# 1

# Stock Market Modeling Using Genetic Programming Ensembles

Crina Grosan[1] and Ajith Abraham[2]

[1] Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş Bolyai University, Kogalniceanu 1, Cluj-Napoca, 3400, Romania, cgrosan@cs.ubbcluj.ro, http://www.cs.ubbcluj.ro/ cgrosan

[2] School of Computer Science and Engineering, Chung-Ang University, 221, Heukseok-Dong, Dongjak-Gu, Seoul, 156-756, Korea, ajith.abraham@ieee.org, http://ajith.softcomputing.net

**Summary.** The use of intelligent systems for stock market predictions has been widely established. This chapter introduces two Genetic Programming (GP) techniques: Multi-Expression Programming (MEP) and Linear Genetic Programming (LGP) for the prediction of two stock indices. The performance is then compared with an artificial neural network trained using Levenberg-Marquardt algorithm and Takagi-Sugeno neuro-fuzzy model. We considered Nasdaq-100 index of Nasdaq Stock Market and the S&P CNX NIFTY stock index as test data. Empirical results reveal that Genetic Programming techniques are promising methods for stock prediction. Finally formulate an ensemble of these two techniques using a multiobjective evolutionary algorithm. Results obtained by ensemble are better than the results obtained by each GP technique individually.

## 1.1 Introduction

Prediction of stocks is generally believed to be a very difficult task. The process behaves more like a random walk process and time varying. The obvious complexity of the problem paves way for the importance of intelligent prediction paradigms. During the last decade, stocks and futures traders have come to rely upon various types of intelligent systems to make trading decisions [1], [2], [7]. This chapter presents a comparison of two genetic programming techniques (MEP and LGP), an ensemble MEP and LGP, artificial neural network and a neuro-fuzzy system for the prediction of two well-known stock indices namely Nasdaq-100 index of Nasdaq$^{SM}$ [19] and the S&P CNX NIFTY stock index [20]. Nasdaq-100 index reflects Nasdaq's largest companies across major industry groups, including computer hardware and software, telecommunications, retail/wholesale trade and biotechnology [21]. The Nasdaq-100 index is a modified capitalization-weighted index, which is designed to limit domination of the Index by a few large stocks while generally retaining the capitalization

ranking of companies. Through an investment in Nasdaq-100 index tracking stock, investors can participate in the collective performance of many of the Nasdaq stocks that are often in the news or have become household names. Similarly, S&P CNX NIFTY is a well-diversified 50 stock index accounting for 25 sectors of the economy [13]. It is used for a variety of purposes such as benchmarking fund portfolios, index based derivatives and index funds. The CNX Indices are computed using market capitalisation weighted method, wherein the level of the Index reflects the total market value of all the stocks in the index relative to a particular base period. The method also takes into account constituent changes in the index and importantly corporate actions such as stock splits, rights, etc without affecting the index value.

MEP and LGP techniques are applied for modeling the Nasdaq-100 and NIFTY stock market indices so as to optimize the performance indices (different error measures, correlation coefficient and so on). Results obtained by MEP and LGP are compared with the results obtained using an artificial neural network trained using the Levenberg-Marquardt algorithm [5] and a Takagi-Sugeno fuzzy inference system learned using a neural network algorithm (neuro-fuzzy model) [3][15]. Neural networks are excellent forecasting tools and can learn from scratch by adjusting the interconnections between layers. Neuro-fuzzy computing is a popular framework wherein neural network training algorithms are used to fine-tune the parameters of fuzzy inference systems. We will build an ensemble between MEP and LGP using a multiobjective evolutionary algorithm (Non-dominated Sorting Genetic Algorithm II (NSGAII)). Results obtained by ensemble are then compared with the results obtained by MEP and LGP individually.

In Section 2, we briefly describe the stock marketing modeling problem. In Section 3, different connectionist paradigms used in experiments are presented. In Section 4, we formulate the ensemble between MEP and LGP followed by experimentation setup and results in Section 5. Some conclusions are also provided towards the end.

## 1.2 Modeling Stock Market Prediction

We analysed the Nasdaq-100 index value from 11 January 1995 to 11 January 2002 [19] and the NIFTY index from 01 January 1998 to 03 December 2001 [20]. For both indices, we divided the entire data into almost two equal parts. No special rules were used to select the training set other than ensuring a reasonable representation of the parameter space of the problem domain. The complexity of the training and test data sets for both indices are depicted in Figures 1 and 2 respectively.

