Framework for Discovery of Data Models Using Genetic Programming

W.J.L.N Wijayaweera[1], A.S. Karunananda[2]
Faculty of Information Technology, University of Moratuwa
Sri Lanka
[1]njaymax@gmail.com, [2]asoka@itfac.mrt.ac.lk

Abstract—**The field of Genetic Programming in Artificial Intelligence strives to get computers to solve a problem without explicitly coding a solution by a programmer. Genetic Programming is a relatively new technology, which comes under automatic programming. After the initial work by John R. Koza in genetic programming, much research work have been done to discover data models in various datasets. These work have been rather domain specific and little attention has been given to develop generic framework for modeling and experimenting with genetic programming solutions for real world problems. This paper discusses a project to develop a visual environment, named as GPVLab, to design and experiment with genetic programming solutions for real world problems. GPVLab has successfully discovered data models for various data sets and according to the main evaluation it is evident that GPVLab can generate solutions which provide better results in 56% of the time. It is concluded that GPVLab can be used to model genetic programming application very conveniently. GPVLab can be used not only for discovering data models but also doing various experiments in genetic programming.**

Keywords: **Genetic Programming, Artificial Intelligence,Automatic programming.**

1. Introduction

Nature develops everything by the means of natural selection and constant evolution. This is the inspiration behind genetic algorithms that mimics the process of natural evolution [1]. In last few decades of twentieth century the field of genetic programming (GP) has been developed, mostly by John R. Koza, as an extension of genetic algorithms (GA) [2]. GP has repeatedly proved to be a much better approach in solving most symbolic regression problems [3]. Also this field helps to make programs which may automatically generate solutions for a particular problem. In Artificial Intelligence this capability is known as automatic programming which attempts to get a computer to solve a problem without explicitly coding a solution by a programmer. GP is considered as a type of automatic programming which inspired by the Darwinian evolution [4]. It progressively breeds a population of computer programs over a series of generations [5].

A project has been launched to develop a visual environment to design and experiment with genetic programming solutions for real world problems. It is named as GPVLab to mean its ability to facilitate the discovery of data models for real world problems through a wide range of experiments in a visual environment. GPVLab addresses the difficulty of discovering a data model in any numerical dataset and automatically generates an evaluable expression to mimic the discovered data model. It is able to discover data models from any numerical dataset, even with noisy data. It also facilitates to conduct genetic programming experiments. Nevertheless GPVLab is user friendly and it can be used by any person even without the genetic programming knowledge.

GPVLab takes any numerical dataset as the input. Users can initiate the main process by feeding data and selecting the reference column. Then the system runs the genetic programming process by generating populations of expressions and evaluating them to find their fitness. Finally the system determines the best fit model for the dataset. Upon completion of the discovering process, GPVLab immediately allow users to evaluate the model by providing required parameters. It has the facility to save resultant models and access and evaluate them via model library as required. GPVLab has been developed in C# language using Microsoft Visual Studio 2010 [6] and Microsoft .NET Framework 4.0 [7]. AForge.NET 2.2.3 Framework [8] is used as the Genetic Programming toolkit. Furthermore GPVLab model library has been developed using a SQL Server Compact 3.5 [9] database.

In the next section, related work in discovering data models will be discussed. Section 3 discusses the technology behind this project. Section 4 presents genetic programming approach to discover data models. Furthermore section 5 discusses the design of the GPVLab and section 6 discusses the implementation of GPVLab. Section 7 presents the details about evaluation of GPVLab. Finally section 8 present the conclusion and further work related to this project.

2. Related Work in Discovering Data Models

In the field of Artificial Intelligence (AI), machine learning approaches are popular for problems related to regression analysis. Generally most of the researchers use Artificial Neural Network (ANN) and Genetic Algorithm (GA) based solutions for regression analysis. In both occasions an insight of the data is necessary even before proceeding to create a good design. ANN does not promise to converge into a good structure if layers and nodes are not carefully designed. It is hard to determine the optimal value for

number of hidden layers needed for specific problem. However, it is possible that if ANN is used, in order to get good results, one may have to redesign the whole system for different datasets rather than training the same system with new data. This may not provide the best answer for typical real world datasets, which we have gathered through various researches and surveys. Nevertheless there are few attempts been made to develop ANN based systems which can eliminate most of these limitations.

D. F. Specht proposes a new memory-based artificial neural network (ANN) that converges to the underlying (linear or nonlinear) regression surface [10]. The new network is called a general regression neural network (GRNN) which is a one-pass learning algorithm with a highly parallel structure. He argues that the algorithmic form of this solution can be used for any regression problem. The resultant system can be used for prediction, modeling, mapping, and interpolating or as a controller [10]. However; this sort of systems are not suitable for problems where we do not have any prior knowledge on the structure of the resultant model.

GA is not always being a good solution for regression problems because it needs a good fitness function prior to development. Nevertheless several attempts have been made to improve GA based regression analysis systems. S. Tomioka and et al. proposed a new method, adaptive domain method (ADM), to change the domain dynamically using several Genetic Algorithms with short lifetimes [11]. They have applied this to overcome a disadvantage of applying several genetic algorithms (GAs) for nonlinear least square (LS) regressions problems. Furthermore they have demonstrated the improvements in terms of both the convergence and the reliability by ADM. Another advantage of this proposed method is that it does not require any specialized knowledge about GAs or their tuning [11]. Nevertheless these kinds of systems are rather difficult to implement and these are not capable of generating evaluable expressions corresponding to the resulting model.

