A Study on Financial Time Series Forecasting and Symbolic Regression by Means of a Hybrid Probabilistic Model-Building Cartesian Genetic Programming Methodology

by

Mahsa Mostowfi

B.Sc., Shiraz Azad University, 1999

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2013

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

A Study on Financial Time Series Forecasting and Symbolic Regression by Means of a Hybrid Probabilistic Model-Building Cartesian Genetic Programming Methodology

Mahsa Mostowfi

M. Sc. in Computer Science, 2013

Ryerson University

Abstract

This work proposes a hybrid algorithm called *Probabilistic Incremental Cartesian Genetic Programming* (PI-CGP), which integrates an *Estimation of Distribution Algorithm* (EDA) with *Cartesian Genetic Programming* (CGP). PI-CGP uses a fixed-length problem representation and the algorithm constructs a probabilistic model of promising solutions. PI-CGP was evaluated on symbolic regression problems and next trading day stock price forecasting.

On the symbolic regression problems PI-CGP did not outperform other approaches. The reason could be premature convergence and being trapped at a local minimum. However, PI-CGP was competitive at stock market forecasting. It was comparable to a fusion model employing a *Hidden Markov Model* (HMM). HMMs are extensively used for time-series forecasting. This result is promising considering the volatile nature of the stock market and that PI-CGP was not customized toward forecasting.

Acknowledgements

I would like to express my sincere gratitude to Dr. Marcus dos Santos, my supervisor, for the continuous support, for his patience, guidance, and motivation which made this thesis possible.

I would also like to thank my thesis committee, Dr. Sadeghian, Dr. Abhari, and Dr. Woungang, for reading my thesis and their insightful comments.

I would like to thank the following friends who supported me: Marmar Matin, Maryam Aslanzad, Amir Farrokh Azmayesh, Leyla Vakilian, Ervis Sofroni, Sweeney Luis, Mohammad Islam, Rong Liu, and Vimalan Yogaratnam.

Dedication

To my parents, Graham, Mahyar, and Sahar for their love, endless support, and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work presents a hybrid methodology which integrates an *Estimation of Distribution Algorithm* (EDA) with *Cartesian Genetic Programming* (CGP). The method is applied to the synthesis of mathematical equations to fit a given data set and to the short-term prediction of a stock's daily price.

This work belongs to the field of *Evolutionary Computation* (EC), a subfield of *Machine Learning* (ML). EC is the name given to a collection of optimization methodologies inspired by the biological evolution of living organisms (*i.e.* reproduction and selection). In EC, a population consists of a set of individuals. Each individual is a candidate solution to a given problem. Individuals are selected based on their fitness, where *fitness* is a metric defining how well a candidate solution solves the given problem, and the evolutionary operators are applied to them to generate the population of the next generation. This process iterates until a satisfactory solution to the given problem is found or a termination criterion is met.

EDAs are a set of optimization methods within the EC field which guide the search towards the optimum solutions by building a probabilistic model of promising solutions and sampling from the probabilistic model. EDAs eliminate the need for search operators in generating the new population of individuals. EDAs differentiate themselves from other stochastic optimization techniques by using probabilistic models to capture the features which makes a candidate solution better than other solutions in the search space.

Genetic Programming (GP) is the field of EC in which individuals are represented by complex structures (*e.g.* trees, graphs, and the like). The internal nodes are selected from a function set and the leaves are chosen from a terminal set. The terminal set consists of constants or variables. The

genetic operators are used to select the fittest individual and then copy it to the next generation or create new offspring from the selected individuals.

Cartesian Genetic Programming (CGP) is a form of GP in which the program structure is a fixed-length array of integers. CGP represents a program as a directed graph. Nodes of the graph are arranged in a number of rows and columns. Each node takes its inputs from the output of the previous nodes or the program inputs.

Prediction of stock price and price direction is a non-trivial problem and is important for investors. There is a large amount of historical data being stored and it is expected to continue to grow. These data contain valuable knowledge which can be extracted and used for prediction purposes. Analyzing such data manually is not practical, so researchers have developed approaches (*e.g.* machine learning and statistical data analysis) to automatically discover patterns in the data. These approaches are the subject of the field known as data mining and knowledge discovery. It is during the *Data Mining* (DM) phase of *Knowledge Discovery* (KD) that knowledge is extracted from the data and large databases are searched to find interesting patterns in the data.

Symbolic regression is a technique for finding a mathematical equation that fits a given set of data points. In this work, symbolic regression is used for knowledge discovery to find the patterns in the stock price data and present them in the form of mathematical expressions. Symbolic regression searches for both the parameters and the form of the equations simultaneously.

1.1 Motivation

Evolutionary Computation has been used as an effective tool in knowledge discovery and data mining because it is adaptive, robust, and flexible [1]. The main motivation for using EC techniques in financial time series data mining is their capability to efficiently explore the search space and discover relationships between various market factors while handling complex fitness functions [2]. Previous works have shown that data mining is able to analyze financial time series data and

uncover hidden patterns that predict future behaviors in financial markets [2–4].

CGP is a good candidate for solving symbolic regression problems because it uses a directed graph and it is immune to *bloat*. CGP represents a program as a directed graph which provides the advantage of reusing the subgraphs [5] compared to GP tree representations where identical sub-trees have to be built separately. When GP is applied on symbolic regression problems, a phenomenon known as bloat is often observed. Bloat is when the program size increases without any changes in its fitness value. CGP does not suffer from bloat because it employs a fixed-length structure. Bloat results in increasing the fitness evaluation time and reduces the efficiency of the search operators.

EDAs can be applied to simple and complex problems using a diverse range of probabilistic models. Previous works have shown that a hybrid methodology which combines a fixed-length program representation to an EDA outperforms the EDA only approach in symbolic regression problems [6]. It is also a promising methodology for Neuroevolution [7].

1.2 Objectives

Our objective is to build a hybrid algorithm that develops a probabilistic model of CGP programs. The model will be learned using a method inspired by the Probabilistic Incremental Program Evolution (PIPE) approach [8]. The system will be integrated into SQL Server Analysis Services (SSAS) as a plug-in algorithm to perform data mining.

The algorithm's performance will be evaluated using a set of experiments in the domain of symbolic regression and stock market forecasting.

1.3 Methodology

The following briefly describes how we developed the hybrid algorithm and integrated it into SQL Server Analysis Services (SSAS) as a data mining algorithm.

To begin, we require a structural representation. The presented system uses the representation developed in CGP. In CGP, the encoding of an individual, or *genotype*, is a fixed-length list of integers and encodes an indexed graph (the *phenotype*). Each integer encodes an input or the function of a node in the graph.The graph can be used for a number of program types including algebraic formula, logic formula, neural networks, etc.

Next, we define the probabilistic model which estimates the distribution of promising solutions. Given that in CGP the genotype has a fixed length, the probabilistic model is also a fixed-length array with the same length as the CGP genotype. Each entry of the array stores a list of probability values for each possible input or function. The probabilistic model is updated to increase the probability of sampling the best individual of the current generation. This is done by increasing the probability values in the model which correspond to the nodes which generated the best individual of the current generation.

Next, we integrate the hybrid algorithm into SSAS. SSAS provides data mining functionality in Microsoft SQL Server (*i.e.* it provides the tools to design data mining solutions, based on which the data source binding, design and deployment of data mining models for predictive purposes is easily done).

In order to integrate the hybrid algorithm into SSAS, a plug-in algorithm must implement a set of Component Object Model (COM) interfaces. COM is a software architecture which allows the re-usability of objects (software components) regardless of the programming language used to implement the objects. A COM object is accessible through a set of interfaces (collection of functions). Through these interfaces the plug-in algorithm exposes the algorithm information, detects and stores the pattern in the data, then exposes the discovered patterns. A plug-in algorithm is a class library which implements the following classes: *Metadata class, Algorithm class, and Navigator class*. They are partially implemented in a plug-in wrapper provided in the API for SQL Server package by Microsoft, but the plug-in developer must implement the mentioned classes using the base classes and interfaces provided.

To test the performance of our system, we ran experiments against a selection of symbolic regression and stock market forecasting benchmarks. For the symbolic regression class of experiment, we compared the performance of our hybrid algorithm with the following methods: GP, CGP with variable crossover, and PIPE. For the stock market forecasting experiments, we compared the performance of our data mining algorithm with GP and a fusion model which combines *Hidden Markov Model* (HMM), *Artificial Neural Networks* (ANNs) and *Genetic Algorithm* (GA).

1.4 Results and Contributions

Our results show that the hybrid algorithm on the set of symbolic regression problems outperformed the GP approach when the target expression was a polynomial of degree 6, but fails to find the target expression of degree 11. We suspect that our algorithm was trapped at a local minimum. Increasing parameters such as number of generations and learning rate resulted in improving the average fitness value. The average convergence rate of our approach against CGP with variable crossover was slower. One possible reason could be the premature convergence of our approach. Increasing the population size resulted in improving the convergence but did not succeed at outperforming CGP with variable crossover. In the non-trivial function regression problem, the target expression was not found using PI-CGP. A possible reason could be that the run required more program evaluations.

The accuracy of our hybrid algorithm on the next day stock price forecasting compared to a GP system with the window period of 300 on the training and cross-validation range was significant. The window period specifies the number of data points collected from the past history of the stock price time series. On the cross-validation period, GP with window period of 450 performed slightly better than our approach but not statistically significant. On the training period, our approach performed better than the GP with window period of 450 but not statistically significant. We found that training our algorithm on wider range of historical price data would result in better Mean

Absolute Error (MAE) on the cross-validation period. Comparing the accuracy of our algorithm to a fusion model of ANN, GA, and HMM we observed that our approach was better for IBM and Apple, but not statistically significant. For Dell, the fusion model performed slightly better but not statistically significant.

Our contribution is a novel EDA-CGP approach for mining financial time series data and the demonstration that our approach performs reasonably well considering the volatile nature of stock market. Considering that our approach was not customized toward forecasting, it is comparable to the fusion model which incorporated tools such as HMM, a tool which is extensively used for time series forecasting because of its proven suitability for modeling dynamic systems.

1.5 Structure of this thesis

In chapter 2, we provide an overview of the concepts and related work. We begin by a brief introduction to the field of *Evolutionary Computation* (EC). Then, we review the following EC algorithms: Genetic Programming and Cartesian Genetic Programming. Next, we review the Estimation of Distribution Algorithms. Finally, we review data mining and the application of Evolutionary Algorithms to mining financial data.

Chapter 3 details our approach including: PI-CGP program representation, the probabilistic model, the learning process which guides the search towards optimal solution, database integration, and performance evaluation. Next, we describe how our PI-CGP is integrated algorithm with SQL Server Analysis Services (SSAS) in order to perform data mining. To end this chapter, we describe the benchmark experiments chosen to evaluate the performance of our system.

Chapter 4 presents the results of the experiments done on symbolic regression and stock market forecasting, specifically the next trading day price prediction.

Chapter 5 discusses our results, contributions, and possible extension of this work.

Chapter 2

Literature Review

This chapter presents the background concepts in the areas of Evolutionary Computation and Data Mining and reviews the research related to this work.

2.1 Evolutionary Computation

Evolutionary Computation is the field of Computer Science that uses techniques inspired by the principles of the natural evolution such as reproduction and selection. The techniques are used to perform optimization given a metric to determine a well-performing solution. Evolution in a population of program solutions is driven by the following four factors [9]:

1. **Reproduction of individuals:** Mutation and crossover produce the new candidate solutions from one or more parents.
2. **Variation:** Mutation and crossover are the variation operators which create the new individuals from the old ones. They increase the diversity of individuals.
3. **Heredity:** Behavioral traits are inherited from generation to generation.
4. **Finite Resources:** Limited number of resources leads to competition of individuals in a population for survival. The selection operators pick the individuals with better fitness.

The execution of an EC algorithm consists of the following steps:

1. A random population of candidate solutions to the problem is generated.
2. The selection operator is applied to the population and the candidates are selected based on their fitness, which determines how qualified the candidates are to solve the problem.

3. The search operators are applied to the selected candidates and generate the next generation individuals.
4. While the termination criterion has not been met the steps 2-4 are repeated.

Generally the termination criteria are defined as either the process runs for a number of generations less than the maximum number of generations or when an acceptable solution to the problem is found.

Some of the best-known algorithms in the class of EC methodologies are: Genetic Algorithms (GA), Genetic Programming (GP), and Cartesian Genetic Programming (CGP). They differ in the way individuals are represented and their genetic operator implementations. In GA, the structure of an individual is a fixed-length string representing the parameters to be tuned. In GP, the individuals are represented as a program tree. In CGP, the individuals are encoded as fixed-length strings that are decoded into directed graphs.

2.1.1 Genetic Programming

In Genetic Programming, the problem solutions known as individuals are initially generated randomly and are evolved by applying genetic operators in each generation to generate better programs.