**Fig. 1.1.** Training and test data sets for Nasdaq-100 Index

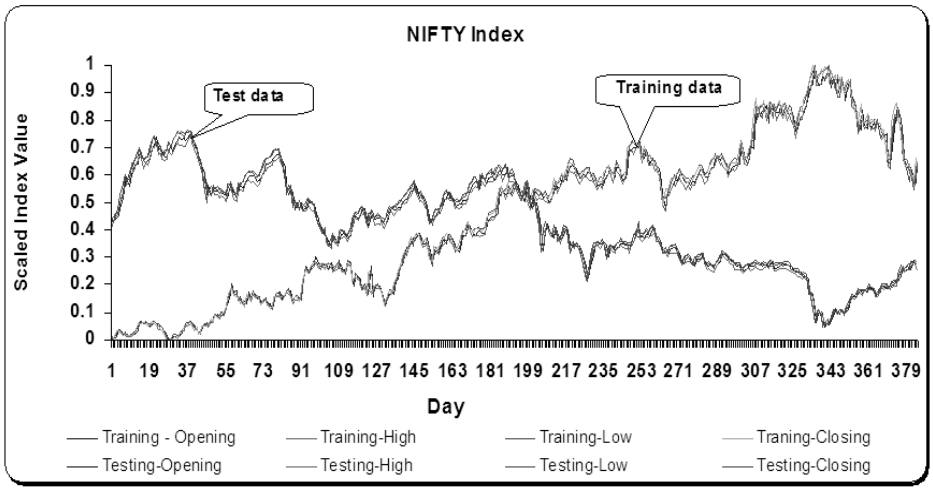
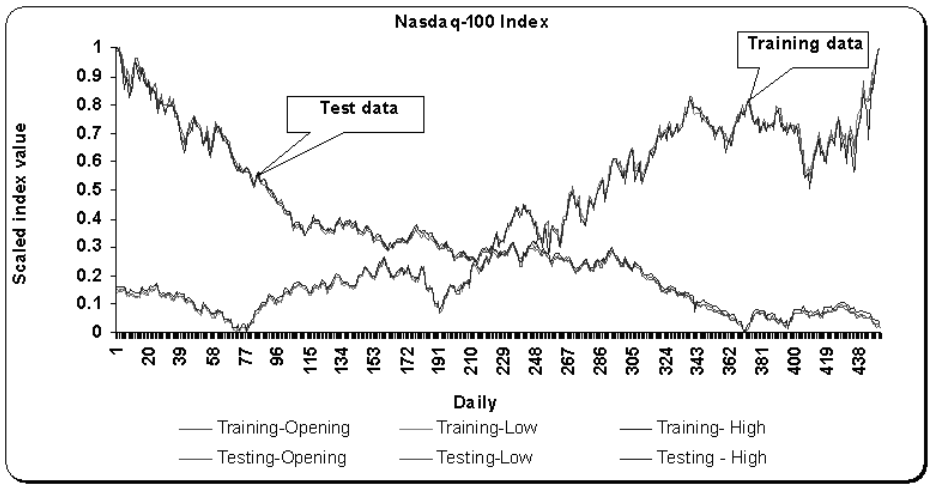



**Fig. 1.2.** Training and test data sets for NIFTY index




Our goal is to optimize several error measures: Root Mean Squared Error (RMSE), Correlation Coefficient (CC), Maximum Absolute Percentage Error (MAP) and Mean Absolute Percentage Error (MAPE):

$$RMSE = \sqrt{\sum_{i=1}^{N} |P_{actual,i} - P_{predicted,i}|} \tag{1.1}$$

$$CC = \frac{\sum_{i=1}^{N} P_{predicted,i}}{\sum_{i=1}^{N} P_{actual,i}},$$

(1.2)

$$MAP = \max\left(\frac{|P_{actual,\,i} - P_{predicted,\,i}|}{P_{predicted,\,i}} \times 100\right)$$

(1.3)

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{|P_{actual,\,i} - P_{predicted,\,i}|}{P_{actual,\,i}}\right] \times 100,$$

(1.4)

where $P_{actual,i}$ is the actual index value on day $i$, $P_{predicted,i}$ is the forecast value of the index on that day and $N$ = total number of days. The task is to have minimal values of RMSE, MAP and MAPE and a maximum value for CC.

## 1.3 Intelligent Paradigms

The different paradigms used in this chapter are described in this Section.

### 1.3.1 Multi Expression Programming (MEP)

MEP is a Genetic Programming variant that uses a linear representation of chromosomes. MEP individuals are strings of genes encoding complex computer programs. When MEP individuals encode expressions, their representation is similar to the way in which compilers translate *C* or *Pascal* expressions into machine code [4].