It is noteworthy that there are some other non-AI approaches to develop such systems as well. Surprisingly some of these systems outperform their AI related counterparts. S. Crino and D. E. Brown presented a procedure for approximating the global optimum in structural design by combining multivariate adaptive regression splines (MARS) with a response surface methodology (RSM) [12]. They argue that even though there are many global optimization routines, few are appropriate for expensive deterministic simulation models such as those used in structural design optimization. According to them the MARS/RSM procedure is a suitable replacement for inefficient techniques such as genetic algorithms (GAs) and simulated annealing (SA). MARS/RSM efficiently and accurately converges to a feasible solution by applying variable screening, domain reduction, and an efficient experimental design [12]. Nevertheless these systems require prior knowledge on the structure of the resultant model.

It is clear that there is a need of a good technology which can automatically identify relationships among attributes in a given dataset. This enables computer systems to discover new models without any intervention of humans. This sort of technology is really important to analyze data in non technical fields like agriculture. That is because the researchers who are working on those areas do not need to have a technical knowledge which is not relevant to their field of research; therefore they can concentrate more on their research rather than tackling technical issues.

In nature everything develops over time through evolution. We can see that nature is using a development approach which grows from the beginning. Nevertheless, we use a compositional approach rather than using developmental procedures [4]. This is rather unnatural, and we still do not know how to grow a complex device like a computer. In the early 20th century, Darwinian theories of evolution formed the idea of development by the means of natural selection and constant evolution. This is the inspiration behind the introduction of genetic algorithms, by John H. Holland, which mimics the process of natural evolution [1]. Until then we were using a compositional approach in every aspects of programming [4]. Nevertheless genetic algorithms do not help to grow a complex structure like a computer program from scratch, which is one of the central goals of computer science, called automatic programming. It is also known as automatic induction of machine code. If the performance of that automatically generated program equals or exceeds the human level, we say it is useful [4]. Nevertheless current methods are not mature enough to develop complete automatic programming systems. According to Arthur Samuel's definition the goal of automatic programming is to "Tell the computer what to do, not how to do it" [13].

In Arthur Samuel's work he mentions the possibility of developing computer programs which will learn and perform better than the person who wrote the program. His work shows that a computer is learning how to play checkers and eventually it performs better than the person who has developed it [13]. If we develop a program which can behave like human being or an animal is doing that particular task, then we can say that process involves learning. Therefore we have to build a program which can learn from available data and evolve. This will eventually eliminate the need of detailed programming effort.

R. Balzer elaborated a simple notion of an automatic compiler into a paradigm for automated software development [14]. He has identified two fundamental flaws in the current software paradigm as the lack of technology for managing the knowledge intensive activities and the maintenance is always performed on source code. He has showed that his proposed paradigm, even without full automation, eliminated those flaws in the current paradigm. It is rather difficult to develop such systems using conventional technologies. There isa need of a new technology which can be easily implemented and capable of producing novel programs to solve significant problems.

Most of the existing paradigms by 1980's involve specialized structures which are nothing like computer programs [2]. When getting computers to solve problems without being explicitly programmed, it should not be required to specify any attributes related to the solution in advance. Instead, these attributes of

the solution should emerge during the problem-solving process. Therefore researchers started to follow the principles of nature instead of the approaches that considered correctness, consistency, justifiability, certainty, orderliness, parsimony, and decisiveness of the solution [2].

Generally if we need to get computers to solve problems without being explicitly programmed, we have to automatically generate programs using segments of basic programs [2]. The first inspiration for automatic generation of simple sequential programs has given by Nichael L. Cramer in 1985 [15]. In his work he stated that even though genetic algorithms have been proved to be useful in many areas, most of these systems focus on adjusting fixed set of parameters in order to optimize the performance of a given algorithm. He clearly explains the possibility of using crossover, selection, mutation, and tree representations to generate programs using genetic algorithms. He shows the possibility of developing simple sequential computer functions, starting with low-level computational primitives, by automatically generating a well-defined two-input, single-output multiplication function [15].

After Cramer's initial work [15], John R. Koza greatly expanded that work to form the modern programming paradigm called genetic programming (GP) [2]. GP is also based on the Darwinian principle of reproduction and survival of the fittest [5]. Even though GP is an extension of Genetic Algorithms, in GP, we evolve a population of computer programs. This automatic problem solving nature of GP does not require the user to know or specify the form or structure of the solution in advance [16]. Hence, GP became the first programming paradigm which is heading towards automatic programming. Since its introduction, GP has been very successful at evolving novel and unexpected ways of solving problems [16].

GP is a form of automatic programming which was inspired by the Darwinian evolution [4]. GP maintains the genetic algorithms' overall algorithm. It progressively breeds a population of computer programs over a series of generations [5]. The dynamic variability of the automatically generating computer programs is an important feature of GP [2]. GP is not a fully automatic programming approach which can generate any computer program [4]. For example it cannot generate a word processer. We use GP as a problem solver, a function approximater, and an effective tool for writing functions to solve specific tasks. It does not replace programmers but it helps them. Programmers should specify the fitness function and identify the problem to which GP should be applied. We should look at GP as another tool for programmers [4]. Over time GP proved its capability of producing human-competitive results [4]. It has produced human-competitive results in areas such as control, design, classification, pattern recognition, game playing, and algorithm design. This became possible with use of the LISP programming language which supports the processes in genetic programming. The symbolic expressions (S-expressions) of the LISP are an especially convenient way to create and manipulate the compositions of functions and terminals (input to the computer program to be discovered) [17].

