade or so in almost all areas of science and engineering. Literature reveals that the applications of GP in ocean related studies are very few and sparse as against the ANNs.

Both techniques are purely nonlinear and ideally suited for employing them in the studies related to natural phenomena involving complex (nonlinear) physical processes between many parameters.

2.2 Artificial Neural Networks
Artificial Neural Network (ANN) tries to mimic the working of human brain. It offers a powerful and distributed computing architecture equipped with significant learning abilities. Since last three decades or so, Artificial Neural Networks (ANNs) are employed in the field of Civil Engineering for simplification of complex problems involving large number of computations.

The first artificial neuron was developed in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts, but the technology available at
that time did not allow them to do further developments (McCulloch and Pitts, 1943). Neuron is a minute processor that receives data, processes them and sends a signal to other connected neurons. Human brain has about 100 billion neurons of many distinct types arranged in many layers. A biological neuron model in Fig. 2.1 is shown to understand the working of an individual neuron.

Fig. 2.1 Working of a Biological Neuron

Neuron has four main parts: (1) cell body that processes and generates impulses, (2) dendrites which are the signal receivers for accepting signal from other connected neurons, (3) axons which send the message triggered by the neuron to the other connected neurons, and (4) synaptic terminals that entail excitatory or inhibitory reactions of receiving neuron. Axons and dendrites are the communication links of neurons. Axons carry the information (Data) from other connected neurons to the cell body (Soma) of a neuron under consideration through synapses. Vector product of all weights ($w_i$) and inputs ($p_i$), and bias ($b_i$) vectors are summed up in soma to get the output at the neuron. Outputs of all neurons in a particular layer constitute the output of that layer. Neuron emits or releases signal only if the weighted sum of inputs is greater than a threshold value (otherwise the signal decays and dies). Thus two operations are performed in soma: (i) computation of weighted sum of all of the inputs, and (ii) conversion of the output of this summation in terms of certain threshold. Fig. 2.2 shows these concepts.
An overview of Artificial Neural Network and its working is given in the following subsection.

### 2.2.1 Fundamentals of an ANN

ANN constitutes a computer program designed to learn in a manner similar to the human brain. A perceptron (an artificial neuron) weighs and sums up the inputs and compares the result with a predefined threshold. There are two types of weights in an ANN: the first between the inputs and the hidden layer/s, and the other between the hidden layer/s to the output layer. Following factors are considered for building up an ANN:

1. The starting values (weight and bias initialization)
2. Number of hidden layers
Designing a neural net is considered as an art rather than a science due to method of trials used for its design and generally it is a time consuming process. To accomplish good prediction performance by applying an ANN the raw data must be scaled between the upper and lower bounds of the transfer function (say, between 0 and 1 or -1).

### 2.2.2 Working of an ANN

The simplest form of an ANN consists of at least two layers: the input and the output layers. Input layer neurons receive the inputs and output layer neuron/ neurons yield the network output (desired result). ANN works similar to the human brain on two grounds: (i) acquiring knowledge from the input data via the network of neurons through a learning process, and (ii) inter-neuron connection strengths are utilized to store the acquired knowledge. The weights represent relative significance or strength of specific input parameters in the computation of output. Thus the network generated output heavily depends on the type of activation functions (also called transfer or squashing function) used. However, the hidden layer in an ANN confers strong learning ability to the ANN. Therefore a three layered ANN architecture is commonly used for most of the practical applications. An ANN model consists of matrices of weights and biases, which can be applied to obtain the desired predictions.

### 2.2.3 Advantages of ANNs

Strengths (advantages) of an ANN found in the literature are enlisted.

1. Universal Approximator
2. No *a-priori* model or knowledge required
3. Data tolerant
4. Better efficiency
5. Adaptive learning
6. Fewer Data required
2.2.4 Drawbacks of ANNs

In spite of the advantages of the ANNs, drawbacks reported in the literature are summarized.

(1) Main drawback related to the application of ANNs has been stated that it is a ‘black box modelling technique’.

(2) Addition of too many hidden neurons and/or executing large number of iterations (epochs) often result into ‘over-fitting’ of the network. It means that the network learns too much or too well from the training data to capture, memorizes the noise or insignificant information from the data and generates inferior results.

