KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY, KUMASI

A MULTIGENE GENETIC PROGRAMMING MODEL FOR THYROID DISORDER DETECTION

OHENEBA-OSEI NYAME FIDELIS , BSc.

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF MATHEMATICS, KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY, KUMASI, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Philosophy in Applied Mathematics (Mphil.)

OCTOBER, 2015

Declaration

I hereby declare that this submission is my own work towards the award of the M. Phil degree and that, to the best of my knowledge, it contains no material previously published by another person nor material which had been accepted for the award of any other degree of the university, except where due acknowledgement had been made in the text.

Oheneba-Osei Nyame Fidelis	.....	.....
Student	Signature	Date

Certified by:

Dr. J. Ackora-Prah	.....	.....
Supervisor	Signature	Date

Certified by:

Prof. S.K. Amponsah	.....	.....
Head of Department	Signature	Date

Dedication

I dedicate this work to my wonderful and lovely family and friends and all those who contributed towards my work.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Dr. J. Ackora-Prah for the continuous support of my MPhil study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my MPhil study.

Besides my Supervisor, I would like to thank the rest of the Lecturers in Mathematics department who impacted my thesis and life for their encouragement, insightful comments, and hard to answer questions.

My sincere thanks also goes to Joseph Mensah, Appati Justice Kwame, Lamptey Parker and Andam S. Perpetual

I thank my fellow Mphil. mates in Applied Mathematics for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last few years. Also I thank my friends Yeboah Emmanuel, Nii Armatey and Jane for motivating and enlightening my research with a foreign experience.

In particular, I would like to thank my family, especially my parents, Mr. Simon Nyame and Mrs. Georgina Nyame, for supporting me spiritually and financially throughout my life.

Most of all, I would like to thank the Almighty God for seeing me this far.

Abstract

Two common diseases of the thyroid gland, which releases thyroid hormones for regulating the rate of the bodys metabolism, are hyperthyroidism and hypothyroidism. The classification of these thyroid diseases is one of the considerable tasks. Before a patient is classified as being normal (thyroid gland is functioning well) or suffering from hyperthyroidism or hypothyroidism, there are a lot of information and tests conducted on the patient by existing models and these are costly in terms of time and money. This research is conducted to enhance the detection of thyroid disorder based on attributes collected from patients. A mathematical model is developed using Multigene Symbolic Regression Genetic Programming technique. A statistical test is conducted to ascertain the goodness of fit of the model. The test result showed that the model is good and is even able to reduce the number of attributes used to classify a patient as Normal, Hyperthyroidism and Hypothyroidism.

Table of Contents

Page

List of Tables

List of Figures

Chapter 1

This chapter gives a description of the background of the study and in effect provides the reason or purpose of the study, that is, the problem statement. It also gives the objective of the study and provides a brief methodology to the study. Finally, this chapter ends with the justification and the organisation of the study.

1.1 Background of the study

The thyroid is a small gland found at the base of the neck, just below the Adams apple (laryngeal prominence). The thyroid produces two main hormones called triiodothyronine (T3) and thyroxine (T4). These hormones travel in the blood to all parts of the body. The thyroid hormones control the rate of many activities in the body. These include how fast the body burns calories and how fast the heart beats. All of these activities are known as the body's metabolism. A well functioning thyroid produces the right amount of hormones needed to keep the body's metabolism working at a normal rate (not too fast or too slow). Thyroid disorders can range from a small, harmless goiter (enlarged gland) that needs no treatment to life-threatening cancer. The most common thyroid problems involve abnormal production of thyroid hormones. Over-production of thyroid hormone results in a condition known as hyperthyroidism. Insufficient hormone production leads to hypothyroidism. The

ability to classify such an abnormality for proper diagnoses and treatment is crucial.

From Kulkarni et al,(2012) Hypothyroidism, or an under-active thyroid, has many causes. Some of the causes are prior thyroid surgery, exposure to ionizing radiation, chronic inflammation of the thyroid (autoimmune thyroiditis), iodine deficiency, lack of enzymes to make thyroid hormone, and various kinds of medication. Hyperthyroidism, or an overactive thyroid, may also be caused by inflammation of the thyroid, various kinds of medications, and lack of control of thyroid hormone production.The seriousness of thyroid disorders should not be underestimated as thyroid storm (an episode of severe hyperthyroidism) and myxedema coma (the end stage of untreated hypothyroidism) may lead to death in a significant number of cases.