GP produces really good results in the areas where the interrelationships among the relevant variables are poorly understood and areas where there is a large amount of data in computer readable form that requires examination, classification, and integration [2]. GP is also good at unveiling unexpected relationships among variables and producing analytical solutions where conventional mathematical analysis cannot be used [16]. Applications of GP range from small subroutines to real-time problems and offline applications such as time-series analysis or data-mining tasks [16]. GP techniques have been used for the evolution of structures other than computer programs, such as neural networks and analog electrical circuits [5]. GP has produced human competitive results in areas such as control, design, classification, pattern recognition, game playing, and algorithm design [4]. GP is beginning to find interesting new programs that humans had not previously discovered and it is becoming more and more popular in solving symbolic regression problems.

D. A. Augusto and H. J. C. Barbosa presented implementation for solving symbolic regression problems using genetic programming (GP) [3]. Since the standard implementations of GP in compiled languages are not much efficient, they have presented a new approach using Read's linear code. They have presented observational data to prove that this approach lead to more simplicity and better performance compared to traditional GP implementations.

H. Tuan-Hao and et al. have presented some experimental results using Incremental Evaluation with Tree Adjoining Grammar Guided Genetic Programming (DEVTAG) [18]. They have used symbolic regression problems such as Fourier series problem (sawtooth problem) and a benchmark polynomial fitting problem in genetic programming. In their study they have presented comparison results to show that the results of their approach outperformed both standard Genetic Programming (GP) and the original Tree Adjoining Grammar Guided Genetic Programming (TAG3P). However they have not directed their study towards a general system which can solve any symbolic regression problem.

A similar work has been conducted by G. Dworman and et al. to discover high-quality negotiation policies using genetic programming [19]. They have presented series of successful experimental results. Nevertheless this study has not directed towards developing a system which can solve symbolic regression problems by working with any input dataset.

We have encountered two similar products related to GP, namely GPdotNet [20] and GPLab [21]. GPdotNet is free software which was developed using Microsoft .NET technologies. This software is capable of handling large datasets and this supports multi core parallel processing as well. Nevertheless this software is not user friendly and it lacks the noise reduction capabilities. GPLab is a relatively old toolbox for the MATLAB [22] or Octave [23]. GPLab is an advanced tool for researchers who are doing advanced experiments using genetic programming. Since this is not a standalone application but a toolbox

for MATLAB, the users should have the knowledge on MATLAB or Octave as well. This toolbox also does not provide noise reduction as a feature.

It is evident that enough work has been done to discover models from datasets using genetic programming. However, there is little or no work has been done towards developing a general solution which can discover data models even from noisy numerical dataset. Such a user friendly general program is an ideal solution for non-technical domains. Considering all these facts we can see that that there's huge potential that we can use GP to unveil the relationship among attributes in any numerical dataset. Therefore if we can develop a software which can take datasets as inputs and produces computer programs to represent the data model in the input dataset, it makes it possible to discover previously unknown data models as well as readily predict the outcomes of future events using the automatically generated computer program.

3. Technology Adapted

Genetic programming (GP) is a domain independent problem solving approach in which computer programs are evolved to solve, or approximately solve, problems. In each generation, GP stochastically transforms populations of programs into new, optimistically better, populations of programs. GP is a random process, and it can never guarantee results. Nevertheless GP's randomness can lead it to go beyond the capabilities of existing paradigms and solve problems in unexpected ways.

GP finds out the performance of a program by running it, and then comparing its behavior to some ideal situation. That means GP can search through the set of possible computer programs to find an individual computer program which has the highest fitness in solving, or approximately solving, the problem at hand. The fitness measure is the most important fact which produces the needed program structure. The final computer program structure that emerges from GP process is a result of this fitness.

In GP, user must perform five major preparatory steps before applying GP to a problem. First step is to determining the set of terminals. The terminals are the inputs to the resultant computer program. Secondly the user should determine the set of primitive functions. These are the set of functions that are to be used to generate the mathematical expression that attempts to fit the given sample of data. Third step is to determine the fitness measure which decides how well each individual computer program, in the population, performs in the particular problem environment. Each individual computer program in the population is executed and then evaluated using this fitness measure. The fourth step is to determine the parameters which control the execution of the GP process. The primary parameters for controlling a run of GP are the population size and the maximum number of generations to be run. The fifth and last step is to determine the method for designating a result and the termination criteria. One frequently used method of result designation is to designate the best individual obtained in any generation of the population as the result for that generation. Termination criteria can be taken as the event of exceeding the maximum number of generations.

In main GP process there are three major steps. First step is to generate an initial population of random compositions of the functions and terminals of the problem. Secondly iteratively perform two sub steps until the termination criterion has been satisfied.

1) Execute each program in the population and assign it a fitness value.

2) Create a new population of computer programs by applying genetic operations. The genetic operations are applied on selected computer programs from the population. Selection process is based on fitness of the each program.