(3) ANN depends heavily on the quantity and quality of the data that the algorithms employed are only as good as the data.

(4) Finalization of ANN architecture can be a time-consuming process since it involves trial and error method. There are no fixed rules for deciding number of neurons in the hidden layers and also for best data division into training, testing and cross-validation sets although we may follow rules of thumb by studying similar works by other researchers.

(5) In case of rare extreme events, ANN predictions are most likely to be unsatisfactory.

(6) In spite of taking all possible care, ANN models are prone to the problems of local optima, which is an important characteristic of nonlinear optimization.

2.2.5 Types of Neural Networks

ANNs are two main types based on the flow of data or information: (1) Feed-Forward Networks, and (2) Feed-Back Networks.

Based on number of layers in the network, we have (i) two-layered Network, and (ii) multi-layered Network. Generally the former has only input and output layers without any hidden layer, while the latter has one or more hidden layers along with input and output layers. The input layer receives inputs in the form of an input matrix and passes its output to the hidden layer. Hidden layer analyzes these outputs and sends the output to the output layer. The error between the targets and outputs is determined and minimized by distributing the error to adjust (change) the weights and biases for the next epoch. This continues until the user defined stopping criterion is satisfied. The
data flows forward while the error is propagated backwards in a feed-forward error back-propagation network (Called as FFBPNN). Refer Fig. 2.3

![Typical Three-Layered Feed Forward error Back-Propagation Neural Network (FFBPNN)](image)

**Fig. 2.3 Typical Three-Layered Feed Forward error Back-Propagation Neural Network (FFBPNN)**

**2.2.6 The ANN Employed in this Work**

Literature pertaining to most of the ocean related studies reveals that three-layered error-back-propagation type of neural network (FFBPNN) is commonly employed with single hidden layer as shown in Fig. 2.3. Therefore the same is developed for ANN Models for prediction of wave heights (Hs) and sea water levels (SWLs).

**2.3 Genetic Programming**

One of the nature inspired soft computing method Genetic Programming (GP) mimics the evolution processes of cross-over, mutation and reproduction. GP follows the Darwinian principle of ‘survival of the strongest (fittest)’. GP is similar to the more widely known technique of genetic algorithm (GA), but its solution is a computer program or an equation as against a set of numbers in the GA (Koza, 1992).

The Programs are evolved by determining their fitness in which the weaker (less fit or less useful) programs are replaced by the stronger ones; until either a desired number of generations are completed or any other fitness criterion is reached/ satisfied. Fitness is a measure used by GP to know how well a Program
has learned to predict the output using the features of learning domain (i.e. inputs). The process in Genetic Programming found in literature is given below:
1. Creation of Initial Population of Individuals (Programs or Equations)
2. Evaluation of Fitness of the Individuals
3. Selection of the strongest or the fittest ‘Parents’ using a tournament or ranking or truncation method.
4. Generation or Creation of ‘Offspring’ (through the genetic evolution processes of Cross over, Mutation, and Reproduction or Copy)
5. Replacement of weakest parents by the new stronger ones in the population
6. Repetition of steps 2 through 5.

The process is terminated as soon as the user defined termination criterion is satisfied, and output is given as the Final Program or Equation. The termination criterion can be the number of generations completed, or specific value of an error measure, or the time of run, etc.

The basic algorithm used by genetic programming consists of four main components: Population, Testing, Selection and Breeding. All of its knowledge regarding a Population is contained in the fitness values of the Individuals. The fitness function measures how close an Individual is to the solution and using these values, the algorithm differentiates between the Individuals during the selection process. This creates a bias towards Individuals with better fitness and the algorithm improves the overall fitness of each successive Population. Hence highly fit individuals from the Population live, while the less or poorly fit individuals die (get eliminated from the Population). Genetic evolution operator of ‘crossover’ selects two Parent Programs and exchanges (swaps) good fragments between the Parents to obtain two Children (Programs). ‘Mutation’ operator cuts good portion of one Parent Program and attaches (attaches) it to other Program or to the same Program and creates new Child Program. Addition of more fit program to the Population is known as the ‘Copy’ or ‘Reproduction’ operator.