Of late, the use of smart methods in control systems, signal processing and sample recognition is a powerful tool in various scientific, technical and engineering, medicine, and medical engineering researches has been greatly emphasized. An example of the use of such systems in different medical and medical engineering fields could be the following: research on various diseases, simulation of various body organs, and simulation of body metabolisms. In this regard, the research on thyroid disease and its diagnosis by means of genetic programming has been emphasized by researchers.

In artificial intelligence, genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user defined task. Essentially GP is a set of instructions and a fitness function to measure how well a computer has performed a task. It is a specialization of genetic algorithms (GA)

where each individual is a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task.

1.1.1 Problem statement

Thyroid is a small gland found at the base of the neck, just below the Adams apple and produces two main hormones called T3 (triiodothyronine) and T4 (tetraiodothyronine). The hormones control the rate of many activities in the body. The most common thyroid problems involve abnormal production of thyroid hormones (hyperthyroidism), whilst insufficient hormone production leads to hypothyroidism.

Before a patient is classified as being normal (i.e the thyroid gland is functioning well) or suffering from hyperthyroidism or hypothyroidism, a physician has to analyze a lot of factors based on laboratory and clinical test (disease symptoms), in addition to his experience before diagnosing the thyroid problem (Thyroid low efficiency and high efficiency) which makes physicians job very difficult. Also there would be demand for a large number of physicians when everybody at risk will need to go through all these test procedures. In short, these are costly in terms of time and money. Hence we develop a mathematical GP model to curb this situation by reducing the information and the test conducted on the patient.

1.1.2 Objectives of the study

In order to address the above mentioned problem, the objective is to develop a mathematical model using GP approach to reduce the attributes used to classify patients as:

- Normal (1)
- Hyperthyroidism (2)
- Hypothyroidism (3)

1.1.3 Methodology

Genetic programming is an evolutionary algorithm based methodology inspired by biological evolution to find computer programs that perform a user defined task. When the task is building an empirical mathematical model of data acquired from a process or system, then GP is said to be performing symbolic regression, Searson,(2009). Symbolic Regression is a type of regression analysis that searches the space of mathematical expressions to find the model that best fit a given data set, both in terms of accuracy and simplicity. Unlike the normal Regression Analysis, here no particular model is provided as the starting point to the algorithm and no prior assumptions are imposed on the algorithm. Both the model structures and parameters are determined automatically and hence has lager search space to search and GP is very good at this area of application.

The data used in this thesis was retrieved from the UCI Machine learning Repository. The data has 7200 instances and consists of twenty one (21) input variables and one(1) output

(1,2,3). The data was divided into training set and testing set.

In our methodology we shall propose genetic programming approach in solving the problem at hand. First, the algorithm will be presented. Then a real life computational study will be performed on the secondary data.

The developed application based on GP uses the basic executional steps as follows to solve the thyroid disorder:

1. Input data file containing m x n matrix of input (predictor) variables, an m x 1 matrix of output (response) variables, and system configuration file is fed to the GP system.

2. Generate a random initial population of expressions based on Step1 using a multigene model.

3. Assign a fitness value to each individual in the population (RMSE).

4. Create a new population of model expressions.

5. Create new solutions by crossover and mutation.

6. The best solution is chosen as the fittest solution (least error margin)

Matlab will be the software package that will be employed to analyze our data.

1.1.4 Justification

Two common diseases of the thyroid gland, which releases thyroid hormones for regulating the rate of body's metabolism, are hyperthyroidism and hypothyroidism. The classification

of these thyroid diseases is a very important task.

Thyroid function diagnosis is an important classification issue. Proper interpretation of the thyroid data, besides clinical examination and complementary investigation, is an important problem in the diagnosis of thyroid disease. The GP model that we will develop will reduce the attributes used in classifying thyroid disorder compared to other existing models and so economically, Ghana as a developing country will benefit immensely in terms of cost reduction.

1.1.5 Organization of the thesis

The thesis is organized in five chapters. Chapter 1 describes the introduction and the background study of the Genetic Programming. Chapter 2 describes the literature review of the thesis. Chapter 3 deals with the methodology. Chapter 4 deals with data collection and analysis and chapter 5 presents the conclusion and recommendations.

Chapter 2

Literature Review

This chapter provides a brief survey on previous works in context of the study. The aim of this chapter is to review relevant literature about symbolic regression and application of genetic programming in general.