The individuals in GP are in the format of LISP parse trees. The internal nodes are selected from a problem-specific function set and leaves are selected from a problem-specific terminal set. The number of arguments a function takes is known as the *arity* (*e.g.* $+, -$ take two arguments and have arity 2.) There is no distinction between the individual's *genotype* and *phenotype* (*i.e.* the LISP tree is both the program being evaluated and the genome being mutated.) Genotype is the genetic structure of an individual and phenotype is the individual's observable characteristics.

The operations used in canonical GP (as defined by Koza [10]) to evolve the program solutions are: selection, reproduction, crossover, and mutation.

The selection operator chooses individuals from the current generation based on fitness. A common selection operator is roulette wheel selection, where the individuals are selected randomly in proportion to their fitness. In reproduction the selected individual is copied into the next generation without any changes. Crossover starts with two individuals and produces two offspring. The offspring are created by selecting a random node in each tree and swapping them. During mutation a random part of the selected individual is chosen and altered based on the mutation technique. The mutation implementation should ensure that the operation results in a valid program tree (*e.g.* Functions with different arity require trimming or adding a random terminal or tree. If a function becomes a terminal the branches are discarded).

The process of program generation, evaluation, selection, and application of the genetic operators is iterated until a solution is found or the maximum number of generations is reached.

2.1.2 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a form of GP developed by Miller and Thomson from their work on evolutionary design of digital circuits [11].

In CGP each individual is represented as a directed graph and each node in the graph is mapped to a string of integers referred to as genes in the genotype. Unlike in GP, in CGP there is a distinction between the genotype and phenotype. In CGP, the genotype encodes the phenotype (directed-graph) as a fixed-length array of integers. The genotype represents the program inputs, outputs, and the graph's structure whereas the phenotype is a graph (grid of nodes).

There exists a many-to-one genotype-phenotype mapping in CGP which means that a number of unique genotypes may produce functionally equivalent phenotypes. This is because of different types of redundancy including: *node redundancy, functional redundancy,* and *input redundancy*. Node redundancy is the result of unconnected nodes in the graph whose output is not connected to a node leading to the program output. Functional redundancy occurs when a function can be

implemented with less nodes. Input redundancy occurs when one or more of the program inputs are not connected to any nodes.

CGP only uses mutation to create offspring for the next generation. The mutation operator is a point mutation in which a gene is randomly chosen and its value is changed to a valid random value (*e.g.* If a function gene is chosen then its value is changed to any function in the function set, whereas if an input gene is chosen its value is changed to the output of any previous node in the phenotype or any program input).

CGP has a number of advantages over other forms of GP. CGP does not suffer from bloat since CGP genotype is a fixed-length array. Bloat is the rapid growth in size of the individuals causing the fitness evaluation to take a longer time.

Since CGP represents a program as a directed graph, the sub-graphs can be re-used, whereas in GP the identical sub-trees have to be constructed separately.

2.1.2.1 Cartesian Genetic Programming with Crossover

In CGP with Crossover (CGPwC) method the CGP genotype has been replaced with a fixed-length array of real-value numbers in the range [0,1]. The decoding process from real value to integer value is achieved by two equations. Equation 2.1 applies when $gene_i$ encodes the function of a node and Equation 2.2 when $gene_i$ encodes the input of a node.

$$floor(gene_i * func_{total}) \tag{2.1}$$

$$floor(gene_i * node_j) \tag{2.2}$$

where $func_{total}$ is the number of functions, $node_j$ is the node number, and $0 \leq \text{j} \leq node_{total}$, $node_{total}$ being the number of nodes and terminals.

Crossover is performed using Equation 2.3 to produce two offspring, o_1 *and* o_2 .

$$o_i = (1 - r_i) * p_1 + r_i * p_2 \tag{2.3}$$

where p_1 and p_2 are the parents, and r_i is a uniformly generated random number in the range (0,1) chosen for each offspring o_i, i being in the range [0,2).

The mutation operator in this method is based on the mutation operator in CGP, but it changes the value of a gene to a uniformly generated random real value in the range [0,1].

2.1.3 Estimation of Distribution Algorithms

Optimization algorithms such as Evolutionary Algorithms use reproduction and selection operations on promising individuals to generate better solutions. Depending on the problem domain these operators may not be effective enough to preserve the important building blocks of the candidate solutions. To address this limitation, a new class of algorithms was introduced called Probabilistic Model Building Genetic Algorithms (PMBGA), also known as Estimation of Distribution Algorithms (EDA).

Estimation of Distribution Algorithms is a class of methodologies that explicitly induces a probabilistic model of the best candidate solutions. New candidates are generated by sampling the distribution model.

The simplest form of the EDA is called Population Based Incremental Learning (PBIL) [12] which is based on a fixed length binary encoded representation of the individuals. In this algorithm, first the probability vector is initialized. Each position in the probability vector specifies the probability of each bit position in the solution string containing a '1'. Then individuals are sampled according to the probability vector and are evaluated. The values in the probability vector are updated towards representing the highest evaluation solutions. The mentioned cycle is repeated until the termination criterion is met.

One of the early works in the field of PMGBA is the Univariate Marginal Distribution Algorithm (UMDA) [13]. UMDA uses the Univariate probability model in which there is no dependency between the variables of an individual.

Another form of EDA is Compact Genetic Algorithm (cGA) [14] which maintains the probability vector as in PBIL but unlike PBIL it samples two individuals. The individual with better fitness is selected and the probability vector is updated. This process continues until the probability vector converges.

Probabilistic Developmental Program Evolution (PDPE) [6] is a probabilistic model building method in which the probability distribution of promising solutions are stored in a fixed length Probabilistic Prototype Chromosome (PPC) and individuals are presented as expression trees.

Probabilistic Developmental Neuroevolution (PDNE), which is an EDA-GP technique, has been used to evolve the topology and weights of an Artificial Neural Network (ANN) [7].

Probabilistic Incremental Program Evolution (PIPE) [8] is an EDA in which the structure of the probability distribution model is a program tree. PIPE uses a model of a complete N-ary tree known as the Probabilistic Prototype Tree (PPT).

PIPE uses two forms of learning: Genetic-Based Learning (GBL) and Elitist Learning. GBL is the main learning algorithm. In this form of learning the population is generated by sampling the Probabilistic Prototype Tree (PPT), then the individuals are evaluated and the PPT is updated to increase the probability of creating the best program from the current generation. During Elitist Learning the PPT is updated to increase the probability of generating the elitist program. During the Elitist Learning there is no population generated nor is the PPT is mutated.

At the end of each generation the PPT is pruned. During pruning subtrees that are not required as function operators are removed and subtrees of nodes with at least one probability value in the probability vector greater than a threshold are also removed.

In this thesis a hybrid algorithm called Probabilistic Incremental Cartesian Genetic Programming (PI-CGP) is proposed. PI-CGP is a combination of PIPE's probabilistic model and learning

algorithm and CGP's program representation is proposed.

2.2 Data Mining (DM)

The amount of data collected every day and stored in databases is increasing rapidly. Analyzing such volumes of data manually would be difficult, if not impossible, so it is important to build tools that automatically uncover valuable information from the data and transform it into knowledge. This is the objective of the field of Knowledge Discovery from Data (KDD).

The knowledge discovery process consists of the following steps:

1. **Data Preprocessing:** The real-world data consists of noisy and incomplete data. The data preprocessing step improves the quality of data and so it improves the accuracy of subsequent steps.

2. **Data mining:** During this step methods are applied to data to extract interesting patterns. Different data mining techniques are used for different purposes. Association, Clustering, Classification, and Regression are the major data mining techniques.

   In regression a numerical value is predicted whereas in classification a categorical or nominal model is built. The regression technique can be used to predict the future values of a financial time series.[1]

   There are several regression algorithms used in data mining such as Genetic Programming (GP), Support Vector Machines (SVM), and Neural Networks (NN).

3. **Pattern evaluation and knowledge presentation:** In this step the discovered patterns are evaluated. The patterns which are accurate and comprehensible are selected and presented to the user. The discovered knowledge is usually presented in terms of rules, classification models, statistical analysis, etc.

[1] A series of data points such as a stock's price measured in regular time intervals.

Knowledge Discovery is an iterative process. When the knowledge is discovered and presented to the user based on the user's feedback the above-mentioned steps may be repeated and some modifications may be applied to them. For example, new data sources may be integrated into the system or evaluation measures may be changed.

2.3 Data Mining with Evolutionary Algorithms

The main reason that Evolutionary Algorithms are applied to Knowledge Discovery and Data Mining is that these algorithms are robust and adaptive and can perform global searches [15]. EAs allow evaluation of multiple criteria simultaneously. These features give the data miner some flexibility in designing the fitness functions.

2.3.1 EAs for Data Mining in Financial Data

Credit fraud prediction [16], credit rating [17], forecasting stock market [18, 19], and currency exchange rates [20] are prominent data mining tasks in finance and banking systems.

Mining financial data is a challenging task as it requires an understanding of how the financial markets work, how to model the financial markets, and how to validate those models [21]. In response to such a difficult task, data mining techniques have been used. Applying data mining to financial time series data can help to discover interesting patterns within the data or to discover the relationship between several time series. Extensive research has been done to solve these problems as it has valuable potential benefits [22].

The existing financial data forecasting mining methods can be categorized into:

- **Classical:** The classical methods are based on the statistical models (e.g. Autoregressive Integrated Moving Average (ARIMA)). The major difficulty encountered when using these methods is that usually many attempts must be made to find the best model.

- **Modern:** The modern methods are based on the algorithms from the Artificial Intelligence field. (e.g. Neural networks, evolutionary computation techniques, e.g. Genetic Programming, Genetic Algorithms)

Using EAs overcomes the major difficulty encountered when using statistical methodologies for financial data forecasting. In [23], a GP-based method was shown to yield a better performance compared to NN in predicting the stock price data. This was justified by that the NN suffered from over-fitting and its convergence time was much longer. Over-fitting occurs when the evolved model performs well on the training data, but does not perform well on unseen data.

Multi-Expression Programming (MEP) [24], Linear Genetic Programming (LGP) [25], and hybrid MEP-LGP [26] performed better than the trained NN with a neuro-fuzzy model in terms of several performance measures in predicting the stock indices. The hybrid MEP-LGP algorithm also performed better than each of the individual GP techniques.

The selected performance measures that were used: Root Mean Squared Error (RMSE), Correlation Coefficient (CC), Maximum Absolute Percentage Error (MAP), and Mean Absolute Percentage Error (MAPE).

Chapter 3

Methodology

This chapter outlines the development of our hybrid algorithm called Probabilistic Incremental Cartesian Genetic Programming (PI-CGP). In this approach we follow CGP's program representation, PIPE's probability distribution model and learning algorithm which directs the exploration in the search space towards the promising solutions.

We first describe the progam representation of PI-CGP and the structure of the probabilistic model used for sampling the program individuals. Then, we explain the technique for updating the probabilistic model and sampling the individuals. Next, we explain how to plug our algorithm into SQL Server Analysis Services (SSAS). In this chapter, the motivation for using SQL Server Analysis Services (SSAS) is also highlighted. Finally, we describe the experiments performed to evaluate our proposed system.

3.1 PI-CGP: The Hybrid Methodology

3.1.1 Program Representation

PI-CGP genotype representation follows the CGP genotype representation. CGP programs are directed graphs that are encoded as a fixed length list of integers (the genome). The integers represent the functions of the nodes and the connections between nodes. Each node is represented as a subset of the integers in the genome. In each node, the last integer (gene) encodes the function (e.g. $+, -, \times, \div$) and other integers encode the input connections to the node. Each node takes its inputs from the program's inputs or the output of the previous nodes. The last integers in the genome encode which node's output becomes the output of the program.

Table 3.1: Program inputs and functions encoding

	Value	Encoding
Program input	0	X
	1	Random constant
Function	0	+
	1	-
	2	$\times$
	3	$\div$

Assuming that all the nodes have function arity of *two*, for the nodes with arity of *one* any extra genes are not considered in the genotype decoding process.

The program inputs are labelled from 0 to *n-1* where *n* is the number of program inputs. The nodes are labelled from *n* to *n + m - 1*, where *m* is the number of nodes. The output nodes are labelled from *n + m* to *n + m + k - 1*, where k is the number of output nodes [27].

An example of CGP genotype and genotype-phenotype mapping is shown in Figure3.1. The figure shows a CGP program with 5 nodes per row and 2 nodes per column. The program has two inputs: *X* and a random constant and one output. Table 3.1 presents the program inputs and functions encoding. Each node has 2 input connections and the *levels back* parameter is 5. The *levels back* parameter determines from how many previous columns a node can get its input from.

Nodes in the same column or upstream columns are not allowed to be connected to each other. The dotted nodes are the unconnected nodes that are not taken into account when calculating the *fitness* of an individual.

3.1.2 The Probabilistic Prototype Array (PPA)

The Probabilistic Prototype Array (PPA), the probability distribution of PI-CGP genes, was inspired by PIPE's Probabilistic Prototype Tree (PPT) [8]. The PPT is a complete *n-ary* tree whereas the PPA is a fixed length array with the same length as the PI-CGP genotype. Each position in the array is matched to a gene in the genotype and stores a list of probability values (see Figure 3.2).