There are three main genetic operations. First operation is the Darwinian reproduction which is also known as cloning. This operation reproduces an existing program by copying it to the new population. Second operation is crossover which creates two new computer programs from two existing programs by genetically recombining randomly chosen parts of two existing programs. Mutation, which is the third operation, creates one new computer program from an existing program by mutating a randomly chosen part of the program. The third and final step is to output the best-so-far individual program as the result. This result may be a solution, or approximately a solution, to the problem.

4. A Genetic Programming Based Approach to Discover Data Models

It is apparent that GP allows us to discover data models and automatically generate evaluable expressions to evaluate those models using any given dataset. It is possible to use any numerical data set as an input to this software system. This system should allow the users to load data using files in a convenient file format such as comma separated value (.csv). It also should facilitate the users to manually enter required data. Furthermore the user should select the reference column before starting the discovering process. If user has not selected any specific column, the system should take the last column on the dataset as the reference column. Furthermore advanced users may change parameters related to the genetic programming process. These parameters may include control parameters related to reduce noise in data as well.

When user initiates the process by entering the data set and selecting the reference column, the system will run the genetic programming process by renaming the attribute names. The system will randomly generate populations of expressions and evaluate them to find their fitness. After going through generations after generations, the system will determine the best fit model for the dataset. If the maximum number of generations exceeded the system will stop the process and output the best so far model. Once the main process is completed, the system should output the discovered model as an expression. This expression can be evaluated as needed by providing values for required parameters. Furthermore users should have the facility to save the expression. Once saved, this expression can be evaluated at any time by entering values for required parameters. Furthermore the user

can export the data model into a more portable form which can be evaluated the model independently.

User for this system can be any person who needs to discover a model out of a collected numerical data set. Advanced users who have knowledge about Genetic Algorithms of Genetic Programming can try changing advanced parameters related to the genetic programming and noise reduction for better results. Nevertheless the default settings will work for almost all the problems. Knowledge about Genetic Programming is not a necessity.

This software should have the features such as, the ability to create a dataset through the system and directly use an existing .csv file to load a dataset. This software also provides advanced Genetic Programming settings for advanced users. It has a feature to save output expressions and evaluate saved expressions at any time by providing values for required parameters. This system allows users to export expressions as well. This feature should allow users to export saved models to independent and portable executables.

5. Design of the Genetic Programming Based Software Solution

This software solution has been named as GPVLab to mean its ability to facilitate the discovery of data models for real world problems through a wide range of experiments. In order to build a user friendly system, user should have a proper way to interact with the system and enter data into system. Once the data is available the main module of GPVLab should have the facility to perform genetic programming process using available data. Once the process is completed GPVLab should output the resultant expression. GPVLab should have the facility to evaluate the resultant expression by entering values for required parameters. Also GPVLab should facilitate to save and export resultant expressions for evaluate them as needed. Fig. 1 shows a detailed design diagram of this software solution.

In the initialization stage system should gather all the relevant information to perform the model discovery process. In this stage GPVLab allow the user to either create a dataset from the beginning or load data from an existing file. The file should be in a convenient format for both user and the system, such as .csv. Furthermore GPVLab should allow users to specify the reference column. If the reference column is not specified the system should automatically select the last column as the reference column. GPVLab should facilitate the users to specify values for most important parameters which are directly involved in Genetic Programming process. However it should specify default values for all these parameters in an optimal manner to make the system usable for average users.

Once the required data and other additional parameters have been set, GPVLab should begin the genetic programming process by generating an initial population. This population consists of randomly generated evaluable expressions. Each evaluable expression is a valid mathematical expression which contains operators and operands. Once the user starts the main process, column names of the input dataset (except the reference column name) will be taken as the terminals. Set of constants along with these column names will create the set of possible operands. Then the system will take arbitrary number of operands (from the operand set) and randomly selected operators to make valid expressions. After a validation process, this newly created evaluable expression will be put into the population. Once the initial population is generated, it should perform the selection of the best fit expression by executing each of them and evaluating the fitness measure of those expressions. Then it should populate the next generation by performing genetic operations (cloning, crossover and mutation) over selected expressions. This process will iteratively be continued until the termination criterion has been satisfied. Finally GPVLab should give the exact expression or the best so far expression as the solution.