A unique MEP feature is the ability of storing multiple solutions of a problem in a single chromosome. Usually, the best solution is chosen for fitness assignment. When solving symbolic regression or classification problems (or any other problems for which the training set is known before the problem is solved) MEP has the same complexity as other techniques storing a single

solution in a chromosome (such as Genetic Programming [17], Cartesian Genetic Programming [18], Gene Expression Programming [11] or Grammatical Evolution [24]).

Evaluation of the expressions encoded into a MEP individual can be performed by a single parsing of the chromosome.

Offspring obtained by crossover and mutation are always syntactically correct MEP individuals (computer programs). Thus, no extra processing for repairing newly obtained individuals is needed. For technical details, the reader is advised to refer to the Chapter on Evolving Intrusion Detection Systems or please consult [22].

### 1.3.2 Linear Genetic Programming (LGP)

Linear genetic programming is a variant of the GP technique that acts on linear genomes [7][6]. Its main characteristics in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LISP), but the programs of an imperative language (like c/c ++). An alternate approach is to evolve a computer program at the machine code level, using lower level representations for the individuals. This can tremendously hasten the evolution process as, no matter how an individual is initially represented, finally it always has to be represented as a piece of machine code, as fitness evaluation requires physical execution of the individuals. The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Since different instructions may have different sizes, here instructions are clubbed up together to form instruction blocks of 32 bits each. The instruction blocks hold one or more native machine code instructions, depending on the sizes of the instructions. A crossover point can occur only between instructions and is prohibited from occurring within an instruction. However the mutation operation does not have any such restriction.

LGP uses a specific linear representation of computer programs. Instead of the tree-based GP expressions of a functional programming language (like *LISP*) programs of an imperative language (like $C$) are evolved.

A LGP individual is represented by a variable-length sequence of simple $C$language instructions. Instructions operate on one or two indexed variables (registers) $r$, or on constants $c$from predefined sets. The result is assigned to a destination register, for example, $ri = rj^*$ $c$.

Here is an example LGP program:
**void** LGP(**double** $v$[8])
[0] = $v$[5] + 73;
$v$[7] = $v$[3] – 59;
if ($v$[1] ¿ 0)
if ($v$[5] ¿ 21)
$v$[4] = $v$[2] .$v$[1];
$v$[2] = $v$[5] + $v$[4];

$v[6] = v[7]$ .25;
$v[6] = v[4] - 4$;
$v[1] = \sin(v[6])$;
if ($v[0]$ ¿$v[1]$)
$v[3] = v[5]$ .$v[5]$;
$v[7] = v[6]$ .2;
$v[5] = v[7] + 115$;
if ($v[1]$ ¡= $v[6]$)
$v[1] = \sin(v[7])$;
}

A LGP can be turned into a functional representation by successive replacements of variables starting with the last effective instruction. The maximum number of symbols in a LGP chromosome is 4 * Number of instructions. Evolving programs in a low-level language allows us to run those programs directly on the computer processor, thus avoiding the need of an interpreter. In this way the computer program can be evolved very quickly.

An important LGP parameter is the number of registers used by a chromosome. The number of registers is usually equal to the number of attributes of the problem. If the problem has only one attribute, it is impossible to obtain a complex expression such as the quartic polynomial. In that case we have to use several supplementary registers. The number of supplementary registers depends on the complexity of the expression being discovered.

An inappropriate choice can have disastrous effects on the program being evolved.

LGP uses a modified steady-state algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: Four individuals are randomly selected from the current population. The best two of them are considered the winners of the tournament and will act as parents. The parents are recombined and the offspring are mutated and then replace the losers of the tournament.

We used a LGP technique that manipulates and evolves a program at the machine code level. The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population space has been subdivided into multiple subpopulation or demes. Migration of individuals among the sub-populations causes evolution of the entire population. It helps to maintain diversity in the population, as migration is restricted among the demes. Moreover, the tendency towards a bad local minimum in one deme can be countered by other demes with better search directions. The various LGP search parameters are the mutation frequency, crossover frequency and the reproduction frequency: The crossover operator acts by exchanging sequences of instructions between two tournament winners. Steady state genetic programming approach was used to manage the memory more effectively.

### 1.3.3 Artificial Neural Network (ANN)

The artificial neural network methodology enables us to design useful nonlinear systems accepting large numbers of inputs, with the design based solely on instances of input-output relationships. For a training set $T$ consisting of $n$argument value pairs and given a $d$-dimensional argument $x$ and an associated target value $t$ will be approximated by the neural network output. The function approximation could be represented as

$$T = \{(x_i, t_i) : i = 1 : n\}$$

(1.5)

In most applications the training set $T$ is considered to be noisy and our goal is not to reproduce it exactly but rather to construct a network function that generalizes well to new function values. We will try to address the problem of selecting the weights to learn the training set. The notion of closeness on the training set $T$ is typically formalized through an error function of the form

$$\psi_T = \sum_{i=1}^{n} \|y_i - t_i\|^2 \tag{1.6}$$

where $y_i$ is the network output.

#### Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm [5] exploits the fact that the error function is a sum of squares as given in (1.6). Introduce the following notation for the error vector and its Jacobian with respect to the network parameters $w$

$$J = J_{ij} = \frac{\partial e_j}{\partial w_i}, i = 1 : p, j = 1 : n \tag{1.7}$$

The Jacobian matrix is a large p $\times$ n matrix, all of whose elements are calculated directly by backpropagation technique. The $p$ dimensional gradient $g$ for the quadratic error function can be expressed as

$$g(w) = \sum_{i=1}^{n} e_i \nabla e_i(w) = Je$$

(1.8)

and the Hessian matrix by

$$H = H_{ij} = \frac{\partial^2 \psi_{_T}}{\partial w_i \partial w_j} = \frac{1}{2}\sum_{k=1}^{n} \frac{\partial^2 e_k^2}{\partial w_i \partial w_j} = \sum_{k=1}^{n} \left( e_k \frac{\partial^2 e_k}{\partial w_i \partial w_j} + \frac{\partial e_k^{\partial} e_k}{\partial w_i \partial w_j} \right)$$

(1.9)

$$= \sum_{k=1}^{n} \left( e_k \frac{\partial^2 e_k}{\partial w_i \partial w_j} + J_{ik} J_{jk} \right) \tag{1.10}$$

Hence defining $D = \sum_{i=1}^{n} e_i \nabla^2 e_i$yields the expression

$$H(w) = JJ$$

$^T$ + D (11) (1.11)

The key to the LM algorithm is to approximate this expression for the Hessian by replacing the matrix $D$ involving second derivatives by the much simpler positively scaled unit matrix $\in I$. The LM is a descent algorithm using this approximation in the form

$$M_k = \left[ JJ^T + \in I \right]^{-1}, w_{k+1} = w_k - \alpha_k M_k g(w_k) \tag{1.12}$$

Successful use of LM requires approximate line search to determine the rate $\alpha_k$. The matrix $JJ^T$ is automatically symmetric and non-negative definite. The typically large size of J may necessitate careful memory management in evaluating the product $JJ^T$. Hence any positive $\in$ will ensure that $M_k$ is positive definite, as required by the descent condition. The performance of the algorithm thus depends on the choice of $\in$.

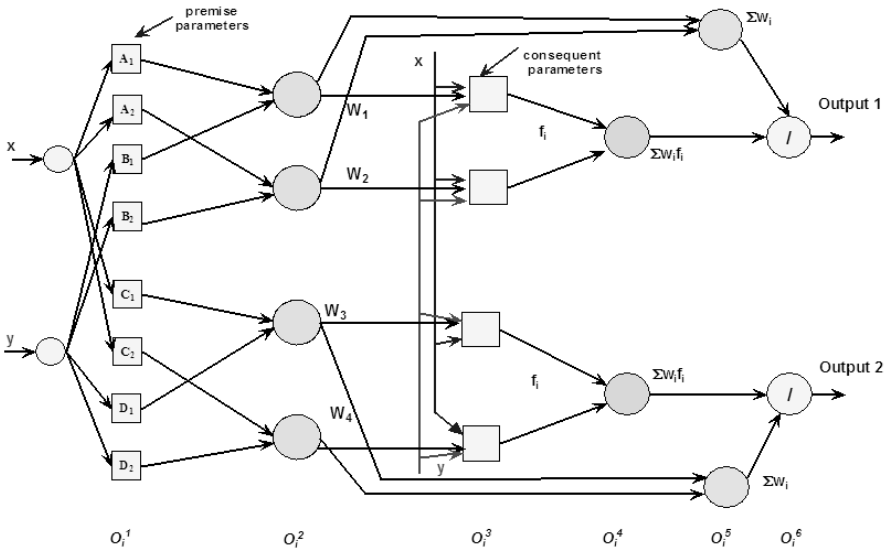
When the scalar $\in$ is zero, this is just Newton's method, using the approximate Hessian matrix. When $\in$ is large, this becomes gradient descent with a small step size. As Newton's method is more accurate, $\in$ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. By doing this, the performance function will always be reduced at each iteration of the algorithm.