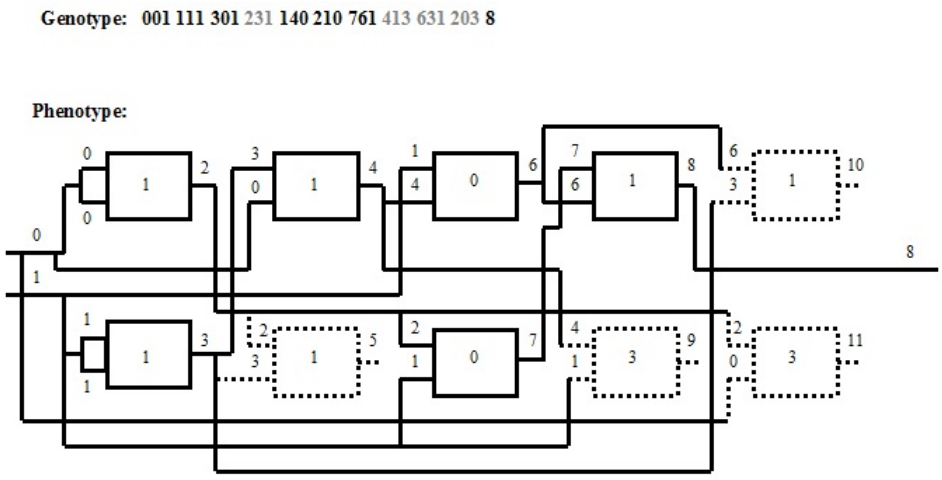

Figure 3.1: A CGP genotype and corresponding phenotype for the function X

The probabilities indicate the likelihood of choosing any of the available functions or inputs. Each probability list is normalized.

For a gene encoding a function, the probability of each value is initialized as:

$$P = \frac{1}{Number\ of\ functions\ in\ function\ set\ F}, \text{ where } F = \{+, -, \times, \div, \sin, \cos, \exp, r\log\} \quad (3.1)$$

If the gene encodes inputs to the node, then the values are initialized as:

$$P = \frac{1}{Number\ of\ previous\ nodes\ +\ Number\ of\ terminals\ in\ terminal\ set\ T}, \text{ where } T = \{x, R\} \quad (3.2)$$

The *R* element, *Generic Random Constant*, is a function with zero argument and provides a random value. Its value remains the same throughout a generation.

3.1.3 The Learning Process

The probabilistic model (PPA) uses the PIPE [8] learning process. The model is said to learn if the probabilities can be moved to make the better solutions more probable.

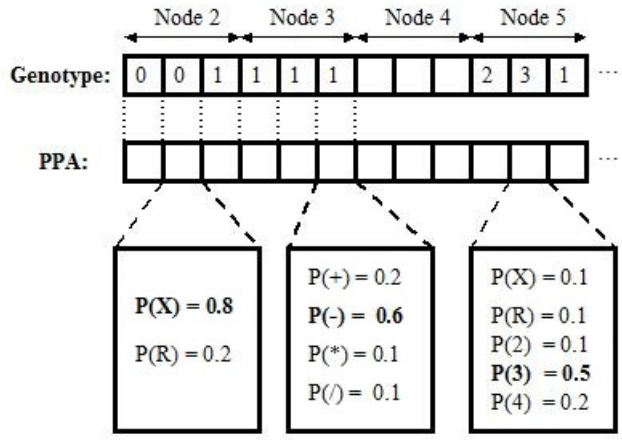

Figure 3.2: Probabilistic Prototype Array (PPA)

The learning process is shown in Algorithm 1.

Algorithm 1: PIPE

```
1:  repeat
2:    if P_el < a random value and it is not the first generation then
3:      Update probabilities based on PROG_el (the elitist program)
4:    else
5:      Creation of the program population
6:      Calculation of the fitness value for each program
7:      Learning from the best program of the population
8:      Mutation of the PPA
9:    end if
10: until Termination Criterion is met
```

Creation of the program population: A population of programs is generated by using the PPA. To create an individual we sample from the PPA. Sampling from the PPA is done through the roulette wheel selection algorithm.

Learning from the best program of the population: The best program of the current generation is chosen ($PROG_b$). The probabilities in the PPA are modified to increase the probability of creating the best program.

The probability of creating $PROG_b$ is calculated as in [8]:

$$P(PROG_b) = \prod_{Nodes\ used\ to\ generate\ PROG_b} P, \text{ where P is the probability of a node.} \tag{3.3}$$

$$P = \prod_{i=0}^{|N|} P_i \tag{3.4}$$

where P_i denotes the probability of the i-th gene in the $PROG_b$ genotype and $|N|$ is the length of the node.

Then a target probability for the best program is calculated. This is the probability that $PROG_b$ will be generated by the PPA [8]:

$$P_{TARGET} = P(PROG_b) + (1 - P(PROG_b)) \cdot lr \cdot \frac{\varepsilon + FIT(PROG_{el})}{\varepsilon + FIT(PROG_b)} \tag{3.5}$$

where: lr is the *learning rate* and ε is the *fitness constant*, $FIT(PROG_{el})$ is the fitness of the elitist individual, and $FIT(PROG_b)$ is the fitness of the best individual in current generation. The *learning rate* and ε influence the step size in adapting the PPA to the *elitist* or the *best* program.

PPA learns the structure of $PROG_b$ by updating the probability of all nodes according to Equation 3.6 while $P(PROG_b) < P_{TARGET}$.

$$P(PROG_b) = P(PROG_b) + c^{lr} \cdot lr \cdot (1 - P(PROG_b)) \tag{3.6}$$

Mutation of the PPA: At the end of each generation the nodes that were accessed to generate $PROG_b$ are mutated with probability P_{M_P}.

The probability is calculated as:

$$P_{M_P} = \frac{P_M}{(T+F) \cdot \sqrt{|PROG_b|}} \tag{3.7}$$

where P_M, the *mutation probability*, is a user defined parameter which defines the overall mutation probability, and $T + F$ defines the number of elements in the Terminal and Function sets, and $|PROG_b|$ is the number of nodes in $PROG_b$.

During mutation, each node used in $PROG_b$ is mutated with the given mutation rate.

$P_i[e] += mr \cdot (1 - P_i[e])$, where e is the element in the probability list of the selected node at index i and mr is a user defined mutation rate. Mutation rate increases the effect of mutation on the next generation which results in a diverse population. All mutated nodes are finally normalized.

Elitist Learning: During elitist learning the PPA is adapted towards $PROG_{el}$, the best individual of all the generations found so far. During this process creation and evaluation and mutation do not happen. The same equations are used to modify the PPA, except that in equations 3.3, 3.4, 3.5, and 3.6 we use $PROG_{el}$ instead of $PROG_b$.

Termination Criterion: The algorithm runs for a fixed number of *program evaluations (PE)* or until a solution with fitness better than *satisfactory fitness* is found.

3.2 Constructing a Plug-in Algorithm and Integrating it into Analysis Services

The hybrid algorithm (PI-CGP) is integrated into SQL Server Analysis Services (SSAS) as a data mining plug-in algorithm. The mining algorithm discovers the patterns in data and stores them in the form of mathematical expressions which are used when prediction requests are received in the form of queries on time series data.

Microsoft SQL Server Analysis Services (SSAS) provides the tools to design, create, and manage data mining solutions. In this environment, binding to data sources, creating models of the data, and deploying the models for predictive purposes is easily done.

Using SSAS releases the user from implementing data source access modules for training and forecasting. It also gives the user the ability to write queries in the DMX (Data Mining Extensions)

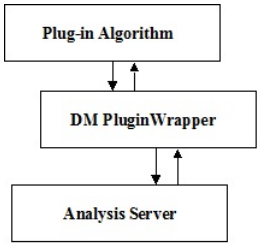

Figure 3.3: Analysis Server and plug-in algorithm communication interface

language to mine the models.

A mining model is an object that performs machine learning using a mining algorithm. It is represented as a subset of columns from the mining structure and how those columns are used (e.g. input, output, or both). The mining model also includes the mining algorithm and its parameters to perform learning on a mining structure.

A mining structure represents the data columns from the database tables which are available for a mining algorithm. This includes: column information (*i.e.* type, for example: Double , Long, DateTime) and column content (*i.e.* categorical or continuous).

Figure 3.3 presents a high-level view of how the plug-in algorithm communicates with Analysis Server.

Analysis Server uses a set of COM interfaces [1] to communicate with the plug-in algorithm [28]. The DMPluginWrapper.dll, the primary interop assembly, translates the COM calls from Analysis Server to the plug-in algorithm. The DMPluginWrapper has to be built and installed into the Global Assembly Cache (GAC) in order to be used by the plug-in algorithm.

Some of the COM interfaces are exposed by Analysis Server to be consumed in the plug-in algorithm and some must be implemented by the plug-in algorithm to be used by the server. For example, *IDMPersistenceWriter* is a COM interface exposed by the server and used by the plug-in algorithm to save the patterns discovered by the plug-in algorithm.

The following interfaces should be implemented by the plug-in algorithm:

[1] A set of functions that a COM (Component Object Model) class implements

- **IDMAlgorithmMetadata**: This describes the features of the plug-in algorithm.
- **IDMAlgorithm**: This performs the training, detects the patterns, and performs prediction. It also instantiates an *IDMAlgorithmNavigation* implementation.
- **IDMAlgorithmNavigation**: This allows the server to access the discovered patterns.
- **IDMPersist**: This stores the detected patterns.
- **IDMAlgorithmFactory**: This instantiates an *IDMAlgorithm implementation*.

A managed plug-in algorithm should be able to describe its features, discover the patterns in data, persist them, and use them for forecasting and finally expose the discovered patterns. These tasks are associated with the above-mentioned COM interfaces, so a managed plug-in algorithm is a class library that should implement the following classes:

- **Metadata class**: This exposes the algorithm information (e.g. name and description, supported types for input parameters and predicted values) and creates an instance of the *Algorithm class*.
- **Algorithm class**: This detects and persists the patterns in the data and uses them to predict. It also creates an instance of the *Navigator class*.
- **Navigator class**: This exposes the patterns found by the Algorithm class.

The above-mentioned classes are partially implemented in the primary interop assembly for managed plug-ins, *DMPluginWrapper.dll*, which is part of a managed API package provided by Microsoft, managed plug-in algorithm API for SQL Server 2005. The managed plug-in API consists of base classes and interfaces that should be derived from the managed plug-in algorithm.

In the implementation of a managed plug-in algorithm the following steps should be followed:

- Add a reference to the *DMPluginWrapper.dll*.

- Derive the Algorithm class from the *AlgorithmBase* class, the Metadata class from the *AlgorithmMetadataBase* class, and the Navigate class from the *AlgorithmNavigationBase* class.

- Add the plug-in algorithm to the GAC to be loaded by Analysis Services.

3.2.1 The plug-in algorithm life time

Analysis Services calls the *ValidateAttributeSet* method of the *Metadata* instance to check the validity of the model structure's attributes (*i.e.* check if at least one input attribute is provided). During the training, the *Metadata* object instantiates a new *Algorithm* object through the *CreateAlgorithm* method. Then Analysis Services calls the *InsertCases* method of the *Algorithm* object with a *caseSet* parameter which carries the training data contained in the mining structure. During the method run, the algorithm caches the *caseSet* to process the cases at one time and present them to the learning method. The learning algorithm iterates on the data to learn the pattern. Once the learning method completes, Analysis Services invokes the *SaveContent* method to save the discovered pattern.

The method for prediction is the *Predict* function implemented in the *Algorithm* class. Analysis Services parses DMX statements and provides the test cases in a similar way it did for the training cases. During prediction, Analysis Services presents each input case row to the *Predict* method and the result is returned to the user.

3.2.2 Connecting to External Data Sources

In order to retrieve the data from an external data source (e.g. a relational database), Analysis Services uses a data source object which contains the information such as the data source name and connection string. The connection string provides the data source provider name (e.g. SQL Server Native Client OLE DB provider) and other settings required by the provider for connection to the data source.

3.2.3 Data Mining Extensions

Data Mining Extensions (DMX) [28] is a query language created by Microsoft for creating, training, and predicting against data mining models.

DMX statements are used to: create new data mining structures and models, drop the existing ones and to predict against mining models.

To create a mining strucure:

```
CREATE MINING STRUCTURE IBM_Mining_Structure(
        IndexID LONG KEY,
        ClosePrice DOUBLE CONTINUOUS,
        NextDayPrice DOUBLE CONTINUOUS
)
```

To create a new mining model based on an existing mining structure (The mining structure must already exist before running the following script):

```
ALTER MINING STRUCTURE [IBM_Mining_Structure]
ADD MINING MODEL [IBM_Mining_Model](
        IndexID,
        ClosePrice,
        NextDayPrice PREDICT_ONLY
) USING [PI-CGP]
```

To create a mining model and automatically generate its mining structure at the same time:

```
CREATE MINING MODEL [IBM_Mining_Model](
    IndexID LONG KEY,
    ClosePrice DOUBLE CONTINUOUS,
    NextDayPrice DOUBLE CONTINUOUS  PREDICT_ONLY
)
USING [PI-CGP]
```

To drop a mining struture and model:

```
DROP MINING STRUCTURE [IBM_Mining_Structure]
DROP MINING MODEL [IBM_Mining_Model]
```

3.3 Development Environment

The class libraries were written in C# using the Microsoft .Net Framework version 3.5 and developed using Microsoft Visual Studio 2008. Microsoft Business Intelligence Development Studio (BIDS) was the development environment for creating the mining structure and model. Data storage and management was done using Microsoft SQL Server 2005.