### 2.3.1 Fundamentals of GP

Genetic programming achieves the goal of creating programs automatically by genetically breeding a population of new computer programs using the principle of Darwinian natural selection (popularly known as ‘survival of the fittest or strongest’) and biologically inspired evolution processes.
A population member in GP is a structured computer program consisting of functions and terminals. The functions and terminals are picked up from the sets of functions and terminals. Fig. 2.4 shows the fundamental concepts in case of a standard or tree genetic programming.

![Diagram of Tree Genetic Programming](image)

**Fig. 2.4 Fundamentals of Tree Genetic Programming**

A function set contains basic mathematical operators (+, -, *, /, etc.), logarithmic and trigonometric functions, Boolean logic functions (AND, OR, NOT, etc.), or user defined functions. The terminal set consists of the arguments for the function and may include numerical constants. The functions and terminals are selected randomly and put together to form a computer model in a specific structure depending on the type of GP. Many efficient variants of the GP are emerging through research in the area of computer software as well as hardware. Three relevant forms of GP representations are described here in brief.

1. **Standard or Tree Based GP (SGP or TGP)** correspond to the expressions (syntax trees) from a ‘functional programming language’ (Koza, 1992). In this type, Functions are located at the inner nodes; while leaves of the tree hold input values and constants.

2. **Linear Genetic Programming (LGP)** evolves sequences of instructions using an ‘imperative programming language’ or a machine language. The term ‘Linear’ refers only to the structure of the (imperative) program representation. Readers are referred to Banzhaf (1997, 1998), Brameier (2004), and Guven (2009).
The instructions in LGP are restricted to operations that accept a minimum number of constants or memory variables called ‘registers’ and assign the result to another register such as

\[
\begin{align*}
\text{register 2} &= \text{register 1} + \text{register 2} \\
\text{register 3} &= \text{register 2} \times \text{register 3} \\
\text{register 3} &= \text{register 2} / \text{register 3}
\end{align*}
\]

Fig. 2.5 shows tree and linear representation of function \( y = (x-2)^2 + (x-2)^3 \)

![Tree and Linear Data Flow Graphs for \( y = (x-2)^2 + (x-2)^3 \)](image)

(3) Automatic Induction of Machine Code Genetic Programming (AIMGP) writes computer programs for the given data and gives ‘Best Program’ as well as ‘Best Team’ results. It creates total 30 programs in a project and produces total five teams of 1, 3, 5, 7, 9 program combinations. (Nordin, 1997; Francone et al. 1999, Francone, 2004)

### 2.3.2 Process of Evolving New Programs

Three genetic or evolution operators – Crossover, Mutation, and Reproduction – are employed in genetic programming for developing the Model, consisting of the best fitting Program as described earlier. Different types of crossover operations are used in GP. Some main types are stated here. A single Point is selected for crossover in case of a one-point crossover and the instructions are swapped between two stronger Parents. In binary system, it is represented as:

**Parent 1**: 10111010111  
**Parent 2**: 100110111

**Child 1**: 10111110111  
**Child 2**: 100010111
In two-point crossover, two points are selected from the instructions of two stronger Parents and swapped between them to create two new Children or Offspring. Fig. 2.6 shows the concept of two-point crossover as an instruction block.

![Fig. 2.6 Concept of Two-Point Cross-Over in Genetic Programming](image)

Two-point crossover has sub-categories as ‘homogeneous’ or ‘non-homogeneous’, depending on the size of instruction blocks swapped between the Parents. In binary system, two-point homogeneous crossover can be explained in an example as:

**Parent A:** 111011011100  **Parent B:** 11000110000

**Child X:** 111110011100  **Child Y:** 11000011000

Similarly two-point non-homogeneous crossover can be shown as:

**Parent I:** 1011101100  **Parent II:** 1100011011101

**Child 1:** 1011101111100  **Child 2:** 1100011001

In an alternate way, these two types of two-point crossovers are of represented in Fig. 2.7 and 2.8

![Fig. 2.7 (a)Blocks of equal size selected from Parents for swapping](image) (NOP means No OPeration)
Fig. 2.7 (b) New programs (Children) created by swapping the Blocks

Fig. 2.7 Two-Point Homogeneous Block Cross-Over in GP

(a) Blocks of unequal size selected from Parents for swapping
(NOP means No Operation)

Fig. 2.8 Two-Point Non-homogeneous Block Cross-Over in GP

(b) New programs (Children) created by swapping the Blocks
Advantage of the genetic operator ‘Mutation’ is that it can operate even on a single parent. Any instruction or code or a block can be modified in a single parent to create a new child. Thus large number of individuals (Programs) can be generated with ease to increase the population and make the individuals more fit. Creation of multiple children from a single Parent in binary system is shown below.