### 1.3.4 Neuro-Fuzzy System

Neuro Fuzzy (NF) computing is a popular framework for solving complex problems [3]. If we have knowledge expressed in linguistic rules, we can build a Fuzzy Inference System (FIS) [8], and if we have data, or can learn from a simulation (training) then we can use ANNs. For building a FIS, we have to specify the fuzzy sets, fuzzy operators and the knowledge base. Similarly for constructing an ANN for an application the user needs to specify the

architecture and learning algorithm. An analysis reveals that the drawbacks pertaining to these approaches seem complementary and therefore it is natural to consider building an integrated system combining the concepts. While the learning capability is an advantage from the viewpoint of FIS, the formation of linguistic rule base will be advantage from the viewpoint of ANN. Figure 3 depicts the 6-layered architecture of multiple output neuro-fuzzy system implementing a Takagi-Sugeno fuzzy inference system. For technical details, the reader is advised to consult [15].

**Fig. 1.3.** Architecture of ANFIS




## 1.4 Ensemble of GP Techniques

Our goal is to optimize four error measures namely Root Mean Squared Error (RMSE), Correlation Coefficient (CC), Maximum Absolute Percentage Error(MAP) and Mean Absolute Percentage Error (MAPE). The task is to have minimal values of RMSE, MAP, MAPE and a maximum value for CC. The objective is to carefully construct the different GP models to achieve the best generalization performance. Test data is then passed through these individual models and the corresponding outputs are recorded. Suppose results obtained by LGP and MEP are $a_n$and $b_n$ respectively and the corresponding desired value is $x_n$. The task is to combine $a_n$ and$b_n$ so as to get the best

output value that maximizes the CC and minimizes the RMSE, MAP and MAPE values.

We consider this problem as a multiobjective optimization problem in which we want to find solution of this form: ($coef_1$, $coef_2$) where $coef_1$, and $coef_2$ are real numbers between -1 and 1, so as the resulting combination:

$$coef_1 * a_n + coef_2 * b_n \quad (1.13)$$

would be close to the desired value $x_n$. This means, in fact, to find a solution so as to simultaneously optimize RMSE, MAP, MAPE and CC. This problem is equivalent to finding the Pareto solutions of a multiobjective optimization problem. For this problem, the objectives are RMSE, MAP, MAPE and CC. We use the well known Multiobjective Evolutionary Algorithm (MOEA) - Nondominated Sorting Genetic Algorithm II (NSGAII) [10] and a short description of this algorithm is given in the subsequent Section.

### 1.4.1 Nondominated Sorting Genetic Algorithm II (NSGA II)

In the Nondominated Sorting Genetic Algorithm (NSGA II)[10] for each solution $x$ the number of solutions that dominate solution $x$ is calculated. The set of solutions dominated by $x$ is also calculated. The first front (the current front) of the solutions that are nondominated is obtained.

Let us denote by $S_i$ the set of solutions that are dominated by the solution $x^i$. For each solution $x^i$ from the current front consider each solution $x^q$ from the set $S_i$.

The number of solutions that dominates $x^q$is reduced by one. The solutions which remain non-dominated after this reduction will form a separate list. This process continues using the newly identified front as the current front. Let $P(0)$ be the initial population of size $N$. An offspring population $Q(t)$ of size $N$ is created from current population $P(t)$. Consider the combined population R(t) = P(t) ∪Q(t).

Population $R(t)$ is ranked according to nondomination. The fronts $F_1$, $F_2$, ... are obtained. New population $P(t+1)$ is formed by considering individuals from the fronts $F_1$, $F_2$, ..., until the population size exceeds N. Solutions of the last allowed front are ranked according to a crowded comparison relation.

NSGA II uses a parameter (called *crowding distance*) for density estimation for each individual. Crowding distance of a solution $x$ is the average side-length of the cube enclosing the point without including any other point in the population. Solutions of the last accepted front are ranked according to the crowded comparison distance

NSGA II works follows. Initially a random population, which is sorted based on the nondomination, is created. Each solution is assigned a fitness

equal to its nondomination level (1 is the best level). Binary tournament selection, recombination and mutation are used to create an offspring population. A combined population is formed from the parent and offspring population. The population is sorted according to the nondomination relation. The new parent population is formed by adding the solutions from the first front and the followings until exceed the population size. Crowding comparison procedure is used during the population reduction phase and in the tournament selection for deciding the winner.

## 1.5 Experiment Results

We considered 7 year's month's stock data for Nasdaq-100 Index and 4 year's for NIFTY index. Our target is to develop efficient forecast models that could predict the index value of the following trading day based on the opening, closing and maximum values of the same on a given day. For the Nasdaq-100index the data sets were represented by the 'opening value', 'low value' and 'high value'. NIFTY index data sets were represented by 'opening value', 'low value', 'high value' and 'closing value'. The assessment of the prediction performance of the different connectionist paradigms and the ensemble method were done by quantifying the prediction obtained on an independent data set.