3.4 Experimental Design

Two sets of problems, of varying types and difficulty, were selected to evaluate the performance of our hybrid algorithm. We started with a symbolic regression problem, then we applied our approach to forecast the stock market behaviour, specifically predicting the next trading day close price of stocks.

In sections 3.4.1 and 3.4.2 the experimental set up for these experiments is detailed.

3.4.1 Symbolic Regression Experiments

In the first two experiments, the performance of the hybrid algorithm was compared to a GP system with an improved search process approach [29]. The third experiment compared the performance of our approach to CGP with variable crossover approach [27]. The fourth experiment compared the performance of the proposed methodology against PIPE on the equation used in [8].

3.4.1.1 Experiment 1: Sextic Polynomial

In this experiment, the target expression is a Sextic polynomial ($x^6 - 2x^4 + x^2$). A sample of 50 equidistant data points taken from the interval [0,1] are used as fitness cases. The fitness of an

Table 3.2: Parameter values for experiments 1, 2, 3, and 4

Parameter	**Experiment**			
	Exp1	**Exp2**	**Exp3**	**Exp4**
Terminal Set	T = { x, R}	T = { x, R}	T = { x, R}	T = { x, R}
R range	[-1, 1]	[-1, 1]	[0, 1]	[0, 1]
Function Set	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div,$ $\sin,\cos,\exp$ $,rlog\}$
Population Size	500	500	2000	500
Number of Generations	51	200	25	20
Number of Runs	30	30	30	10
PI-CGP:				
Number of Rows	1	1	1	1
Number of Columns	10	10	10	200
Levels Back	10	10	10	200
Satisfactory Fitness, FIT_s	0.0	0.0	0.01	0.01
Elitist Update Probability, P_{el}	0.2	0.2	0.2	0.2
Learning Rate, lr	0.05	0.05	0.025	0.05
Fitness Constant, ε	0.01	0.01	0.01	1.0
Mutation Probability, P_M	0.2	0.2	0.2	0.2
Mutation Rate, mr	0.6	0.6	0.6	0.8
Random Constant Threshold, T_R	0.3	0.3	0.3	0.3

individual is calculated as the mean squared error over all the fitness cases. Let the o_i be the population member's value and t_i be the true function value on the i-th example of the fitness cases set, then the fitness f of the population member is calculated as follows:

$$f = \frac{1}{n}\sum_{i=1}^{n}(o_i - t_i)^2 \tag{3.8}$$

The ideal fitness is zero, meaning we want to minimize the difference between the fitness cases and the individual's result. The parameters chosen for this experiment are shown in Table 3.2.

3.4.1.2 Experiment 2: Polynomial of Degree 11

The target expression in our second experiment is a polynomial of degree 11. Similar to Experiment 1, a sample of 50 equidistant data points are taken from the interval [0,1]. The fitness of an individual is calculated as 3.8 and the ideal fitness is zero.

Below is the target polynomial:

$$(x+0.44)(x+0.54)(x+0.27)(x+0.04)(x+0.41)(x-0.43)(x-0.71)(x+0.82)(x+0.63)$$
$$(x-0.75)(x-0.91) \tag{3.9}$$

The parameters chosen for this experiment are shown in Table 3.2. We ran this experiment with several parameter sets (*i.e. rounds*) to observe how changing parameters would affect the performance of our system. Parameters under test included: learning rate, ε, population size, and number of generations.

3.4.1.3 Experiment 3: Sextic Polynomial over interval [-1,1]

The third symbolic regression experiment used the same equation as in experiment 1. The difference being that a sample of 50 data points are taken from interval [-1,1] as opposed to [0,1] in experiment 1. The fitness function is the sum of absolute values of the differences between the population member's value and the function value at each data point as opposed to the fitness function in experiment 1, which is the mean squared error over all the fitness cases.

Let the o_i be the population member's value and t_i be the true function value on the i-th example of the fitness cases set, then the fitness f of the population member is calculated as follows:

$$f = \sum_{i=1}^{n} |o_i - t_i| \tag{3.10}$$

The parameters chosen for this experiment are shown in Table 3.2. Several rounds of experimentation were run by varying the parameters such as: population size, number of generations, learning rate, mutation rate, and mutation probability.

3.4.1.4 Experiment 4: Non-trivial Function

The equation chosen for this experiment is a non-trivial function to benchmark the system.

$$f(x) = x^3 \cdot e^{-x} \cdot \cos(x) \cdot \sin(x) \cdot (\sin^2(x) \cdot \cos(x) - 1) \tag{3.11}$$

A sample of 101 equidistant data points are taken from interval [0,10]. The cost function is calculated as (3.10).

The parameters chosen for this experiment are shown in Table 3.2. Experimenting with different parameter settings (rounds) was done on parameters such as: population size, number of columns, and mutation rate.

3.4.2 Stock Market Forecasting Experiments

In the following two experiments the performance of our approach in prediction of the next trading day stock price was compared to a GP approach and a fusion model combining HMM, ANN, and GA.

3.4.2.1 Genetic Programming Approach

In this experiment, the performance of our approach was compared to the approach which used GP to predict the stock price of the next trading period based on the changes of stock prices of the past N periods [30]. Where N is a user-defined variable which defines the number of past accessible daily prices of a stock.

The parameters chosen for this experiment are shown in Table 3.3. The training data consists of stock prices for IBM in the period 01 November 2002 to 19 August 2003. The test data consists of stock prices for IBM in the period 20 August 2003 to 12 November 2003. Of the total collected data, 77% is used as the training data and 23% as the test data.

Table 3.3: Parameter values for stock market forecasting

Parameter	**Experiment**	
	Exp1	**Exp2**
Terminal Set	T = {Close price, R}	T = {Close price, R}
R range	[0, 1)	[0, 1)
Function Set	F = $\{ +, -, \times, protected\ division, \sin, \cos, \exp$ $, protected\ log, power, protected\ square\ root \}$	ditto
Population Size	3000	20
Number of Generations	66	100
Number of Runs	10	10
PI-CGP:		
Number of Rows	1	1
Number of Columns	10	10
Levels Back	10	10
Satisfactory Fitness, FIT_s	0.01	0.01
Elitist Update Probability, P_{el}	0.2	0.2
Learning Rate, lr	0.05	0.05
Fitness Constant, ε	0.01	0.01
Mutation Probability, P_M	0.2	0.2
Mutation Rate, mr	0.6	0.6
Random Constant Threshold, T_R	0.3	0.3

The fitness function is the sum of absolute values of the differences between the actual daily close price and the predicted value.

3.4.2.2 HMM, ANN, and GA approach

In this experiment, we compared the performance of our approach to a fusion model which combines the Hidden Markov Model (HMM), Artificial Neural Network (ANN), and Genetic Algorithms (GA) to forecast financial market behaviour [31]. In the fusion model, the ANN transforms the actual observation sequence (*i.e* training data in form of open, high, low, and close price) and GA is used to optimize the initial parameters for HMM. This fusion model tries to find other days in the historical data which show similar behaviour to the current day and adds the weighted average of price differences to the current day price.

We have used the daily stock price of Apple Inc., International Business Machines Corp. (IBM), and Dell Inc. Data has been collected from *http://www.google.com/finance*.

The parameters chosen for this experiment are shown in Table 3.3. The training and test data consists of stock prices for International Business Machines Corp., Apple Inc., and Dell Inc. in the period 10 February 2003 to 10 September 2004 and 13 September 2004 to 21 January 2005, respectively. Of the total collected data for each stock, 81% is used as the training data and 19% as the test data.

The fitness function is the sum of absolute values of the differences between the actual daily close price and the predicted value.

Chapter 4

Results and Discussion

This chapter presents the results of the experiments described in Chapter 3. We begin with the symbolic regression set of experiments. First, we compare our system performance to a GP system with an improved search process in approximating polynomials of degree 6 and 11, then we compare our system performance to a CGP with variable crossover approach in approximating a polynomial of degree 6. Next, the performance of PI-CGP is compared against PIPE. Finally, we compare our system performance in prediction of the next trading close price of stock prices to a GP approach and a fusion model.

4.1 Symbolic Regression Results

In this set of experiments, we use fitness to compare the systems under test. The fitness of an individual is calculated as the mean squared or absolute error over all the fitness cases. The results represent values averaged over the number of runs.

4.1.1 Experiment 1: Sextic Polynomial

In this experiment the equation being approximated is $x^6 - 2x^4 + x^2$. Table 4.1 shows the calculated parameters for each approach.

In order to determine which methodology performed better, the *p-value* was calculated using the *Student's T-test*.

The mean fitness of PI-CGP (0.048) was less than the mean fitness of GP with improved search (2.002). The *p-value* for this experiment was 4.2502, which means that the difference between the

Table 4.1: Best fitness measure of runs

Methodology	**Min.**	**Max.**	**Mean**	**Std Dev.**
PI-CGP	0.0	0.138	0.048	0.049
GP with improved search	0.254	2.455	2.002	0.461

Table 4.2: Parameter settings for the polynomial of degree 11 experiment.

Parameters	**Setting**						
Round[1]	R1	R2	R3	R4	R5	R6	R7
Population Size	500	500	500	500	500	1000	1000
Number of Generations	51	100	100	100	200	25	200
Fitness Constant, ε	0.01	0.01	0.01	1E-06	0.01	0.01	0.01
Learning Rate, lr	0.05	0.01	0.05	0.05	0.05	0.05	0.05

means was significant with $\alpha = 0.1\%$.

4.1.2 Experiment 2: Polynomial of Degree 11

In this experiment, the test equation is Equation 3.9.

The result of this experiment was that the PI-CGP algorithm did not find the target expression in any of the trials. Several parameter sets were tried and the results can be found in Table 4.3. Parameters under test include: learning rate, ε, population size, and number of generations.

Table 4.2 shows the different parameter settings used and Table 4.3 shows the results based on the best fitness of runs when different parameter settings were applied.

When the *p-value* between different rounds was calculated the difference between the means

[1] As mentioned in Section 3.4.1.2, round represents a given parameter setting used in an experiment.

Table 4.3: Results for different parameter settings for the polynomial of degree 11 experiment.

Rounds	**Best**	**Worst**	**Mean**	**Std Dev.**
R1	0.00008	0.00274	0.00079	0.00062
R2	0.00016	0.00216	0.00118	0.00051
R3	0.00006	0.00299	0.00069	0.00086
R4	0.00006	0.00299	0.00077	0.00082
R5	0.00004	0.00299	0.00073	0.0008
R6	0.00006	0.00305	0.00059	0.00064
R7	0.00005	0.00243	0.00035	0.00048

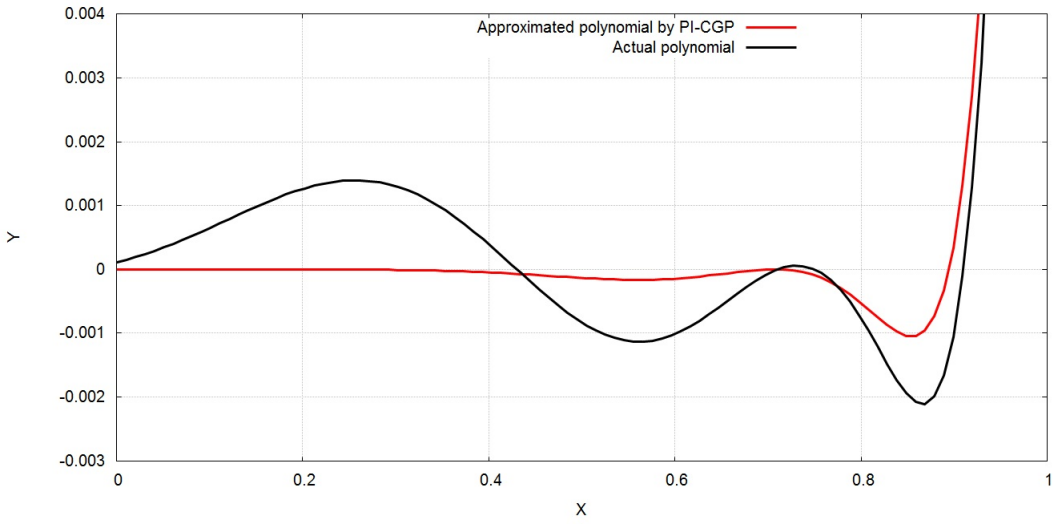

Figure 4.1: Approximated polynomial by PI-CGP versus the actual polynomial

for R5 and R2, R1 and R2, R3 and R2 was significant with $\alpha = 2\%$.