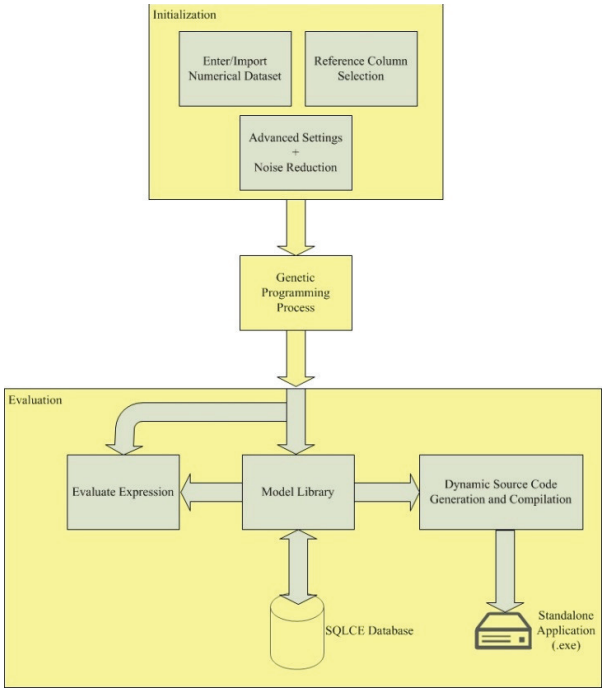

Figure1: Detailed design diagram of this software solution

As soon as the discovery process is completed, users should have the facility to evaluate this expression through the system. GPVLab should also facilitate the users to save the expressions for future reference. Inbuilt facility should be provided to browse and evaluate saved expressions. At the time of execution of the resultant expression, system should prompt the users to enter values for required parameters. Once the required parameters have been assigned with user input values, the system should execute the expression and the result of the evaluated expression should be present to the user. Furthermore GPVLab should have the facility to export saved data model. This facility can be provided by dynamically generating a code and compiling it to form a standalone executable. This feature is important when user need a portable simple application that does not depend on the main software system. Therefore user

should have the ability to generate portable standalone applications from saved data model expressions.

6. Implementation of GPVLab

GPVLab has been developed using C# language with .NET framework 4.0. The AForge.NET library has been used as the main framework for genetic programming. GPVLab also extends some classes of AForge.NETframework to accommodate data with arbitrary number of attributes and to remove noise in data. GPVLab has been implemented as a MultipleDocument Interface (MDI) application. Inside the GPVLab main window, users can open multiple instances of explorer windows. Each explorer window has the facility to independently conduct experiments using numerical datasets. This feature is very important in performing parallel experiments. Fig. 2 shows a screenshot of an explorer window. GPVLab's Explorer window takes any numerical dataset as the input. There are two possible ways to feed datasets into the system. Inside the explorer window, either user can enter a required data using the inbuilt facility or import a dataset from existing comma separated value (*.csv) file. These options are available inside the "Data Input Methods" section of the explorer window. If user selects to manually enter data, the system prompts the user to enter the number of columns for the dataset. Then GPVLab will generate a dataset with specified number of columns with automatically generated column names. Once the dataset is entered or imported into GPVLab, user should select the reference column from the given drop down list. By default GPVLab automatically selects the last column as the reference column.

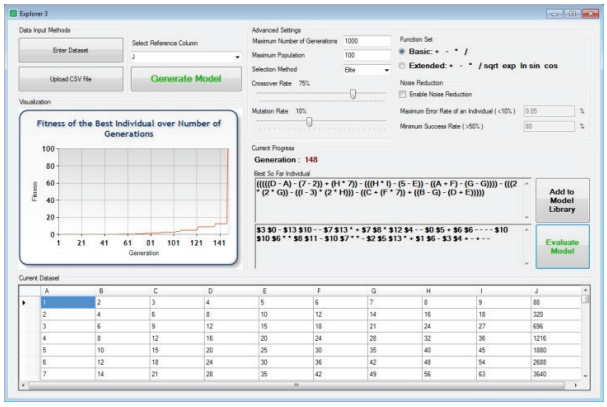

Figure2: A screenshot of an explorer window

Furthermore an advanced settings section is provided for advanced users, where control parameters of the genetic programming process can be adjusted. This section includes settings related to the noise reduction feature as well. Advance settings section contains six major parameter settings which directly involves in the genetic programming process. First the user can define the maximum number of generations to be run. This is one of the termination criteria of the geneticprogramming process. Next, users can adjust maximum number of population. Then user can choose the selection method. It contains three selection methods, namely "Elite", "Rank" and "Roulette Wheel". The system also allows the user to select the mathematical function set to be used in the genetic programming process. The system has an option to select one out of two function sets for this purpose, namely basic and extended. Basic function set contains operators such as addition, substations, multiplication and division. Extended faction set has the ability to generate models consisting of sqrt (Square root), sin, cos, ln and exp (exponential) in addition to the basic operators. Finally the advanced settings section facilitates the user to adjust the crossover and mutation rates.

Furthermore if the data set contains noisy data, users have the facility to enable inbuilt noise reduction feature. Once this is enabled the system allows users to change two additional parameters related to the noise reduction. Those are the maximum error rate and the minimum success rate. If the noise reduction is enabled,those two parameters also affect the genetic programming process. Once the initialization phase is over, the user can start the model discovery process by clicking on the "Generate Model" button. This starts the genetic programming process. Genetic programming process generates the initial population by randomly creating expressions consist of operators from the selected function set, terminals and constant set. Terminals will be the column names other than the reference column. The constant set contains values such as 1, 2, 3, 5 and 7. Fig. 3 gives some important code segments involved in main genetic programming process. Once this process is started GPVLab displays the current generation and the real time visualization of the change in fitness of the best individual for each generation.

Once the genetic programming process starts; it will continuously breed evaluable expressions. Each of these individuals is evaluated for its fitness. First of all, an individual is validated to see whether it consists of all the required terminals. Once an individual passes the validation check, that particular individual is evaluated for the entire dataset to get the total error. This is performed by evaluating the individual for each record in the dataset. Error of an individual is determined by calculating the absolute difference between the result it gives for a particular record and the corresponding reference column value. If the noise reduction is enabled the error is calculated in a different way. If the error of the individual is in the acceptable range, which is defined as maximum error of an individual, then this individual is counted as a success for that particular record in the dataset. This is performed for the entire dataset and calculated for the number of successful encounters within the dataset. Furthermore if the noise reduction is enabled, before returning the fitness value, a method checks the eligibility of that particular individual. If the individual is eligible, which means it achieved the minimum success rate, and then this method adds this expression to the list of successful individuals.