Genetic Programming (GP) is the latest paradigm within the research area called Evolutionary Computation (EC). Created by John Koza (first book published in 1992), GP has its origins in Genetic Algorithms (Gas). John Koza was a PhD student of John Holland the father of Gas.The crucial difference between GP and Gas is the representation of individuals. Ga uses fixed length numerical strings where as GP uses variable length structures containing whatever ingredients are needed to solve the problem.

In I954, GP began with the evolutionary algorithms first used by Nils Aall Barricelli applied to evolutionary simulations. In the 1960s and early 1970s, evolutionary algorithms became widely recognized as optimization methods. Ingo Rechenberg and his group were able to solve complex engineering problems through evolutionary strategies as documented in his 1971 PhD thesis and the resulting 1973 book. John Holland was highly influential during the 1970s.

In 1964, Lawrence J. Fogel, one of the earliest practioners of the GP methodology, applied

evolutionary algorithms to the problem of discovering finite-state automata. Later GP-related work grew out of the learning classifier system community, which developed sets of sparse rules describing optimal policies for Markov decision processes. The first statement of modern tree-based genetic programming (that is, procedural languages organized in tree-based structures and operated on by suitably defined GA-operators) was given by Cramer (1985).

This work was later greatly expanded by John R. Koza, a main proponent of GP who has pioneered the application of genetic programming in various complex optimization and search problems. Gianna Giavalli, a student of Kozas, later pioneered the use of genetic programming as technique to model DNA expression.

In the 1990s, GP was mainly used to solve relatively simple problems because it is very computationally intensive. Recently, GP has produced many novel and outstanding results in areas such as quantum computing, electronic design, game playing, sorting and searching, due to improvements in GP technology and the exponential growth in CPU power. The results include the replication or development of several post-year-2000 inventions.

Zhang,(2012) applied genetic programming to solve a symbolic regression. His objective was to find a function fitting the given input-output data as close as possible. In his experiment, he observed that the function and terminal sets had a powerful impact on the performance of the GP. Too many functions and terminals increased the search space exponentially, resulting in long training times and more local minimums, so the algorithm was more likely to be trapped by some suboptimal solutions. On the other hand, too few

functions failed in finding a fitting function. Secondly, the depth of trees can influence the search space. Constraint of the maximum depth of trees in the training helped find good functions more quickly. Thirdly, too low variety made the solution converge to some local minimums. He concluded that the size of population, the selection method and selection percentage are main factors that affect the variety of generations.

Dufy et al,(2010) used genetic programming symbolic regression technique for inferring the strategies that are being played by subjects in economic decision, making experiments and applied to experimental data from the repeated ultimatum game. Their goal was to uncover player's strategies from the actions these players played. The main advantage of their approach over standard regression analyses was that they did not have to prespecify the structure of the regression equation. Instead they only needed to provide a set of terminals and non terminals.

Basim et al,(2012) also used genetic programming for symbolic regression of crop pest forecasting and compared their GP model to the ordinarily linear regression model. Based on their experiments, the developed GP model that is based on statistical calculations of the RMS accomplished good results and can successfully build a new model with a precision of around 99.9%. Finally, from their obtained results, the GP model presented a more advanced mathematical function with a set of independent variables and the result indicated that their proposed approach was a valuable approach and can significantly support an accurate and automatic prediction of a fitted model compared to the linear regression model.

Kotanchek et al,(2012) used genetic programming for symbolic regression to extract outliers from a data set. Their model-based outlier detection approach were used to identify records which are systematically under- or- predicted by diverse ensembles of (thousands of) global non-linear symbolic regression models. They applied their proposed method to a country data and produced an insight into outlier vs. prototypes division among world countries and therefore used it to predict gross domestic product (GDP) per capita.

Handley,(1993) used genetic programming to predict the shape of proteins. He was able to evolve programs which, using the protein's chemical composition, were able to predict whether each part of a protein would have a particular geometric shape (an $\alpha - helix$) or not. Genetic programming was able to do this broadly as well as other techniques but all suffered from the fact that the structure depends upon more than local composition.

Andre has successfully used genetic programming in optical character recognition problems (OCR). In Andre,(1994), he combines genetic programming with a two dimensional genetic algorithm to produce an OCR program from scratch. In Andre1,(1994), he showed that genetic programming can be used to maintain existing hand coded programs. He shows genetic programming automatically extending an existing manually written OCR program so that it can be used with an additional font.