To find the best solution of all rounds we proceeded as follows: Since Mean(R5) $<$ Mean(R2), Mean(R1) $<$ Mean(R2), and Mean(R3) $<$ Mean(R2), we selected R5, R1, and R3, where $Mean(R_i)$ denotes the mean of the fitness values of round R_i. The difference between the Mean(R5) and Mean(R1), Mean(R5) and Mean(R3), Mean(R1) and Mean(R3) was not significant.

Then the fitness of the best individual of R5, R1, and R3 was compared. The best individual of R5 had the best fitness (the lowest), therefore R5 was picked as the best solution among all rounds.

Below is the simplified version of the best individual in R5:

$$x^{14} - 1.8x^{12} + 1.05x^{10} - 0.2x^{8} \tag{4.1}$$

Figure 4.1 shows the approximated polynomial versus the actual polynomial. Figure 4.2 shows the fitness improvement for PI-CGP for different parameter settings. The horizontal axis represents the generation number and the vertical axis is the fitness value of the best individual (averaged over runs).

Increasing the population size from 500 in R5 to 1000 in R7 with all other factors remaining

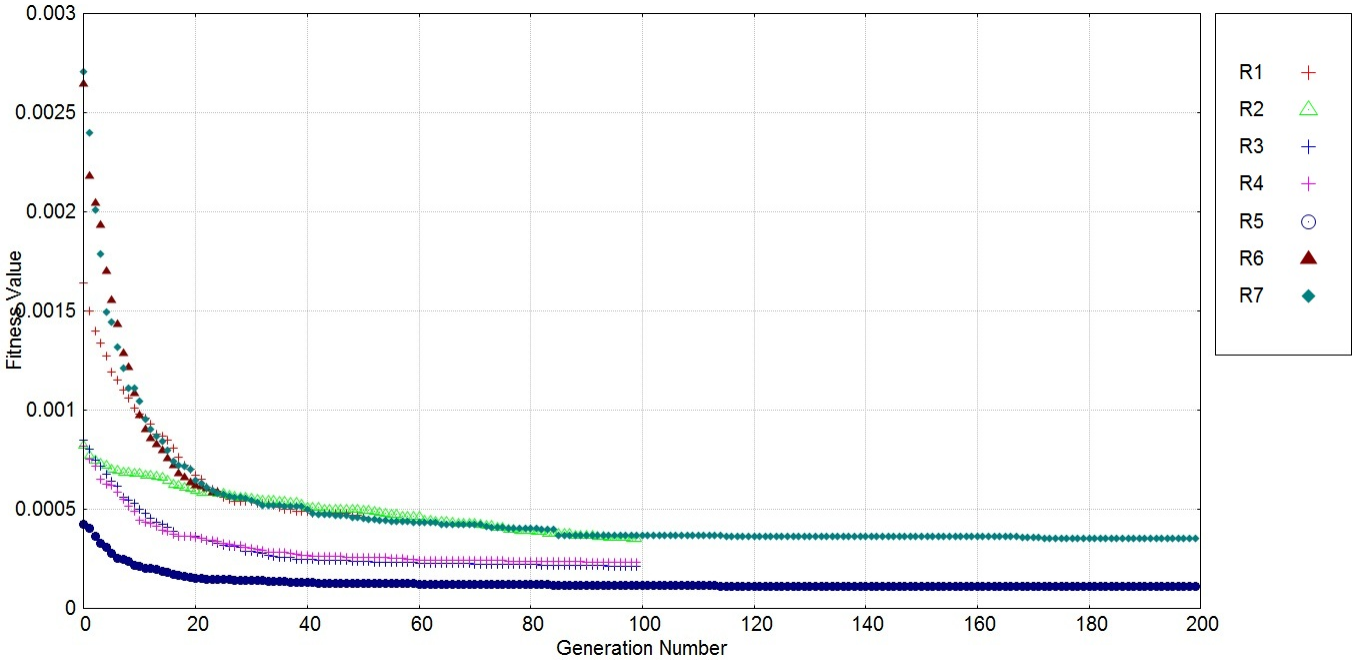

Figure 4.2: Fitness improvement for PI-CGP on the polynomial of degree 11

the same did not contribute to improving the average fitness. In R7 the average fitness in generation 0 was 0.0027 compared to 0.00042 in R5 and the average fitness in the last generation (e.g. generation number 199) in R7 was 0.00035 compared to 0.0001 in R5 but the rate of average fitness improvement in R7 (86.88%) was higher than that in R5 (73.82%). The rate of average fitness improvement is calculated as:

$$100 * \frac{AvgFitness_0 - AvgFitness_{Last}}{AvgFitness_0} \tag{4.2}$$

In equation 4.2, $AvgFitness_0$ is the average fitness in generation number 0 and $AvgFitness_{Last}$ is the average fitness in the last generation.

Increasing the number of generations from 51 in R1 to 100 in R3 and 200 in R5 with all other factors remaining the same contributed to improving the fitness (e.g. decreasing the average fitness value). The average fitness in R5 in generation 0 was 0.0004 compared to 0.0008 in R3 and 0.0016 in R1 and in generation 50 the average fitness in R5 was 0.0001 compared to 0.0002 in R3 and

0.0004 in R1. The rate of improvement between the first and last generation in all the R1, R3, and R5 was the same (70%).

Increasing the learning rate from 0.01 in R2 to 0.05 in R3 with all other factors remaining the same increased the average fitness improvement from 56.9% to 75.17% and the change in the mean fitness value was statistically significant with $\alpha = 2\%$.

Decreasing the fitness constant (ε) from R3 to R4 causes the learning algorithm to take smaller steps toward programs with better fitness by decreasing the target probability of $PROG_b$, but the difference in mean fitness value was not statistically significant and the average fitness improvement between the first and the last generations in these runs was the same (72%).

4.1.3 Experiment 3: Sextic Polynomial over interval [-1,1]

This experiment uses the same equation as experiment 1 with the parameter settings used in [27] (shown in Table 4.4). This experiment compares the average number of generations required for PI-CGP to converge against CGP with variable crossover.

In CGP with variable crossover, the algorithm is said to converge when for all the data points in the interval [-1,1] the absolute difference between the function and the individual's value is less than 0.01. For PI-CGP, the algorithm is considered to converge when the fitness function value (Equation 3.10) is less than 0.01, which is a tighter convergence criterion compared to that of CGP with variable crossover. The drawback with this criterion could be the possibility of some of the absolute values being more than 0.01, but the sum of them is less than 0.01.

During the CGP with variable crossover run, the crossover rate started at 90% in the first generation and reduced linearly such that by the 180th generation the crossover rate was 0%.

Figure 4.3 shows the average convergence for PI-CGP based on the parameters shown in Table 4.4. Figure 4.4 shows the average convergence for CGP with variable crossover rate.

CGP with variable crossover showed better convergence than PI-CGP. The average number of

Table 4.4: Configuration used in CGP with variable crossover technique for $x^6 - 2x^4 + x^2$

Parameters	**Setting**
Terminal Set	T = { x, R}
R range	[0, 1]
Function Set	$\{ +, -, \times, \div \}$
Population Size	50
Number of Generations	1000
Number of Runs	1000
Number of Rows	1
Number of Columns	10
Levels Back	10

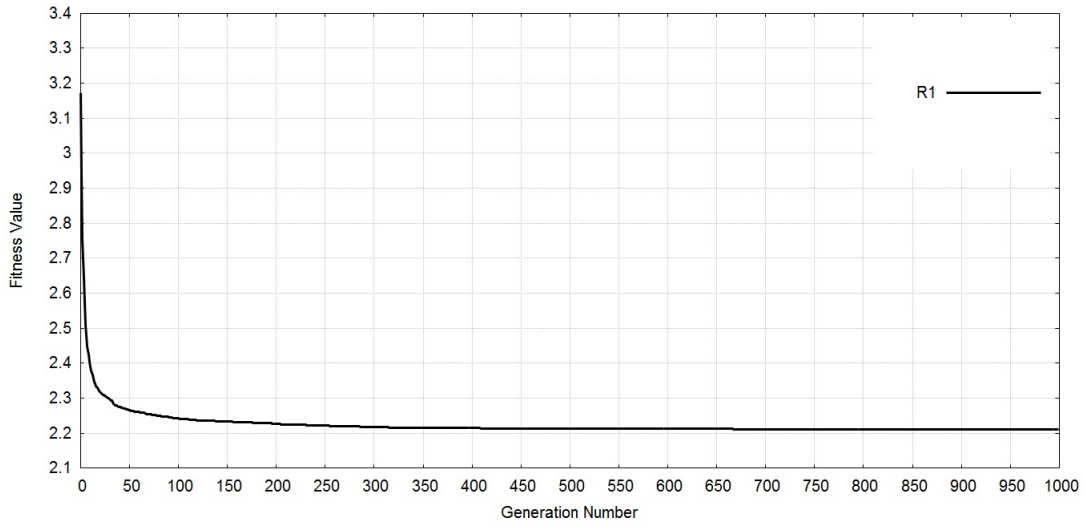

Figure 4.3: Average convergence for PI-CGP based on the parameters in Table 4.4 on $x^6 - 2x^4 + x^2$

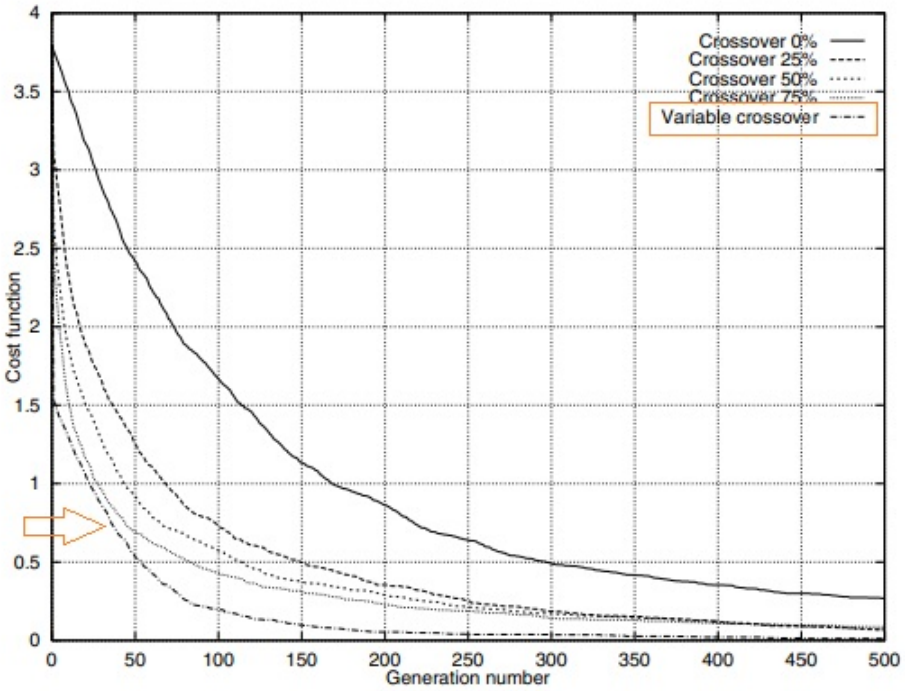

Figure 4.4: Average convergence for CGP with variable crossover rate [27]

generations required for PI-CGP to converge is 179 compare to 47 (CGP with variable crossover), 71 (CGP with crossover rate of 75%), 57 (CGP with crossover rate of 50%), 84 (CGP with crossover rate of 25%), and 168 (CGP without crossover).

The average fitness value for our approach was higher (less fit) than the other approach. It is noticeable in Figure 4.3 that the convergence rate of PI-CGP after generation number 300 gradually decreases.

Several rounds of experimentation were run on PI-CGP while varying the following parameters: population size, learning rate, mutation rate, and mutation probability. Tables 4.5 and 4.6 show the parameter settings for each round, Table 4.7 shows the results of the runs, and Figure 4.5 shows the average convergence for each run. Where the number of generations varied we kept the number of program evaluations [2](PE) consistent. In the figure only the average convergence for the initial 100 generations is shown since the number of generations is different for different

[2]It is the product of population size and number of generations.

Table 4.5: Configuration for PI-CGP for R1 to R9

Parameters	**Setting**								
Round	R1	R2	R3	R4	R5	R6	R7	R8	R9
Population Size	50	50	500	500	500	500	500	500	1000
Number of Generations	1000	1000	100	100	100	100	100	100	50
Mutation Rate, mr	0.6	0.8	0.2	0.4	0.6	0.8	0.6	0.6	0.6
Learning Rate, lr	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Mutation Probability, P_M	0.2	0.2	0.2	0.2	0.2	0.2	0.4	0.5	0.2
Number of Runs	1000	30	30	30	30	30	30	30	30

Table 4.6: Configuration for PI-CGP for R10 to R12

Parameters	**Setting**		
Round	R10	R11	R12
Population Size	1000	1000	2000
Number of Generations	50	50	25
Mutation Rate, mr	0.6	0.6	0.6
Learning Rate, lr	0.01	0.025	0.025
Mutation Probability, P_M	0.2	0.2	0.2
Number of Runs	30	30	30

rounds and for the rounds with more than 100 generations the population converges gradually.