The fitness function uses in the genetic programming process is "100 / (1 - sum of absolute errors)". If the error is zero then this makes the fitness value of 100 for that particular expression, which means a perfect solution. If the system has found a perfect solution, the genetic programming process will immediately end and resultant expression is presented to the user. If a perfect solution has not been discovered within the given number of generations ,

then the GPVLab outputs the best so far expression. Either way the system outputs the best so far evaluable expression in reverse polish notation (RPN). If the resultant expression contains only basic operators, GPVLab automatically converts the resultant expression in RPN notation into a human readable expression in infix notation. Furthermore if the noise reduction feature is enabled, the expression with the lowest operator count, out of eligible expressions, will be considered as the best solution. Nevertheless GPVLab may stop the process and output the solution if the genetic programming process founds a 100% successful solution for the particular dataset.

Upon completion of the discovering process, users can immediately evaluate the model by providing required parameters. Also GPVLab has the facility to save resultant models and access and evaluate them via model library as required. Model library is developed using SQL compact edition database, which does not require SQL Server instance to operate. Hence, GPVLab is highly independent and can be installed or run on any computer with just .NET Framework 4.0 installed. Fig. 4 shows a screenshot of the model library. Furthermore the model library also provides a facility to export saved expressions as standalone executables (.exe). This has been achieved using the Microsoft CodeDom technology. This mechanism generates a highly portable executable, which can be executed as needed on any computer with just .NET Framework 4.0 installed.

7. Evaluation of GPVLab

The main experiment has been done using a real world noisy dataset. The data was taken from the World Bank Data Catalog [24]. This repository contains data about each country in the world. For this experiment author has taken the data sheet under the "World Development Indicators" section in Sri Lanka. This data sheet contains various indicators about the country's development. For the purpose of this experiment author has taken eight important data columns. These are "Total reserves (includes gold, current US$)" (TR), "Inflation from consumer prices (annual %)" (I.CP), "General government final consumption expenditure (% of GDP)" (GGFCE), "Exports of goods and services (current US$)" (EOG.S), "Gross domestic product (current US$)" (GDP), "Official exchange rate (LCU per US$, period average)" (OER), "Life expectancy at birth" (LEABT) and "Total population" (PT). Apparently there is no relationship between these data columns. Out of all the data author has taken the fifty records from year 1960 to 2010 for this experiment.

The purpose of this experiment is to generate an expression to find the GDP. Further this experiment compares the result of GPVLab with the result of a conventional data modeling mechanism. For this purpose author has selected the Linear Regression functionality in WEKA (Waikato Environment for Knowledge Analysis) [25]. First author imported the dataset into GPVLab. Then the author has selected the "Gross domestic product (current US$)" (GDP) as the reference column. After selecting the 'GDP' as the reference column, without changing default settings, the genetic programming process has been initiated by clicking on the "Generate Model" button.

```
The main genetic programming method needs the input dataset as
a multidimensional double array. Therefore the following code
segment is used to convert the data in the data grid into a
multidimensional double array.
double[,] InputDataset =
CommonUtilities.DataGridViewToDoubleArray(dgvInputData);

Following code segment initializes the fitness function.
ExtendedSymbolicRegressionFitness Fitness = new
ExtendedSymbolicRegressionFitness(InputDataSet,
InputConstants);

Following code segment is used to initialize the population.
IGPGene Gene = (IGPGene)new
SimpleGeneFunction(NumberOfConstants + ArgumentsCount);
if (FunctionSet == 1) {
  Gene = (IGPGene)new
ExtendedGeneFunction(NumberOfConstants +
ArgumentsCount);}

Creating the Chromosome
IChromosome GPChromosome = (IChromosome)new
GPTreeChromosome(Gene);

Initializing the selection method
ISelectionMethod GPSelectionMethod = (ISelectionMethod)new
EliteSelection();
if (SelectionMethod == 1){
  GPSelectionMethod = (ISelectionMethod) new RankSelection(
); }
else if (SelectionMethod == 2){
  GPSelectionMethod = (ISelectionMethod) new
RouletteWheelSelection( );}

Initializing the Population
Population GPPopulation = new Population(MaxPopulation,
GPChromosome, Fitness, GPSelectionMethod);

Setting Crossover Rate
GPPopulation.CrossoverRate = CrossoverRate;

Setting Mutation Rate
GPPopulation.MutationRate = MutationRate;

Once the population is initialized following code segment is used
for run the genetic programming process for a single generation.
GPPopulation.RunEpoch();

Finally we can access the best so far individual using following
code segment.
BestSoFarIndividual =
GPPopulation.BestChromosome.ToString();
```

Figure3: Important code segments of the main method related to the genetic programming process.

At the beginning GPVLab couldn't find a good solution with normal settings. After several experiments GPVLab has found a solution within 132 generations. GPVLab has achieved this after setting maximum number of generations to 10000, maximum population to 500 and maximum error rate of an individual to 25. Please note that extended function set was selected and noise reduction was enabled. All other settings were set to their respective default values. GPVLab has identified the following expression (RPN) as the solution.