Perry,(1994) used GP to predict winning horses using horse racing past performance data. He divided the data into training and test sets, using only the former to evolve the prediction program, and the latter to check the generality of the best evolved program. The results of the best program showed profit levels of 181.2% during the training phase and

111.1% during the test phase

Faris (2013) used a symbolic regression approach via genetic programming (GP) to model a cutting machine temperature and compared to other approaches which based on estimating the parameters of the nonlinear regressive curve of the cutting tool. They claim that their developed GP model showed promising results compared to models developed based on parameter estimation such as Least Squares regression (LS) Genetic algorithms (GA) and Particle Swarm Optimization (PSO). The four approaches were compared using different evaluation criteria (VAF, MSE, ED and MD).It can be noticed that the symbolic regression model developed by the genetic programming, outperformed all other techniques.

One of the areas that GP has also been applied is time series prediction. Koza,(1990), the father of GP, gave the first example on time series prediction by symbolic regression in his book "Genetic Programming", by reinventing a known econometric equation. Since he was successful, he proved the liability of symbolic regression with GP. In this example, Koza used a canonical GP without further enhancements or tunning.

Yoshihara et al,(2000) suggested a combination of GP and MA for time series .This way, they removed the restrictions of GP to static predetermined constant- terminals as elements of the GP tree. They used a canonical GP for symbolic regression and each generation they used an additional local search algorithm to optimize the constant- terminals used in the GP tree. They basically implemented a Lamarckian model of MA, since the descendants inherited the optimized constant- terminals from their parents.

Santini et al,(2001) were the winners in a contest initiated by the CEC 2000 to predict

the Dow Jones. First they planned to use ARCH and GRACH models for prediction and optimize the parameters of these models with EA methods. But later they found that a simple GP for multiple point prediction outperformed the optimized ARCH and GARCH. A different strategy to predict time series would be to develop trading rules that make a simple short term predictions, whether a given time series will rise or fall in the near future. A predictive trading rule will most likely be profitable , if it is used to trade the asset represented by the time series and a lot of people have used GP for generating trading rules.

One of the earliest papers using GP was published by Allen et al,(1999). In their paper, they found that GP was not able to outperform the Buy and Hold strategy on the Out-Of-Sample set.This was also the case if trading cost was added. But Allen and Karjalainen noted that GP was able to identify relevant indices, use them for prediction and to ignore irrelevant ones.

A critique of the work by Allen and Karjalainen was published by Ready,(1997). He noted that the transaction costs and the occurrence of the actual transaction relative to the signal produced by the trading rule were not correctly implemented. He noted again that the rules found by Allen and Karjalainen had expired after five years. Ready therefore suggested that trading rules should be periodically recreated to meet changes in the market. But in the end he sustained the work of Allen and Karjalainen in spite of the proposed improvements he made, the GP was unable to beat Buy and Hold strategy on the Out-Of Sample set.

O'Neil et al,(2001) also criticized the work of Allen and Karjalainen, claiming that Allen and Karjalainen did not note that trading rules generated by the GP were only in the market for about 60% of the time the trading rules should have been assigned a more positive risk profile than simple Buy and Hold strategy. Here they used a variation of the canonical GP, the Grammatical Evolution (GE) to find profitable trading rules on the UK FTSE 100 index. They found that the rules developed with GE considerably outperformed the simple Buy and Hold strategy.

Jonsson et al,(1997) used a canonical GP to find trading rules on international currency markets. They repeated the optimization process periodically to prevent over fitting of the trading rules. They also used a realistic trading model with transaction costs on intraday data. They emphasized that their GP approach was able to find trading rules that produced a higher return than the Buy and Hold on the Out-Of Sample set.

Another publication was made by Iba et al,(1999) and they used a GP to find trading rules on the Nikkei 225. Here they trained Neural Net as benchmark. They seem to be less accurate with the implementation of trading costs and the time lag between the trading signal and the actual trade. But one of the most important results they presented is independent of these inaccuracies. They found that the predictive power of the rules found was dependent on the operators that were allowed to occur in the GP program tree. AGP program tree that was restricted to simple operators like $\{+, -, *, /\}$ was performing worse than a tree that could use more sophisticated time series operators like moving average.

Andrews et al,(1994) used genetic programming to create strategies which have traded in

simulated commodity and futures markets (the double auction tournaments held by the Santa-Fe Institute, Arizona, USA). Their automatically evolved strategies have proved superior to many hand-coded strategies. A number of commercial firms are also active in this area.

Artificial Life is the study of natural life by computer simulation. There is a ready connection to autonomous robots. For example, evolving a program to control a robot's leg may be considered either as an engineering problem or as simulation of real insect's leg.