### 1.5.1 Parameter Settings

#### MEP Parameter Settings

Parameters used by MEP in these experiments are presented in Table 1.1. Next two experiments analyze the results obtained by MEP by considering different population sizes and different values for chromosome length. The values for the other parameters are adapted from Table 1.1. Population size was considered 150 for both test data. Average results obtained from 10 different runs and the best results obtained are presented in Table 1.2. Results obtained for different population sizes and various chromosome lengths are presented in Tables 1.3 and 1.4 respectively for both Nasdaq and Nifty test data.

As evident from Table 1.2 the best results for Nasdaq is obtained using a population of 100 individuals and for Nifty using a population of 150 individuals. Table 1.3 illustrates that the best result for both Nasdaq and Nifty is obtained using a chromosome length of 40 and 50.

#### LGP Parameter Settings

Parameters values used by LGP for Nasdaq and Nifty test data are presented in Table 1.4.

**Table 1.1.** Values of parameters used by MEP

| **Parameter** | | **Value** |
|---|---|---|
| Population size | Nasdaq | 100 |
| | Nifty | 50 |
| Number of iteration | Nasdaq | 60 |
| | Nifty | 100 |
| Chromosome length | Nasdaq | 30 |
| | Nifty | 40 |
| Crossover Probability | | 0.9 |
| Functions set | | +, - , *, /, sin, cos, sqrt, ln, lg, $\log_2$, min, max, abs |

**Table 1.2.** Performance measures obtained by MEP or population sizes

| | | | **Population size** | | | |
|---|---|---|---|---|---|---|
| | | | **50** | **100** | **150** | **200** |
| **Nasdaq** | **RMSE** | Best | 0.022 | 0.032 | 0.035 | 0.0168 |
| | | Average | 0.024 | 0.027 | 0.03 | 0.022 |
| | **CC** | Best | 0.999 | 0.097 | 0.97 | 0.992 |
| | | Average | 0.995 | 0.984 | 0.98 | 0.997 |
| | **MAP** | Best | 97.43 | 103.31 | 103.31 | 103.31 |
| | | Average | 109.59 | 100.37 | 109.33 | 100.7 |
| | **MAPE** | Best | 18.13 | 9.02 | 9.08 | 8.69 |
| | | Average | 23.32 | 13.58 | 18.8 | 18.23 |
| **Nifty** | **RMSE** | Best | 0.0187 | 0.163 | 0.015 | 0.0138 |
| | | Average | 0.02 | 0.0196 | 0.0197 | 0.019 |
| | **CC** | Best | 0.999 | 0.997 | 0.999 | 0.999 |
| | | Average | 0.991 | 0.979 | 0.978 | 0.988 |
| | **MAP** | Best | 38.99 | 31.7 | 27.64 | 30.03 |
| | | Average | 53.02 | 42.225 | 41.85 | 48.81 |
| | **MAPE** | Best | 4.131 | 3.72 | 3.17 | 3.027 |
| | | Average | 4.9 | 4.66 | 4.81 | 4.34 |

### Ensemble Using NSGAII Parameter Setting

Parameters values used by NSGAII for combining (ensembling) MEP and LGP are given in Table 1.5. These parameters were used for both Nasdaq and Nifty test data.

### ANN and NF parameter settings

We used a feed forward neural network with 4 input nodes and a single hidden layer consisting of 26 neurons. Tanh-sigmoidal activation function was for the hidden neurons. The training using LM algorithm was terminated after 50 epochs and it took about 4 seconds to train each dataset. For the neuro-fuzzy system, we used 3 triangular membership functions for each of the input

**Table 1.3.** Performance measures obtained by MEP for different chromosome lengths

| | | | **Chromosome length** | | | |
|---|---|---|---|---|---|---|
| | | | **20** | **30** | **40** | **50** |
| **Nasdaq** | **RMSE** | Best | 0.021 | 0.032 | 0.028 | 0.0165 |
| | | Average | 0.021 | 0.027 | 0.024 | 0.022 |
| | **CC** | Best | 0.998 | 0.976 | 0.977 | 0.993 |
| | | Average | 0.998 | 0.987 | 0.985 | 0.994 |
| | **MAP** | Best | 97.43 | 103.31 | 103.31 | 103.31 |
| | | Average | 97.43 | 100.38 | 118.5 | 115.55 |
| | **MAPE** | Best | 18.11 | 9.02 | 8.91 | 8.53 |
| | | Average | 18.12 | 13.52 | 18.74 | 15.86 |
| **Nifty** | **RMSE** | Best | 0.0187 | 0.0169 | 0.015 | 0.014 |
| | | Average | 0.0193 | 0.023 | 0.0197 | 0.02 |
| | **CC** | Best | 0.999 | 0.990 | 0.999 | 0.992 |
| | | Average | 0.994 | 0.977 | 0.98 | 0.981 |
| | **MAP** | Best | 38.99 | 42.98 | 27.64 | 34.87 |
| | | Average | 43.37 | 52.1 | 38.78 | 40.67 |
| | **MAPE** | Best | 4.125 | 4.08 | 3.17 | 3.30 |
| | | Average | 4.33 | 5.68 | 4.81 | 4.75 |