**Parent:** 1100001111

**Child A:** 1111110000  **Child B:** 11000011111

As stated earlier, copy or reproduction is the genetic operator that finally adds stronger or fitter individual to the population for the next iteration.

### 2.3.3 Advantages of GP

The advantages of GP are listed below.

1. No *a-priori* assumption of any form of equation.
2. Dimensional Awareness: GP has major advantage in its ability to select input variables that have significant contribution to the model and disregard the others. Hence it substantially reduces dimensionality of input variables.
3. It can iteratively generate new programs until they reach a certain level of acceptance as per the selection criterion.
4. Flexible mathematical structures capable of identifying the non-linear relationship between input and output data sets (hence between the variables) that are modified in the iterative process by applying genetic operators are possible with GP.
5. Inherent functional input-output relationships provided by GP, can offer possible interpretations of the underlying processes.

### 2.3.4 Drawbacks of GP

There are also some drawbacks of genetic programming found in the literature as stated below.
Model development time increases with the increase in volume (or length) of data.

(2) It is susceptible for the problem of over-fitting just like ANNs.

(3) The model entirely depends on the initial population used.

(4) It does not work well with constants (variables with no dimensions)

### 2.3.5 LGP and AIMGP

The commercial software Discipulus (version 4.0) based on the AIMGP is employed in the present work.

Main characteristics of LGP and AIMGP in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LIST Processing – LISP), but the programs of an imperative language (like C or C++). This hastens the evolution process tremendously. The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Non-effective code can be removed from the created program to reduce the program length. Refer Fig. 2.9

![Individual Program Code](image)

**Fig. 2.9 Creation of Efficient Program Code in AIMGP**

Imperative programs define sequences of commands for the computer to execute. C++ is designed to be a compiled language, meaning that it is generally translated into machine language that can be understood directly by the system, making the generated program highly efficient.

### 2.3.6 Initial GP Parameters

Initial parameters used for genetic programming models in the present work are tabulated below.
It should be noted that after each iteration the program and population sizes as well as percentages of cross-over and mutation rates get automatically modified until the final model is created (on the basis of the Mean Squared Error in this software). Due to this reason, the initial parameters are not changed while conducting the runs (Projects) with genetic programming (i.e. with the software Discipulus).

### 2.4 Closure

For station specific predictions of phenomena involving complex (nonlinear) relationship between many variables; soft computing techniques, falling in the domain of data-driven modelling techniques are attractive alternatives to the traditional physics-based numerical models. Inventions in the software and hardware have made the soft computing techniques, especially GP very fast and efficient in arriving at the final model.

From the literature review also it is evident that the techniques of GP and ANN have enough strengths and advantages to try them for predictions of peak wave heights produced by the extreme events like hurricanes as well as sea water levels. Literature review reveals that many researchers found results by the GP models better than those with the ANN, although sometimes the results of both techniques are also found to be at par.

---

**Table 2.1: Initial Parameters for Genetic Programming Model**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Population Size</td>
<td>250</td>
</tr>
<tr>
<td>Program Size</td>
<td>Initial 80, Maximum 512</td>
</tr>
<tr>
<td>Cross-over Frequency</td>
<td>50%</td>
</tr>
<tr>
<td>Homologous Cross-over Frequency</td>
<td>95%</td>
</tr>
<tr>
<td>Mutation Frequency</td>
<td>95%</td>
</tr>
<tr>
<td>Block Mutation Rate</td>
<td>30%</td>
</tr>
<tr>
<td>Instruction Mutation Rate</td>
<td>30%</td>
</tr>
<tr>
<td>Instruction Data Mutation</td>
<td>40%</td>
</tr>
<tr>
<td>Function Set</td>
<td>+, -, X, /, sqrt, exp, trig</td>
</tr>
</tbody>
</table>