Spencer,(1993), used GP to generate programs that enable a "six-legged insects" to walk in a simulated environment. He reports that in every experiment, GP evolved programs capable of the efficient walking and furthermore that the performance of these programs were at least as good and often better than that of hand-generated programs.

Banzhaf,(1993) proposed an extension to the genetic programming paradigm which allowed users of traditional Genetic Algorithms to evolve computer programs. He devised a GP system using a traditional GA binary string representation. The mechanisms of transcription, editing and repairing are used to convert these bit strings into computer programs. Each individual in the population consists of a string of 225 bits. The 225 bits are interpreted as sequence of 5-bit words. A transcription table is used to map these 5-bit words to functions and terminals. The bit strings are made to satisfy two requirements by a repair operator. These requirements are that the number of terminals in the resulting program be greater than the number of functions, and secondly that each sequence begins with a function call. Bit level mutation and crossover genetic operators were used to generate new individuals.

In almost all genetic programming work, the terminals and functions are chosen so that they can be used with each other without restriction. This is achieved by requiring that the system be closed so that all terminals are of one type and all functions can operate on this type and always yield a result also of this type. For example measures are taken to protect against division by zero errors.

In Montana's Strongly Typed Genetic Programming, Montana,(1994), a genetic program can contain multiple different types simultaneously. When manipulating the program (eg. using crossover) care is taken that not only are there the correct number of arguments for each function but that they are of the expected type. This leads to an increase in the complexity of the rules that must be followed when selecting crossover points. However Montana shows that tight type checking can considerably reduce the size of the search space, which he says will reduce the effort to find the solution. Generic types are allowed in order to reduce the number of functions which must be explicitly specified.

Where as with module acquisition, subtrees are chosen at random to become modules, Rosca et al,(1994) perform an analysis of the evolutionary trace to discover building blocks. Building blocks are compact subtrees which add to the fitness of individuals. These fit blocks are parameterized and then added to the function set to extend the representation. The current population is then enriched by replacing individuals by new randomly created individuals using the new function set. In their experiments they compared the application of standard GP, GP with ADFs and GP with adaptive representations to EVEN-N-PARITY problems of difficulty ranging from $N = 3$ to $N = 8$. The results they report indicate su-

perior performance compared to both of the other methods in terms of the number of generations required to fined the solution and the size of the solution found.

Langdon,(1995) used GP with index memory to successfully evolve queue and stack abstract data types (ADTs) using a chromosome structure with multiple trees. Each tree in the chromosome was used to represent one of the required ADT member functions. The results suggest that it may be possible to extend the level of abstraction in GP from pure functional abstraction as in ADFs, NAs and adaptive representations to full ADTs. This may help increase the scalability of GP.

Andre,(1994), used GP to evolve programs that are capable of storing a representation of their environment (map-making), and then using that representation to control the actions of a simulated robot. He used GP to effectively co-evolve two programs (each making use of ADFs) for each individual. He split the fitness evaluation of each individual into two phases hence the term multi-phase fitness. In the first phase, the first program is run to extract and store features from the environment. Only the first program is provide with the set of functions to probe the environment. In the second phase, the second program is executed to use the stored representation to control the actions of a robot. The fitness vale was awarded only for the success of this second phase. He found that using this approach, he was able to successfully evolve programs that solve simple problems by examining their environment, storing a representation of it, and then using the representation to control the action.

Tackett,(1993), describes the use of genetic programming to extract targets from low con-

trast noisy pictures. Various standard metrics are abstracted from the image using standard techniques, which are then processed by a genetic program to yield target details. In his experiments, Tackett compared GP with a neural network and a binary tree classifier system. He reports better results from GP than both of the other techniques. Furthermore, the ability to analyse the evolved program enabled him to get insight into the features that assist in pattern discrimination, which would not have been possible with the "more opaque methods of neural learning".

Genetic programming has been used to solve diseases related problems such as diabetes.Muhammed et al.(2010), used genetic programming (GP) and a variation of genetic programming called GP with comparative partner selection (CPS) for diabetes detection. The proposed system consisted of two stages. In the first stage, they used genetic programming to produce an individual from training data, that converted the available features to a single feature such that it has different values for healthy and patient (diabetes) data. In the next stage, they used test data for testing of that individual. They claim that their proposed system was able to achieve $78.5^{+}_{-}2.2\%$ accuracy. Their results showed that GP based classifier can assist in the diagnosis of diabetes disease.

Again Sharief et al.,(2014) used a multigene genetic programming technique to detect diabetes based on set of attributes collected from patients.Their technique showed significant advantages on evolving nonlinear model which can be used for prediction. Their developed GP model was evaluated using Pima Indian data set and showed higher capability and accuracy in detection and diagnosis of Diabetes.