**Table 1.4.** LGP parameter settings

| **Parameter** | **Value** |
|---|---|
| Population size | 100 |
| Mutation frequency | 95% |
| Crossover frequency | 50% |
| Number of demes | 10 |
| Number of constants | 60 |

**Table 1.5.** Parameters values used by NSGAII for ensembling MEP and LGP

| **Parameter** | **Value** |
|---|---|
| Population size | 250 |
| Mutation probability | 0.3 |
| Crossover probability | 0.5 |
| Number of generations | 1,000 |
| Chromosome length | 30 |

variable and the 27 *if-then* fuzzy rules were learned for the Nasdaq-100 index and 81 *if-then* fuzzy rules for the NIFTY index. Training was terminated after 12 epochs and it took about 3 seconds to train each dataset [1].

### 1.5.2 Comparisons of Results Obtained by Intelligent Paradigms

Table 1.6 summarizes the results achieved for the two stock indices using the five intelligent paradigms (ANN, NF, MEP, LGP and the ensemble between LGP and MEP).

**Table 1.6.** Results obtained by intelligent paradigms for Nasdaq and Nifty test data

| | **ANN** | **NF** | **MEP** | **LGP** | **Ensemble** |
|---|---|---|---|---|---|
| **Test results for Nasdaq** | | | | | |
| **RMSE** | 0.0284 | 0.0183 | 0.021 | 0.021 | 0.0203 |
| **CC** | 0.9955 | 0.9976 | 0.999 | 0.9940 | 1.000 |
| **MAP** | 481.71 | 520.84 | 97.39 | 97.94 | 96.92 |
| **MAPE** | 9.032 | 7.615 | 14.33 | 19.65 | 19.25 |
| **Test results for Nifty** | | | | | |
| **RMSE** | 0.0122 | 0.0127 | 0.0163 | 0.0124 | 0.0127 |
| **CC** | 0.9968 | 0.9967 | 0.997 | 0.995 | 1.000 |
| **MAP** | 73.94 | 40.37 | 31.7 | 40.02 | 31.9 |
| **MAPE** | 3.353 | 3.320 | 3.72 | 2.83 | 2.80 |

The combination obtained by ensemble between LGP and MEP are reported below: Nasdaq: $0.669668 * a_n + 0.334354 * b_n$
Nifty: $0.632351 * a_n + 0.365970 * b_n$

where $a_n$ and $b_n$ correspond to LGP and MEP indices respectively.

As depicted in Table 1.6, for Nasdaq test data, MEP gives the best results for MAP (97.39). Also LGP gives results very close to MEP while the other techniques did not perform that well. While MEP obtained the best result for CC, the performance obtained were close to the results obtained for RMSE by the other paradigms. Results obtained by ensemble clearly outperforms almost all considered techniques. For both Nasdaq and Nifty test data, the values obtained by the ensemble for CC is 1.000.

Since the multiobjective evolutionary algorithm was used to simultaneously optimize all the four performance measures (RMSE, CC, MAP and MAPE), more than one solution could be obtained in the final population (which, in fact, means more than one combination between LGP and MEP). Some examples of the solutions obtained for Nifty is given below:

- Best value for RMSE obtained by the ensemble is 0.012375, while CC = 0.999, MAP = 36.86 and MAPE = 2.71.
- Best value for MAP obtained by the ensemble is 25.04, while RMSE = 0.0138, CC = 0.998 and MAPE = 3.05.
- Best result obtained by the ensemble for MAPE is 2.659513, while RMSE = 0.0124, CC = 0.998 and MAP = 41.44.

## 1.6 Summary

In this chapter, we presented five techniques for modeling stock indices. The performance of GP techniques (empirical results) when compared to ANN and NF clearly indicate that GP could play a prominent role for stock market modeling problems. The fluctuations in the share market are chaotic in the sense that they heavily depend on the values of their immediate fore running fluctuations. Our main task was to find the best values for the several performance measures namely RMSE, CC, MAP and MAPE. We applied a multiobjective optimization algorithm in order to ensemble the GP techniques. Experiment results reveal that the ensemble performs better than the GP techniques considered separately.

According to the No Free Lunch Theorem (NFL), for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class [26]. Taking into account of the NFL theorem, it would be a rather difficult task to predict which paradigm would perform the best for different stock indices [13].