Figure 4.5 shows that experimenting with different settings for parameters such as population size has improved the average convergence of the round compare to that of R1 (the round with the same setting as in [27]), as the lines corresponding to the rounds show below the R1 line. (e.g. Increasing the population size from 50 in R1 to 1000 in R9, R10, R11 and 2000 in R12)

When the *p-value* between different rounds was calculated the difference between the means for R1 and R9, R1 and R10, R1 and R11 was statistically significant with $\alpha = 1\%$. The difference between the means for R1 and R12 was significant with $\alpha = 0.1\%$). Among all the rounds with better convergence rate than R1, R12 was picked as the best round since it had the lowest mean value and better average convergence than the others. Comparing the average convergence between R12 and CGP with variable crossover shows that CGP with variable crossover has better average convergence (higher).

The average convergence of R12 for the initial 100 generations is higher (better) than the CGP

Table 4.7: Results for each round based on different parameter settings in Table 4.5 and 4.6

Rounds	Best	Worst	Mean	Std Dev.	Number Of Successful Runs	Average Generations Over Successful Runs
R1	0.0	3.80385	2.21145	0.40741	12	179
R2	2.2596	3.80385	2.35816	0.27139	0	-
R3	0.0	2.34226	2.02859	0.51583	1	28
R4	0.85254	2.34299	2.02197	0.47199	0	-
R5	0.0	2.35816	1.95434	0.65740	2	23
R6	0.05713	2.34258	2.12956	0.46863	0	-
R7	0.07431	2.34233	2.03174	0.64608	0	-
R8	0.6948	2.34327	2.04346	0.44663	0	-
R9	0.0	2.34233	1.81934	0.67688	2	11
R10	0.0	2.2761	1.83822	0.62591	2	44
R11	0.0	2.34211	1.78262	0.82295	3	29
R12	0.0	2.26197	1.64014	0.73765	3	14

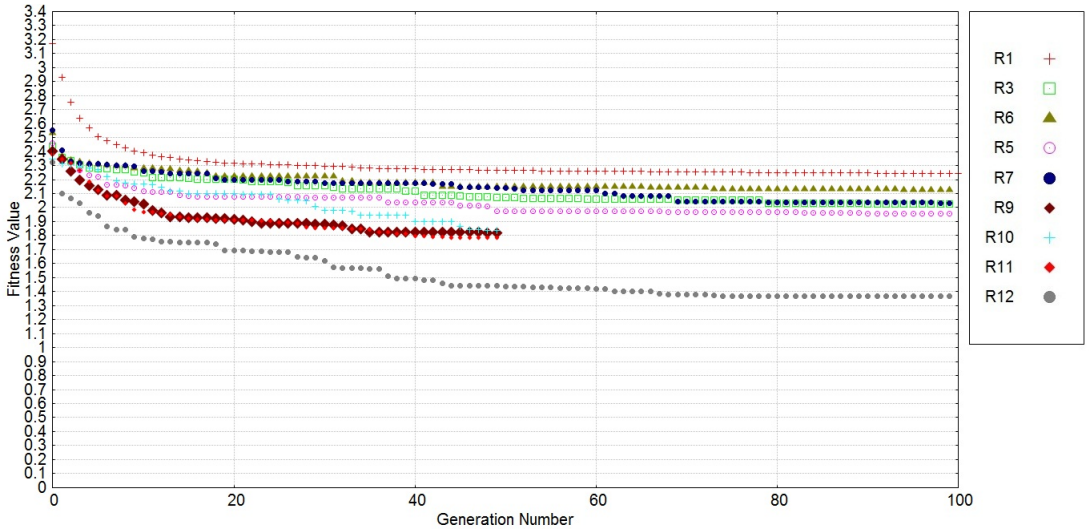

Figure 4.5: Average convergence for PI-CGP with different parameter settings on $x^6 - 2x^4 + x^2$

Table 4.8: Parameter settings for PI-CGP on equation 4.3

Parameters	**Setting**								
Round	R1	R2	R3	R4	R5	R6	R7	R8	R9
Population Size	10	10	10	50	100	100	100	100	500
Number of Generations	100	100	10	20	10	100	200	100	20
Number of Columns	50	100	100	100	100	100	100	200	200
Mutation Rate, *mr*	0.6	0.6	1.0	0.8	0.8	0.8	0.8	0.8	0.8
Number of Runs	21	21	10	10	10	10	10	10	10

with a crossover rate of 0% (*i.e.* no crossover) but for the rest of the generations it is lower.

4.1.4 Experiment 4: Non-trivial Function

In this experiment the following equation is approximated:

$$x^3 \cdot e^{-x} \cdot \cos(x) \cdot \sin(x) \cdot (\sin^2(x) \cdot \cos(x) - 1) \tag{4.3}$$

Experiments were performed on PI-CGP using the parameters outlined in Table 4.8. The fitness improvement for each round is shown in Figure 4.6. As it is shown in the figure, the R9 line is below the others and this indicates that the average fitness has been improved more than the other rounds. Increasing the population size contributed positively to average fitness improvement.

When the *p-value* between different rounds was calculated the difference between the means for R9 and the rest of the runs was significant with $\alpha = 5\%$ with the exception R6 and R7. The results are shown in Table 4.9.

Figure 4.7 shows the best approximated equation in R9, the best approximated equation in all the runs, and the actual equation respectively.

In this experiment the PI-CGP algorithm did not find the target expression in any of the trials.

4.2 Stock Market Forecasting

This set of experiments applies our approach to the problem of stock market forecasting. Specif-

Table 4.9: Results for each round based on different parameter settings in Table 4.8

Rounds	Best	Worst	Mean	Std Dev.
R1	17.742	43.930	22.378	6.580
R2	15.496	27.311	20.995	2.228
R3	20.459	28.394	21.511	2.350
R4	14.131	20.633	19.576	1.878
R5	19.706	20.504	20.356	0.235
R6	12.419	20.462	18.722	2.491
R7	14.165	20.462	18.506	2.293
R8	11.750	20.464	19.155	2.501
R9	14.172	19.835	16.764	1.877

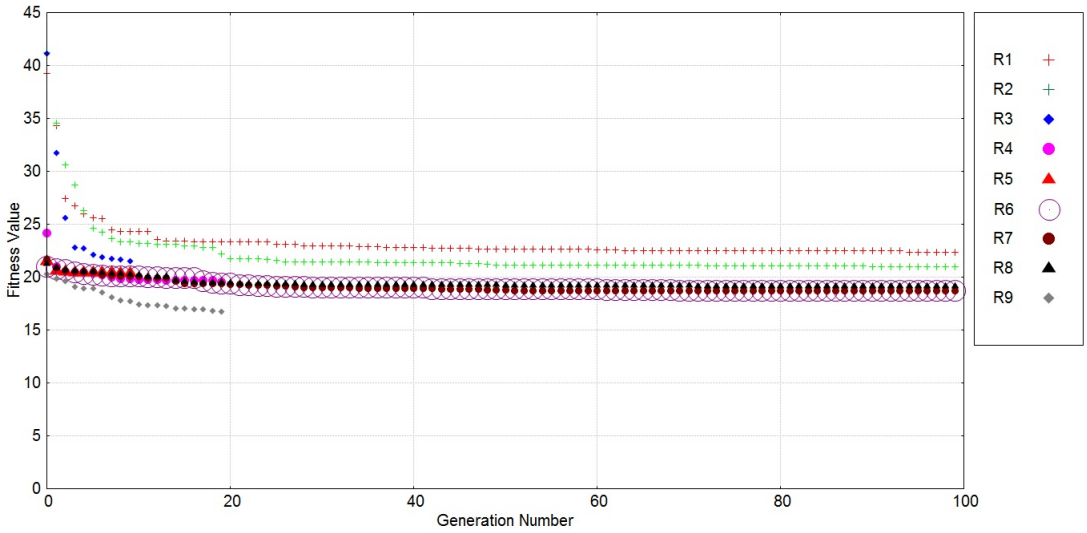

Figure 4.6: Fitness improvement for PI-CGP with different parameter settings on equation 4.3

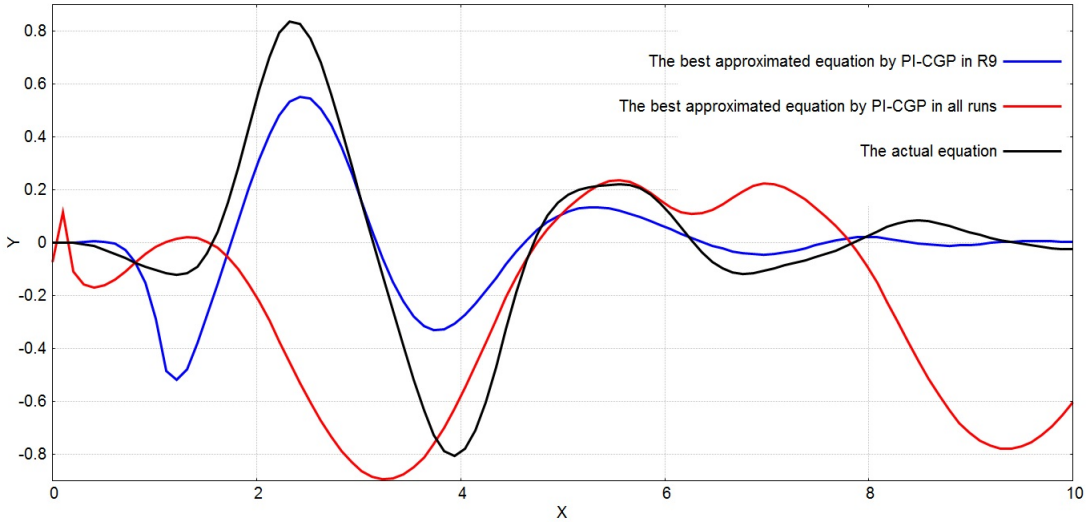

Figure 4.7: The best approximated equation by PI-CGP in R9, all runs, and the actual equation

ically, using historical prices to predict the next trading day price. In measuring our system's performance, we use Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) as our performance metrics.

Equations 4.4 and 4.5 show how to calculate MAE and MAPE.

$$Mean\ Absolute\ Error = \frac{1}{r}\sum_{i=1}^{r}|p_i - y_i| \tag{4.4}$$

$$Mean\ Absolute\ Percentage\ Error = \frac{100\%}{r}\sum_{i=1}^{r}|\frac{y_i - p_i}{y_i}| \tag{4.5}$$

Where r is the total number of test data sets, y_i is the actual stock price on day i, and p_i is the forecast stock price on day i.

4.2.1 Genetic Programming approach

We ran several trials with different population sizes and consistent number of generations (shown in Table 4.10). The *p-value* between R1 and R4, R2 and R4, and R3 and R4 is significant with $\alpha = 5\%$. The best round was R6 and the best-of-run individual in this round with the fitness value of 204.805 was:

$$ClosePrice - 0.32\exp^{\sin(ClosePrice)}\sin(ClosePrice^4) \tag{4.6}$$

Table 4.10 shows that as the population size increases the fitness improves.

Table 4.11 and Table 4.12 present the results from experiments on IBM stock prices. The results show the forecast accuracy of our approach compared to the GP approach [30] using the same cross-validation period data. Our training period data does not include the amount of data covered by the window period in the GP approach as the training set should contain only 200 days.

On the cross-validation period, the PI-CGP system performs better than the GP approach with

Table 4.10: Results for each round based on different population sizes

Rounds	Population Size	Number of Generations	Best	Worst	Mean	Std Dev.
R1	1000	200	212.1872	214.7924	213.8668	0.9322
R2	1500	200	212.3212	214.7936	213.6303	0.9077
R3	2000	200	212.0545	214.7914	213.7477	0.8579
R4	2500	200	208.6726	214.7521	212.2519	1.8434
R5	3000	200	206.9992	214.4921	212.5282	1.9871
R6	3000	66	204.8055	214.7919	212.1148	3.0357

Table 4.11: Forecast accuracy comparison on cross-validation period. Training range: 1 November 2002 to 19 August 2003

Stock name	MAE for the cross-validation period data		
	PI-CGP	GP with N = 300	GP with N = 450
IBM	0.82911	1.272369	0.765702

N of 300 (The result was significant with $\alpha = 5\%$). The forecast accuracy of the GP with N of 450 is slightly better than PI-CGP but not statistically significant (*t-value* = 0.31).

On the training period, the PI-CGP system performs better than the GP approach with N of 300 (The result was significant with $\alpha = 0.1\%$) The forecast accuracy of PI-CGP is slightly better than GP approach with N of 450 but not statistically significant (*t-value* = 0.44).