Guo et al,(2006) proposed a method for breast cancer diagnosis using the feature generated by GP. They used a modified Fisher criterion for this purpose. They developed a new feature extraction measure (modified Fisher linear discriminant analysis (MFLDA)) to overcome the limitation of Fisher criterion. GP as an evolutionary mechanism provides a training structure to generate features. They developed a modified Fisher criterion to help GP optimize features that allow pattern vectors belonging to different categories to distribute compactly and disjoint regions. First, the MFLDA was experimentally compared with some classical feature extraction methods (principal component analysis, Fisher linear discriminant analysis, alternative Fisher linear discriminant analysis). Secondly, the feature generated by GP based on the modified Fisher criterion was compared with the features generated by GP using Fisher criterion and an alternative Fisher criterion in terms of the classification performance. The classification was carried out by a simple classifier (minimum distance classifier). Finally, the same feature generated by GP was compared with a original feature set as the inputs to multi-layer perceptrons and support vector machine. Their results demonstrated the capability of this method to transform information from high-dimensional feature space into one-dimensional space and automatically discover the relationship among data, to improve classification accuracy.

Oakley,(1994) describes obtaining blood flow rates within human toes using laser Doppler measurement. These measurements are both noisy and chaotic. He compares the effectiveness (at removing noise but preserving the underlying signal) of special filters evolved by genetic programming and various standard filters. He concludes that a combination of

genetic programming and heuristics is the most effective.

In terms of education , genetic programming has been used to predict students failure. Carlos et al,(2012) proposed genetic programming algorithm model to obtain accurate and comprehensive classification rules. They used a real data containing about 670 high school students from Zacatecas, Mexico.

Jinjun,(2007) proposed a method called seed selection genetic programming to find an approximate analytic solution to differential equations. In his work, he claimed that his method has three advantages over the normal genetic programming as the latter has a convergence speed to be slow, finding is blindness and easily gains into local infinitesimal, but his method overcomes these problem as the initialization of population is seeded.

Genetic programming has successfully been used to evolve new constructive heuristics comparable to human designed heuristics for a number of problem domains.

Burke et al(2006) showed that stand-alone heuristics generated using genetic programming could outperform the human designed 'bestfit' heuristics on unseen instances of the same class of one dimensional bin packing problems. This work was extended to three dimensional bin packing by Allen.

Drake et al,(2009) investigated the suitability of using genetic programming as a hyper-heuristic methodology to generate constructive heuristics to solve the multidimensional 0-1 knapsack problem. Their work was that, a population of heuristics to rank knapsack items are trained on a subset of test problems and then applied to unseen instances. The results over a set of standard benchmarks show that genetic programming can be used to generate

constructive heuristics which yield human-competitive results.

Poli et al,(2007) used genetic programming to quickly generate disposable heuristics to solve the satisfiability problem. Again, this work generated heuristics comparable to those which were human designed. However, only a limited search space of heuristics was covered.

Kumar et al(2008) used genetic programming as a hyperheuristics to evolve heuristics for biobjective 0-1 knapsack problem. This system successfully created reusable heuristics was able to produce a set of Pareto-optimal solutions. The Pareto fronts generated using this approach are indistinguishable from those obtained using the human-designed profit-to-weight ratio heuristic.

Zhao et al. (2000) demonstrated the ability of genetic programming to evolve multiagent cooperation without requiring communication. The simulation involved two agents placed in a grid world, with fitness measured by their ability to exchange places. In- between them was a bottleneck through which only one agent could pass at a time. This restraint meant that the agents had to learn not to get in each other's way. Genetic programming successfully evolved a program that, when executed by both agents, allowed them to exchange locations in several different test worlds.

In research conducted by Karlsson et al. (2000), genetic programming evolved a program that could classify the angle to the location of a sound relative to two input microphones placed inside a dummy head. Programs were given raw digital audio as input, with one sample from each microphone per execution of the program. Two variables were also provided

to store state information between invocations. Programs were then executed repeatedly over all the audio samples recorded for each fitness case. The solutions evolved were able to classify the location of sounds with varying success. Performance was good with saw-toothed waveforms presented at fixed distances from the microphones. Performance was much worse with a human voice presented at varying distances. For all the fitness cases, however, performance was better than random.