$3 $3 + $5 $4 + + $4 $5 + $1 $9 - / + $3 $4 + $0 $2 + $7 $0 / + + + $1 $4 + $5 $4 - / $3 $6 sin - + $4 $9 sqrt - $9 $1 + sqrt - / +

Note that "$0, $1, $2, $3, $4, $5, $6, $7, $8 and $9" represents 'TR', 'I.CP', 'GGFCE', 'EOG.S', 'OER', 'LEABT', 'PT', 1, 2, 3 respectively. After replacing the real values the RPN expression reads as follows.

EOG.S EOG.S + LEABT OER + + OER LEABT + I.CP 3 - / + EOG.S OER + TR GGFCE + 1 TR / + + + I.CP OER + LEABT OER - / EOG.S PT sin - + OER 3 sqrt - 3 I.CP + sqrt - / +

Then author imported the data into WEKA. After starting WEKA, author selected the Explorer option. Inside the explorer window there is an option to open files. Author selected the same .csv file using this option. Then on the 'Classify' tab author selected the 'LinerRegression', under functions category, as the classifier. Then under 'test options' author selected 'Use training set'. Finally author selected the '(Num) GDP' as the output attribute. Then author started the process by clicking on the 'Start' button. Just after 0.04 seconds WEKA found a solution. WEKA has identified the following as the solution.

GDP = (2.6561 * TR) + (73187869.1757 * I.CP) +
(1083693823.1583 * GGFCE) + (2.8882 * EOG.S) +
(-128547505.6347 * OER) + (966158549.8081 * LEABT) +
(-557.9261 * PT) + (- 65820373354.9022)

After generating models from both systems, user has evaluated these models for each record of that dataset. According to the results the error rate of the solution generated by WEKA falls between -93.74% and 52.00%. Nevertheless the error rate of the solution generated by GPVLab falls between -24.15% and 24.51%. Furthermore in 28 occasions out of 50 records, the expression generated by GPVLab gives more accurate answers than the expression generated by WEKA. That means 56% of the time the expression generated by GPVLab gives better results than the expression generated by WEKA.

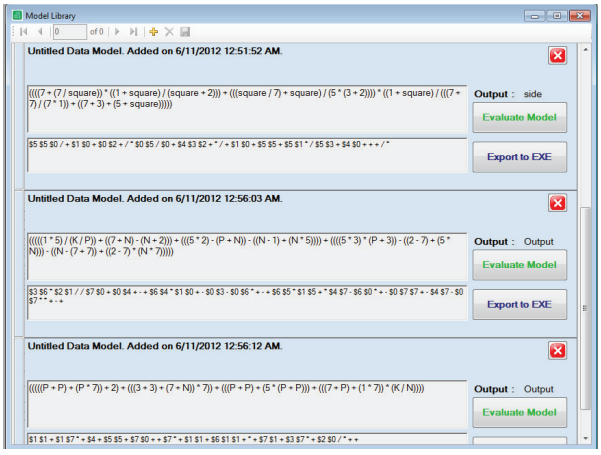

Figure4: –A screenshot of the model library

Furthermore GPVLab has successfully discovered data models in simple known data sets including addition of three numbers, square root of a number and dataset with ten columns with a known data model. All these experiments were conducted by importing the first 80% of the respective dataset into GPVLab. Once the model generation process completed, the resultant model was evaluated for the remaining 20% of the dataset to validate the model. GPVLab has found perfect solutions for the dataset related to addition of three numbers in just 11 generations. The square root of a number experiment has been performed for 5 times with default settings. However, GPVLab has failed to find a perfect solution in all 5 occasions. Then the same experiment has been conducted by changing the function set to "Extended". This time the GPVLab directly provided the answer using sqrt (square root) function within the first generation. Another experiment has been conducted using a sample dataset with 10 columns. GPVLab has successfully found a perfect in 148 generations. Nevertheless the resultant expression is different than the expected data model.

Another experiment has been done using a noisy real world datasetas well. For this experiment authors have taken "Life expectancy at birth, total (in years)" (LE) and "Official exchange rate (LCU per US$, period average)" (ER) columns from the same world bank dataset. Then authors have imported first 60% of the dataset into GPVLab. Then we have performed series of experiments by changing the function set, maximum number of generations, maximum population and mutation rate. None of these experiments gave a good data model. Therefore we used the noise reduction feature of the GPVLab to find a model for this data set. We were able to find a good data mode after a series of experiments conducted by adjusting the maximum error of an individual. After evaluating the resultant expression for the remaining 40% of the dataset, we saw that this model can approximately predict an answer for a given input. Even though this resultant expression is not a perfect data model, we have found that the margin of error for this resultant expression falls within the range of -2.235 and 3.829.

Furthermore we have conducted various experiments by changing advance settings as well. First, each experiment was conducted using default settings. After recording observations, we performed the same experiment by changing advanced settings. Then we compared the results by the means of accuracy and the number of generations it has taken to find a solution. Furthermore we have given this software to three users who had no knowledge about genetic algorithms or genetic programming. They have successfully adapted to the system and found the system very usable for discovering data models. All three users were happy about the overall system and its capabilities.

8. Conclusion and Further Work

Major aim of this project was to develop a visual environment to design and experiment with genetic programming solutions. This aim has been achieved by developing the GPVLab. According to the main evaluation it is evident that solutions generated through GPVLab provide better results in 56% of the time, compared to the solution generated through conventional linear regression with multiple variables.