Increase in the size of N results in more historic price data available to the algorithm and could have an effect on finding better patterns in the price data. Therefore to make the two experiments use the same date range, we increased the training period range: 02 January 2001 to 19 August 2003 and calculated the MAE on cross-validation period. The result MAE was: 0.770659, which is slightly better but not a statistically significant difference from the forecast accuracy when the training period was from 01 November 2002 to 19 August 2003 (0.82911) and not a statistically significant difference from the forecast accuracy generated by GP with N of 450 (MAE:0.765702). The MAE on training range was:1.07924 which is slightly higher but not a statistically significant difference from 1.0239 when the training period was from 01 November 2002 to 19 August 2003.

Figure 4.8 shows the predicted close prices by PI-CGP and actual close prices for IBM.

Table 4.12: Forecast accuracy comparison on training period. Training range: 1 November 2002 to 19 August 2003

Stock name	MAE for the training period data		
	PI-CGP	GP with N = 300	GP with N = 450
IBM	1.0239	1.514626	1.083708

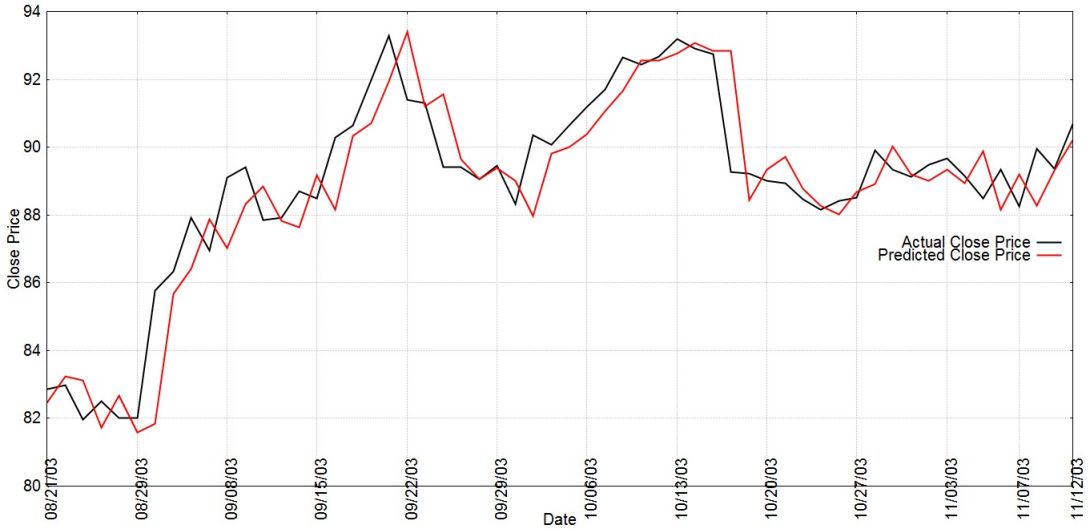

Figure 4.8: IBM close price forecast by PI-CGP

4.2.2 HMM, ANN, and GA approach

In this experiment, we tested PI-CGP with the same parameter values set in [31]. Table 4.13 presents the experiment results for each of the three stocks.

Looking at Table 4.13, we find that the accuracy of the PI-CGP approach is slightly better but not a statistically significant difference from that of the fusion model for IBM and Apple, while for Dell the accuracy of the fusion model is slightly better but not a statistically significant difference from that of the PI-CGP model.

Table 4.13: Forecast Accuracy Comparison between PI-CGP and the fusion model

Stock name	MAPE in forecast for the 91 test datasets	
	PI-CGP	The fusion model
Apple Computer Inc.	1.79940	1.9247
IBM	0.65403	0.84871
Dell Inc.	0.8136	0.699246

Table 4.14: The fitness and the representation of the best individual by PI-CGP

Stock name	Min Fitness	Individual equation
Apple Computer Inc.	81.94234	$ClosePrice + \frac{0.2}{ClosePrice}$
IBM	316.16017	$\frac{0.83 \times \frac{0.87}{ClosePrice}}{0.01 \times ClosePrice} + ClosePrice$
Dell Inc.	154.20978	$ClosePrice + 0.9^{ClosePrice}$

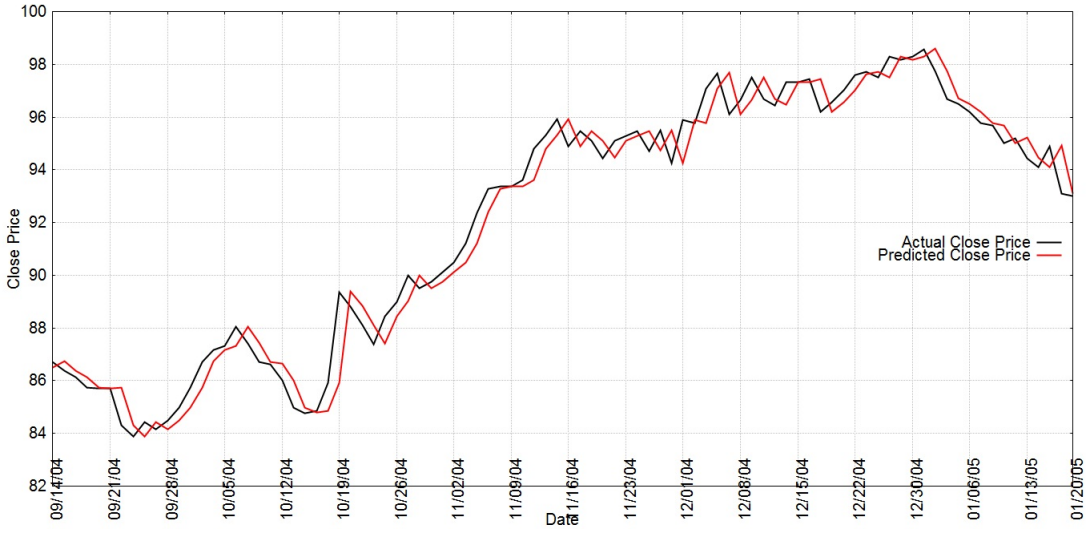

Figure 4.9: IBM close price forecast by PI-CGP

The fitness and the representation of the best individual for each stock is shown in Table 4.14.

We tested our system while varying population size and number of generations. The IBM data set was used for these tests. The population sizes used were: 10, 50, 100, and 500. The number of generations: 200 and 1000. We kept the number of program evaluations consistent. The calculated *p-value* between the rounds was not statistically significant.

The estimated value of the PI-CGP model for three stocks are plotted in Figures 4.9, 4.10, and 4.11.

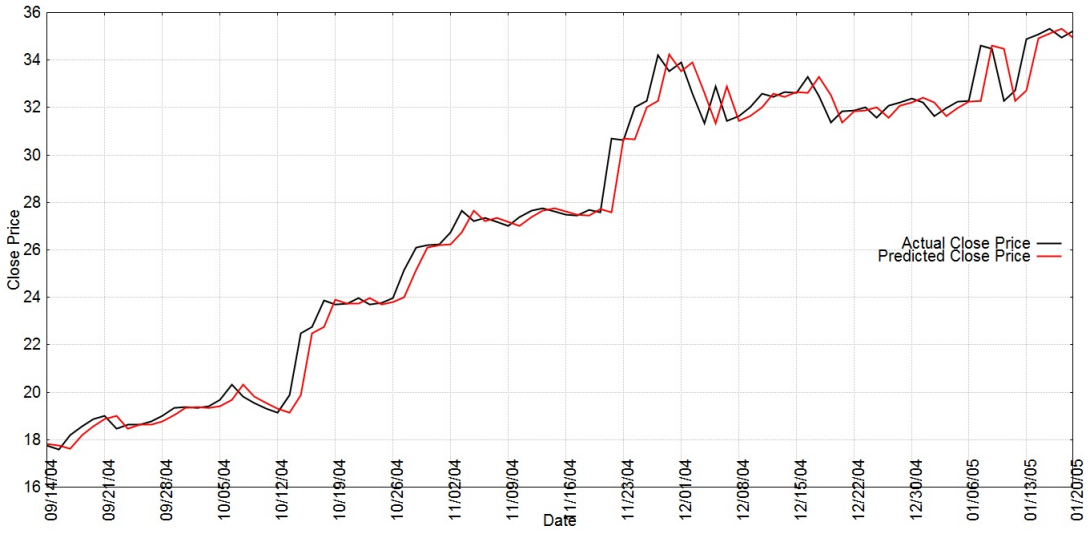

Figure 4.10: Apple close price forecast by PI-CGP

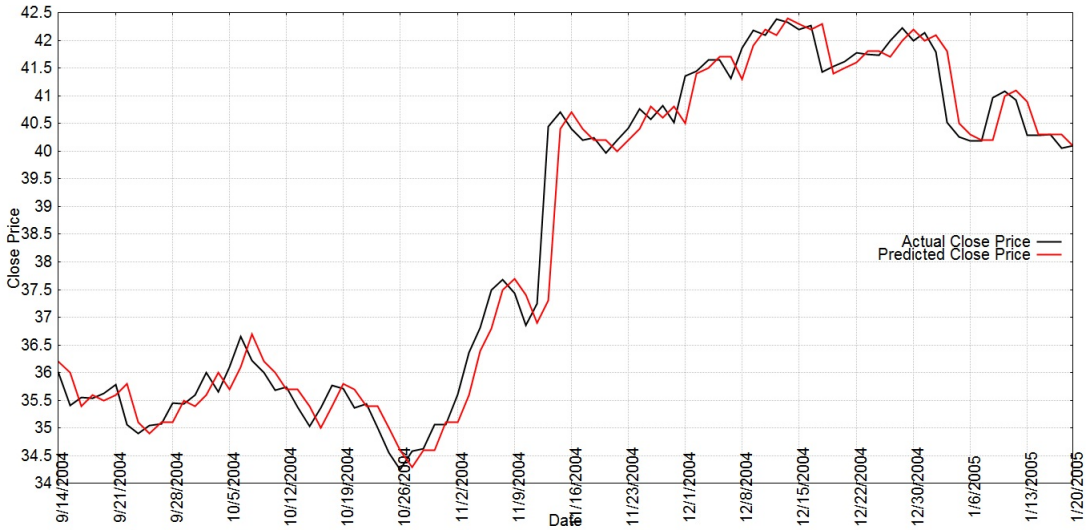

Figure 4.11: Dell close price forecast by PI-CGP

Chapter 5

Conclusions and Future Work

This work introduced PI-CGP, a hybrid algorithm using the structure of CGP genotypes and the probabilistic modeling of PIPE. The performance of the algorithm was evaluated on several benchmarks from the domains of symbolic regression and stock market forecasting.

Performance tests of PI-CGP uncovered mixed results, but overall the results show promise.

On symbolic regression tests with polynomial of degree six, PI-CGP outperformed a GP implementation with modified search. On a polynomial of degree 11, PI-CGP failed to find a target expression. We hypothesize that when the system failed it had become stuck in a local minimum and therefore we attempted to increase the breadth of the search by increasing the population size which contributed positively to the rate of average fitness improvement. We also increased the number of generations to give more time to the population to evolve which resulted in improving the average fitness value. Increasing the learning rate also improved the average fitness and the change in the mean fitness value was statistically significant.

On another symbolic regression problem, we compared the average convergence rate of our approach against CGP with variable cross-over. In this case, PI-CGP had a slower convergence rate. One possible reason for this result is premature convergence. Premature convergence is when programs in the population become similar, but are not the optimal solution to the problem. Increasing the probability of mutation and the mutation rate did not improve convergence but increasing the population size did improve convergence.

In the final symbolic regression problem (the non-trivial function), we compared the performance of our approach against PIPE. PI-CGP failed to find the target model. A possible reason is that the run required more program evaluations. The number of program evaluations used dur-

ing the PIPE run was much higher than the PI-CGP run. Unfortunately runs using more program evaluations failed to finish at the time of writing.

We integrated PI-CGP into SSAS as a plug-in algorithm. Our plug-in algorithm was used to create a data mining solution. In this work, we used PI-CGP to discover patterns in the price time series of several stocks and to forecast stock prices by one day.

During the forecasting problems, we compared our approach to a GP system and a fusion model forecasting tool. In the test against the GP system, the accuracy of our approach on the cross-validation and training period was significant when the window period of the GP system was 300. On the cross-validation period GP with window period 450 performed slightly better than our approach but not significantly better. On the training period the forecast accuracy of our approach was better than GP with window period of 450 but not significantly better.

In the test against the GP system, we found that training our model on a wider range of historical price data would result in better MAE of forecast on the cross-validation period.

In the test against the fusion model, a combination of ANN, GA, and HMM, we found that the accuracy of our approach is similar to that of the fusion model. Specifically, the forecast accuracies of our model were better but not statistically significant for Apple and IBM. The forecast accuracy of the fusion model was slightly better but not statistically significant than our approach for Dell. Considering PI-CGP has not been customized toward forecasting, it is significant that it was comparable to a system designed with specific tools for stock market forecasting (e.g. Hidden Markov Model (HMM) is a statistical tool which has been extensively used for time series forecasting).