Koza (1992) demonstrated that Genetic programming could evolve an agent that took into account multiple sub-goals. The domain was a full simulation of a traditional Pac Man game. The goal was to evolve a player that collected the most possible points in a given period of time. The sub-goals a successful player had to take into account include path finding, ghost avoidance, and ghost seeking after power up consumption. In Kozas experiments, genetic programming was applied to high level functions, such as distance-to-ghost, distance-to-power-up, move-to-ghost, and move-from-ghost, etc. The Pac Man simulation was deterministic (no random numbers were used during the simulation). On the other hand, Koza was able to evolve relatively successful agents for this problem that pursued all three sub-goals, and opportunistically changed between them as the situation varied.

Koza(1992) also showed that genetic programming could evolve programs that exhibit collectively fit behavior when executed in parallel by many identical agents. In the food collection problem, the goal is to evolve agents that move spread-out bits of food into single pile. The fitness was the summed distance of each bit of food from all other bits of food (lower = better). The instruction set consisted of pick-up-food, drop-food, move-randomly,

and conditionals that checked if the agent was near food or holding food. Using this starting point, evolution quickly found programs that, when executed in parallel, collected separate food pieces into larger and larger piles, until all the food was concentrated in one area.

For the 1997 competition, Luke et al., (1997) used evolution to create a program that would competitively play soccer. Single programs were evolved, which were instantiated eleven (11) times to control each player separately. Programs were tree-based, with a conditional instruction and soccer specific instructions like is-the-ball-near-me, return-vector-to-ball, and kick-ball-to-goal. Co-evolution was used to measure fitness by pitting two programs against each other at a time. Fitness was simply the number of goals scored by each team over several short fitness cases.Using this framework, evolution was able to create a program that defeated two human developed teams in the RoboCup 97 competition.

Again Koza (1992) used evolution to create a program that controlled a simulated robot with wall following behavior. The fitness function selected for programs that found the edge of a room and then followed it all the way around its perimeter. The agent was given sensors that reported the distance to the wall at eight (8) equally spaced angles around the robot. In Kozas work, evolved code was constrained into a subsumption themed structure. Specifically, code was inserted into a nested if statement three deep of the form (if statement: action; else if statement). Each if, roughly corresponded to a layer of subsumption. Programs were given access to all of the sensor values, a conditional instruction, and movement functions. Notably, there were no ways for the layers to operate in parallel, so the theoretical relationship to subsumption is limited. Evolution was able to find a program

that fit within the constraints of the code representation and satisfied the fitness function. Starting in a random location, the program would head toward the nearest wall, and then follow it all the way around the room.

In research conducted by Andersson, et al. (2000), genetic programming was shown to be able to evolve a controller that produces walking behavior for a real robot with four legs and eight degrees of freedom. In their research, programs were evolved that were executed multiple times per fitness case. For each execution, a program outputs the angles that each motor should turn to. To maintain state between executions, a program is given the output angles calculated by the previous execution. Fitness was the distance the robot moved forward during the time the program controlled it. The instruction set only included basic mathematical operations. Given this structure, genetic programming was able to evolve controller programs that moved the robot forward from the origin using a variety of different methods. These included chaotic paddling behavior, crawling, carefully balanced walking where three legs always maintained contact with the floor, and dynamic walking gates (like used by most animals when running) where balance is maintained via forward momentum. Frequently, evolution progressed through several stages, starting with paddling behavior and eventually finding balanced walking or dynamic gates.

Although in the literature GP has been applied to health but nothing is said on the classification of thyroid disorder and therefore we will use GP to explore into this area of application.

Chapter 3

Methodology

3.1 Introduction

Genetic programming (GP) is a component of evolutionary algorithm and therefore, it is imperative to look at evolutionary algorithm before we delve much into the GP.

Evolutionary Algorithms (EAs) refers to a subset of algorithms in evolutionary computation that are generic population-based metaheuristic optimization algorithms. The origin of evolutionary algorithms (EAs) was an attempt to mimic some of the processes taking place in natural evolution.Sivananda et al,(2008). EAs use mechanisms inspired by biological evolution: reproduction, mutation, recombination and selection. In the most general terms, evolutionary algorithm can be described as two-step iterative process, consisting of random variation followed by selection. Just as natural evolution starts from an initial population of creatures, the algorithmic approach begins by selecting an initial set of contending solutions for a particular problem. The set may be chosen by generating solutions randomly or by utilizing any available knowledge about the problem. These parent solutions then generate offspring by a preselected means of random variation. The resultant solutions are evaluated for their effectiveness-their 'fitness' and undergo selection. Just as

nature impose the rule of survival of the fittest those solutions that are the least fit are removed from further consideration, and the process is repeated over successive generations. An evolutionary algorithm begins by initializing a population of solutions .Each individual (member of the population) is a candidate solution. The goodness of individuals is determined by a fitness function. Evolutionary algorithms have been highly successful in solving complex problems in science and engineering. Figure 3.1 describes the general overview of evolutionary algorithms.