## 1.7 Acknowledgements

This research was supported by the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the MIC (Ministry of Information and Communication), South Korea.

## References

1. Abraham, A. and AuYeung, A., Integrating Ensemble of Intelligent Systems for Modeling Stock Indices, In Proceedings of 7th International Work Conference on Artificial and Natural Neural Networks, Lecture Notes in Computer Science-Volume 2687, Jose Mira and Jose R. Alverez (Eds.), Springer Verlag, Germany, pp. 774-781, 2003.
2. Abraham, A. Philip, N.S. and Saratchandran, P., Modeling Chaotic Behavior of Stock Indices Using Intelligent Paradigms. International Journal of Neural, Parallel & Scientific Computations, USA, Volume 11, Issue (1&2) pp.143-160, 2003.
3. Abraham, A., Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence, Springer-Verlag Germany, Jose Mira and Alberto Prieto (Eds.), Granada, Spain, pp. 269-276, 2001.
4. Aho, A., Sethi R., Ullman J., Compilers: Principles, Techniques, and Tools, Addison Wesley, 1986.
5. Bishop, C. M., Neural Networks for Pattern Recognition, Oxford: Clarendon Press, 1995.

6. M. Brameier and W. Banzhaf, A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining, IEEE Transactions on Evolutionary Computation, 5, pp.17-26, 2001.
7. M. Brameier and W. Banzhaf, Explicit Control of Diversity and Effective Variation Distance in Linear Genetic Programming,in Proceedings of the Fourth European Conference on Genetic Programming, edited by E. Lutton, J. Foster, J. Miller, C. Ryan, and A. Tettamanzi (Springer-Verlag, Berlin, 2002).
8. Cherkassky V., Fuzzy Inference Systems: A Critical Review, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Kayak O, Zadeh L A et al (Eds.), Springer, pp.177-197, 1998.
9. Collobert R. and Bengio S., SVMTorch: Support Vector Machines for Large-Scale Regression Problems, Journal of Machine Learning Research, Volume 1, pages 143-160, 2001.
10. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T., A fast elitist non-dominated sorting genetic algorithms for multiobjective optimization: NSGA II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.
11. C. Ferreira, Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, Complex Systems, 13 (2001) 87129.
12. E.H. Francis et al. Modified Support Vector Machines in Financial Time Series Forecasting, *Neurocomputing* 48(1-4): pp. 847-861, 2002.
13. C. Grosan and A. Abraham, Solving No Free Lunch Issues from a Practical Perspective, In Proceedings of Ninth International Conference on Cognitive and Neural Systems, ICCNS'05, Boston University Press, USA, 2005 .
14. S. Hashem. Optimal Linear Combination of Neural Networks. *Neural Network*, Volume 10, No. 3. pp. 792-994, 1995.
15. J.S.R. Jang, C.T. Sun and E. Mizutani. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Prentice Hall Inc, USA, 1997.
16. T. Joachims . Making large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (Eds.), MIT-Press, 1999.
17. J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection (MIT Press, Cambridge, MA, 1992).
18. J. Miller and P. Thomson. Cartesian Genetic Programming, in Proceedings of the Third European Conference on Genetic Programming, edited by Riccardo Poli, Wolfgang Banzhaf, Bill Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty (Springer-Verlag, Berlin, 2002).
19. Nasdaq Stock Market$^{SM}$: http://www.nasdaq.com.
20. National Stock Exchange of India Limited: http://www.nse-india.com.
21. M. Oltean and C. Grosan. A Comparison of Several Linear GP Techniques. *Complex Systems*, Vol. 14, Nr. 4, pp. 285-313, 2004
22. M. Oltean M. C. Grosan. Evolving Evolutionary Algorithms using Multi Expression Programming. *Proceedings of The 7$^{th}$ European Conference on Artificial Life*, Dortmund, Germany, pp. 651-658, 2003.
23. N.S. Philip and K.B. Joseph. Boosting the Differences: A Fast Bayesian classifier neural network, *Intelligent Data Analysis*, Vol. 4, pp. 463-473, IOS Press, 2000.
24. C. Ryan C. J.J. Collins and M. O'Neill. Gramatical Evolution: Evolving programs for an arbitrary language. In *Proceedings of the first European Workshop on Genetic Programming,* Springer-Verlag, Berlin, 1998.

25. R.E. Steuer. Multiple Criteria Optimization: Theory, Computation and Applications. New York: Wiley, 1986
26. D.H. Wolpert and W.G. Macready. No free lunch theorem for search. Technical Report SFI-TR-95-02-010. Santa Fe Institute, USA, 1995.