5.1 Contributions

We have presented PI-CGP, a novel probabilistic model-building CGP which integrates the CGP program representation with the PIPE learning algorithm. PI-CGP does not require pruning to control program size as does PIPE and bloat does not happen in PI-CGP.

We have also integrated the PI-CGP model into SSAS as a plug-in algorithm. In order to discover patterns, we applied the plug-in algorithm to daily closing stock price time series of three stocks. The results show the viability of EDA-GP based systems in the domain of stock market forecasting since PI-CGP, a generic EDA-GP system, was competitive with a system which contains forecasting specific tools. It is possible that by training those specific tools with PI-CGP the results would be an improvement over the fusion model.

5.2 Future Work

Our results on the symbolic regression set of experiments have shown signs of premature convergence to local minimum. Attempts to increase the population diversity by tuning the population size, mutation rate, and mutation probability did not succeed in finding the target expression and improving the convergence to be the at the same level or better than the other approach. Future work could look into techniques to prevent premature convergence such as the approach presented in [32].

Tuning the predefined set of parameters (*i.e.* finding appropriate values) of an Evolutionary Algorithm for a given problem is essential, as it it is beneficial to better performance of the algorithm. It would worth trying an automated parameter tuning approach, *Bonesa* [33], for symbolic regression and stock market forecasting experiments.

It would be interesting to use PI-CGP with a trading agent to buy and sell stock. PI-CGP would be used to generate trading rules by combining the technical indicators [34], which are used to understand stock market behavior. Such a system could be evaluated based on return on investment [35]. This problem could also be considered as a classification task in data mining, as it will generate rules in the form of $If < condition_1 > and < condition_2 > and ... < condition_n > then < buy\ signal >$.

The tests described in this work used the daily close price of stocks, a worthwhile experiment

would test if the predictions could be improved with additional inputs, for example: open price, high price, low price, and volume. It would also help to get an insight to the relationship between these parameters.

Testing the model on data sets from different time periods would be well worth trying and applying the model to more stocks price time series will allow us to obtain more accurate conclusions about the performance of our model.

Artificial Neural Networks (ANN) have been widely used in time series forecasting, so an interesting future work would be to compare the PI-CGP approach with ANNs.

Using PI-CGP approach for training the ANN is an interesting avenue to be explored, as the combination of GA and ANN has outperformed other models which used statistical and technical models to train the ANN [36].

Our system can be customized toward forecasting by linking it with HMM. PI-CGP may optimize the HMM parameters such as: The number of hidden states, the number of unique observations per state, the state transition probability distribution, the emission probability distibution, and the initial state distribution. HMM is used to identify the patterns in data.

Bibliography

[1] Shelly X. Wu and Wolfgang Banzhaf. The use of evolutionary computation in knowledge discovery: The example of intrusion detection systems. In Satchidananda Dehuri and Sung-Bae Cho, editors, *Knowledge Mining using Intelligent Agents*, pages 27–59. WorldSciBook, Dec 2010.

[2] D. Srinivasan and V. Sharma. Evolutionary computation and economic time series forecasting. In *IEEE Congress on Evolutionary Computation*, pages 188 –195, 2007.

[3] Richard J. Povinelli. Using genetic algorithms to find temporal patterns indicative of time series events. In *in GECCO 2000 Workshop: Data Mining with Evolutionary Algorithms*, pages 80–84, 2000.

[4] Dongsong Zhang and Lina Zhou. Discovering golden nuggets: data mining in financial application. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,*, 34(4):513 –522, Nov 2004.

[5] Julian F Miller. *Cartesian Genetic Programming*. Springer Berlin Heidelberg, 2011.

[6] Elmira Ghoulbeigi and Marcus Vinicius dos Santos. Probabilistic developmental program evolution. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1138–1142. ACM, 2010.

[7] Graham Holker and Marcus Vinicius dos Santos. Toward an estimation of distribution algorithm for the evolution of artificial neural networks. In *Proceedings of the Third C* Conference on Computer Science and Software Engineering*, C3S2E '10, pages 17–22. ACM, 2010.

[8] Rafal Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evol. Comput.*, 5:123–141, June 1997.

[9] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[10] John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[11] Julian F. Miller and Peter Thomson. Cartesian Genetic Programming. In *EuroGP*, pages 121–132, 2000.

[12] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. pages 38–46, 1995.

[13] H. Mhlenbein and G. Paa. From recombination of genes to the estimation of distributions i. binary parameters. pages 178–187. Springer-Verlag, 1996.

[14] Georges Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. In *IEEE Transactions on Evolutionary Computation*, pages 523–528, 1998.

[15] Alex A. Freitas. A review of evolutionary algorithms for data mining. In *Soft Computing for Knowledge Discovery and Data Mining*, pages 61–93, 2007.

[16] Shiguo Wang. A comprehensive survey of data mining-based accounting-fraud detection research. In *International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 50–53, 2010.

[17] Yap Bee Wah and I.R. Ibrahim. Using data mining predictive models to classify credit card applicants. In *The 6th International Conference on Advanced Information Management and Service (IMS), 2010*, pages 394–398, 30 Dec. 2010. 2.

[18] K Senthamarai Kannan, P Sailapathi Sekar, M Mohamed Sathik, and P Arumugam. Financial stock market forecast using data mining techniques. *Computer*, I:555–559, 2010.

[19] Tsung-Sheng Chang. A comparative study of artificial neural networks, and decision trees for digital game content stocks price prediction. *Expert Syst. Appl.*, 38(12):14846–14851, November 2011.

[20] Zoran Vojinovic, Vojislav Kecman, and Rainer Seidel. A data mining approach to financial time series modelling and forecasting. *Intelligent Systems in Accounting, Finance & Management*, 10(4):225–239, 2001.

[21] Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar. *Next Generation of Data Mining*. Chapman & Hall/CRC, 2008.

[22] Richard James Povinelli. *Time series data mining: identifying temporal patterns for characterization and prediction of time series events*. PhD thesis, Marquette University, Milwaukee, WI, USA, 1999. AAI9953495.

[23] H. Iba and T. Sasaki. Using genetic programming to predict financial data. In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, pages 244–251 Vol. 1.

[24] Mihai Oltean and C. Grosan. A comparison of several linear genetic programming. *TECHNIQUES, COMPLEX-SYSTEMS*, 14:282–311, 2003.

[25] M.F. Brameier and W. Banzhaf. *Linear Genetic Programming*. Genetic and Evolutionary Computation Series. Springer Science+Business Media, LLC, 2007.

[26] Crina Grosan and Ajith Abraham. Stock market modeling using genetic programming ensembles. *Genetic Systems Programming Theory and Experiences*, 13(2):133–148, 2006.

[27] Janet Clegg, James Alfred Walker, and Julian Frances Miller. A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1580–1587, New York, NY, USA, 2007. ACM.

[28] Jamie MacLennan, ZhaoHui Tang, and Bogdan Crivat. *Data Mining with Microsoft SQL Server 2008*. Wiley Publishing, Inc., 2009.

[29] S. Gustafson, E.K. Burke, and N. Krasnogor. On improving genetic programming for symbolic regression. In *The 2005 IEEE Congress on Evolutionary Computation, 2005.*, 2005.

[30] Anthony Hui. Using genetic programming to perform time-series forecasting of stock prices. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2003*, pages 83–90. Stanford Bookstore, Stanford, California, 94305-3079 USA, 4 December 2003.

[31] Md. Rafiul Hassan, Baikunth Nath, and Michael Kirley. A fusion model of HMM, ANN and GA for stock market forecasting. *Expert Syst. Appl.*, 33(1):171–180, July 2007.

[32] L. DelaOssa, J.A. Gamez, J.L. Mateo, and J.M. Puerta. Avoiding premature convergence in estimation of distribution algorithms. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 455–462, May.

[33] S.K. Smit and A.E. Eiben. Multi-problem parameter tuning using bonesa. In *Artificial Evolution*, pages 222–233, 2011.

[34] Mehnul N. Vora. Genetic algorithm for trading signal generation solution to traders dilemma: Is it right time to trade? In *Economics, Business Innovation*, pages 316–320. IACSIT Press, 2010.

[35] R.S. Tsay. *Analysis of Financial Time Series*. Wiley, 2010.

[36] AndreasS. Karathanasopoulos, KonstantinosA. Theofilatos, PanagiotisM. Leloudas, and SpiridonD. Likothanassis. Modeling the Ase 20 Greek index using artificial neural ner-

works combined with genetic algorithms. In Konstantinos Diamantaras, Wlodek Duch, and LazarosS. Iliadis, editors, *Artificial Neural Networks ICANN 2010*, pages 428–435. Springer Berlin Heidelberg, 2010.

Glossary

Artificial Neural Network (ANN) is a mathematical model inspired by the biological nervous systems. They have a number of inputs and outputs.

Autoregressive Integrated Moving Average (ARIMA) is a class of statistical time series analysis models for forecasting a time series.

bloat is when the program size increases without any changes in its fitness value.

building blocks are simple components for building more complex objects.

Cartesian Genetic Programming (CGP) is a Genetic Programming technique which uses a fixed-length structure to encode the programs. CGP represents a program as a directed graph. See Section 2.1.2.

Compact Genetic Algorithm (cGA) is a form of EDA which maintains the probability vector as in PBIL but unlike PBIL it samples two individuals. See section 2.1.3.

Component Object Model (COM) is a software architecture which allows the re-usability of objects regardless of the programming languages used to implement the objects. A COM object is accessible through a set of interfaces.

Correlation Coefficient (CC) is a measure of the linear dependence between two variables.

crossover is a genetic operator which switches a part of an individual encoding with a part of another individual encoding in the population.

Data Mining Extensions (DMX) is a query language for creating, training, and predicting against data mining models.

Estimation of Distribution Algorithms (EDAs) is a class of methodologies that explicitly induces a probabilistic model of the best candidate solutions. See Section 2.1.3.

Evolutionary Algorithms (EAs) are search methods (i.e. population-based search methods) inspired by natural selection and survival of the fittest in living organisms.

Evolutionary Computation (EC) is a subfield of Machine Learning. EC methodologies are inspired by the biological evolution of living organisms. See Section 2.1.

functional redundancy occurs when a function can be implemented with less nodes.

Genetic Algorithm (GA) is a subfield of Evolutionary Algorithms that evolves a population of individuals by methods inspired by the principles of natural evolution, *e.g.* mutation, crossover, and selection.

Genetic Programming (GP) is an extension of Genetic Algorithms in which the individuals are represented by complex structures. See section 2.1.1.

genotype is the encoding of an individual.

Global Assembly Cache (GAC) is a machine-wide cache which stores the assemblies which have to be shared by several applications on a computer.

Hidden Markov Model (HMM) is a powerful statistical tool for modeling sequences. It can also be used to calculate the probability that a particular sequence was generated by HMM.

input redundancy occurs when one or more of the program inputs are not connected to any nodes.

Knowledge Discovery from Data (KDD) is a field focusing on methodologies to extract knowledge from data. See section 2.2.

local minimum is a minimuml within a neighboring set of solutions.

Machine Learning (ML) is a field which studies how computer programs can automatically improve their performance through experience.

Mean Absolute Error (MAE) is a quantity that measures how close forecast values are to the actual values. See equation 4.4.

Mean Absolute Percentage Error (MAPE) is a quantity that measures the accuracy of a method in constructing fitted time series values.4.5.

mutation is a genetic operator which modifies an individual by changing at least a single gene in its encoding.

node redundancy is the result of unconnected nodes in the CGP graph whose output is not connected to a node leading to the program output.

over-fitting happens when the evolved model performs well on the training data set, but does not perform well on unseen data.

phenotype is the individual's observable characteristics.

Population Based Incremental Learning (PBIL) is the simplest form of EDA and is based on a fixed length binary encoded representation of the individuals. See section 2.1.3.

premature convergence is when programs in the population become similar, but are not the optimal solution to the problem.

primary interop assembly is a unique assembly that contains the definitions of the types defined by COM. It must be signed by the publisher of the COM type.

Probabilistic Incremental Program Evolution (PIPE) is an Estimation of Distribution Algorithms approach in which the structure of the probability distribution model is a program tree. See section 2.1.3.

Probabilistic Prototype Array (PPA) is a fixed length array which stores the probability distribution of promising solutions.

Probabilistic Prototype Tree (PPT) is a complete N-ary tree which stores the probability distribution of promising solutions.

Root Mean Squared Error (RMSE) is a quantity that measures the difference between the forecast values and the actual values. It is defined as the square root of the mean square error.

SQL Server Analysis Services (SSAS) provides the tools to design, create, and manage data mining solutions.

stock market is a public market in which company shares are traded.

Support Vector Machines are supervised learning algorithms that learn by example to classify objects.

symbolic regression tries to find symbolic expressions which fit a given data set.

time series is a series of data points such as stock prices measured in regular time intervals.

Univariate Marginal Distribution Algorithm (UMDA) is a form of EDA in which there is no dependency between the variables of an individual. See section 2.1.3.