Furthermore according to the results obtained through experiments such as addition of three numbers, square root of a number and the experiment with ten column dataset, GPVLab has proved its capability to handle different type of data sets. More importantly the square root of a number experiment showed the importance of 'Advanced Settings' section and the "Extended" function set. The experiment with the noisy real world dataset proved that GPVLab is capable of handling noisy real world datasets. After enabling noise reduction and tweaking noise reduction parameters, system was able to find relatively simple and accurate solutions for noisy data sets. Experiments with advanced settings showed that GPVLab can be used as an advanced tool to experiment with genetic programming problems. After comparing observations collected through numerous experiments, we found that when increasing the maximum population, the system increases the chance of finding a solution within considerably lower number of generations. After going through all the evaluation results we saw that GPVLab has provided some unexpected solutions as well. This proved that GPVLab may lead us to discover a new theory which was previously unknown. At the beginning of the development one of the main objectives is to develop a solution which can be used by non technical persons. For an example a researcher from a non technical domain like agriculture can benefit from this system even without having any knowledge about genetic programming. The results obtained through users with no knowledge about genetic algorithms or genetic programming, proved that this can be a really good tool for the researches in various fields. Considering all these facts it is concluded that GPVLab can be used to model genetic programming application very conveniently.

However, GPVLab is not optimized for large data sets; further work includes optimizing GPVLab to work with large scale datasets. This system may also be extended to take any dataset as an input by encoding non numeric data into numeric representations.

References

[1] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence", 2nd Edition, The MIT Press, 1992.

[2] J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection", Cambridge, MA: The MIT Press, 1992.

[3] D. A. Augusto and H. J. C. Barbosa, "Symbolic Regression via Genetic Programming", *Proc.of the 6th Brazilian Symposium on Neural Networks (SBRN'00)*, 2000, pp. 173 – 178.

[4] W. Banzhaf, J.R. Koza, C. Ryan, L. Spector and C. Jacob, "Genetic programming", *IEEE Intelligent Systems and their Applications*, vol. 15, no. 3, 2000, pp. 74 – 84.

[5] J. R. Koza, "Darwinian invention and problem solving by means of genetic programming", *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC '99)*, vol. 3, 1999, pp. 604 – 609.

[6] Microsoft Visual Studio 2010, http://www.microsoft.com/visualstudio/en-us, 2012.

[7] Microsoft .NET Framework, http://www.microsoft.com/net, 2012.

[8] AForge.NET Framework, http://www.aforgenet.com/, 2012.

[9] Microsoft SQL Server Compact, http://msdn.microsoft.com/en-us/data/ff687142.aspx, 2012.

[10] D. F. Specht, "A General Regression Neural Network", *IEEE Transactions on Neural Networks,* vol. 2, no. 6, 1991, pp. 568 – 576.

[11] S. Tomioka, S. Nisiyama, and T. Enoto, "Nonlinear Least Square Regression by Adaptive Domain Method with Multiple Genetic Algorithms", *IEEE Transactions on Evolutionary Computation,* vol. 11, no. 1, 2007, pp. 1 – 16.

[12] S. Crino and D.E. Brown, "Global Optimization with Multivariate Adaptive Regression Splines", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,* vol. 37, no. 2, 2007, pp. 333 – 340.

[13] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, vol. 3, no. 3, 1959, pp. 210 – 229.

[14] R. Balzer, "A 15 Year Perspective on Automatic Programming", *IEEE Transactions on Software Engineering,* vol. SE-11, no. 11, 1985, pp. 1257 – 1268.

[15] N. L. Cramer, "A Representation for the Adaptive Generation of Simple Sequential Programs", *Proc. of an International Conference on Genetic Algorithms and Their Applications (ICGA85)*, 1985, pp. 183-187.

[16] R. Poli, W. B. Langdon and N. F. McPhee, "A Field Guide to Genetic Programming", 2008.

[17] J. R. Koza, "Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence", *Proc. of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, 1990, pp. 819 – 827.

[18] H. Tuan-Hao, R. I. McKay, D. Essam, and N. X. Hoai, "Solving Symbolic Regression Problems Using Incremental Evaluation in Genetic Programming", *IEEE Congress on Evolutionary Computation (CEC 2006)*, 2006, pp. 2134 – 2141.

[19] G.Dworman, S.O.Kimbrough, and J.D.Laing, "On Automated Discovery of Models Using Genetic Programming in Game-Theoretic Contexts", *Proc. of the 28th Hawaii International Conference on System Sciences*, vol. 3, 1995, pp. 428 – 438.

[20] GPdotNET, Genetic Programming Tool, http://gpdotnet.codeplex.com/, 2012.

[21] GPLAB, A Genetic Programming Toolbox for MATLAB, http://gplab.sourceforge.net/, 2012.

[22] MathWorks MATLAB, http://www.mathworks.in/products/matlab/, 2012.

[23] GNU Octave, http://www.gnu.org/software/octave/, 2012.

[24] World Bank Data Catalog, http://data.worldbank.org/, 2012.

[25] WEKA (Waikato Environment for Knowledge Analysis), Version 3.6.4, http://www.cs.waikato.ac.nz/~ml/weka/, 2012