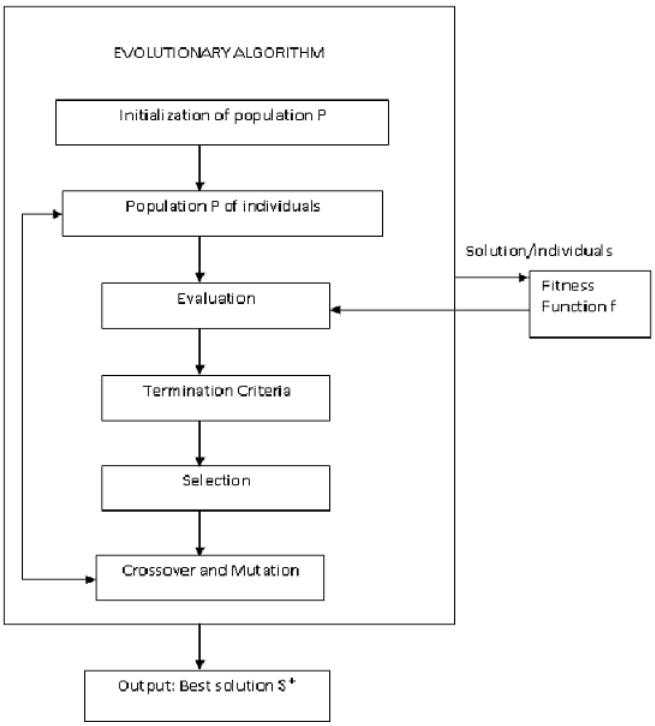

Figure 3.1: Evolutionary Algorithm

In the case of evolutionary algorithm, there are four historical paradigms that have served as the basis for much of the activity of the field: genetic algorithms (Holland, 1975), genetic programming (Koza, 1992, 1994), evolutionary strategies (Recheuberg, 1973), and

evolutionary programming (Forgel et al., 1966). The basic differences between the paradigms lie in the nature of the representation schemes, the reproduction operators and selection methods.

The goal of GP is to evolve computer programs. Given all the elements of a programming language, GP has a potential to find the computer that solves a particular problem. Theoretically, GP can solve any problem whose candidate solutions can be measured and compared in terms of quality or "how well they solve the problem''

3.1.1 GP Algorithm

There are a lot of steps needed to run GP and these are:

1. Population initialization
2. Fitness Evaluation
3. Selection for Breeding
4. Genetic operators
5. Selection for survival
6. Stopping criterion

In order to have a good understanding of the Genetic programming algorithm, it is necessary to first have a good understanding of some key terms as used in Genetic programming problems.

3.1.1.1 Population Initialization

The generation of initial population of GP run is performed by the random generation of individuals. Starting from the root node of program tree, one of the functions from the function set is selected at random to be the root node. From this root function, functions or terminals are selected at random from the function and terminal sets to form the argument of the root function. Subsequently for each function selected, new functions or terminals are selected to form the arguments of the new function, and eventually filling out the tree. In basic GP, each function and terminal operate on one data type, and so any function and terminal can be used as an argument for any other function. This means that generated programs are synthetically correct. Initial trees are generated randomly in such a way that they don't exceed a certain depth (typically six (6)). Koza described three initialization methods namely:

1. Full
2. Grow
3. Ramped Half-and-Half

Both the Grow and Full method are subject to a pre-established maximum depth D_{max}. The depth of a node in a tree is the number of edges that need to be traversed to reach that node when starting from the tree's root node (which is considered to have a depth of 0). The depth of a tree is the depth of its deepest leaf.Starting from the root of the tree, nodes are added until the leaves(terminal set). The choice of node is mostly random among

the function and terminal sets, obeying the restrictions.

3.1.1.2 Full Method

In the full method, nodes are taken at random from the function set until the maximum tree depth is reached, and beyond that depth only terminals can be chosen. That is, all the leaves of the generated trees are at the same depth, hence the method generates full trees of size D_{max}.The range of program sizes (i.e. measured as the total number of nodes in a tree) and shapes this method can produce is rather limited. Figure 3.2 illustrates full method.

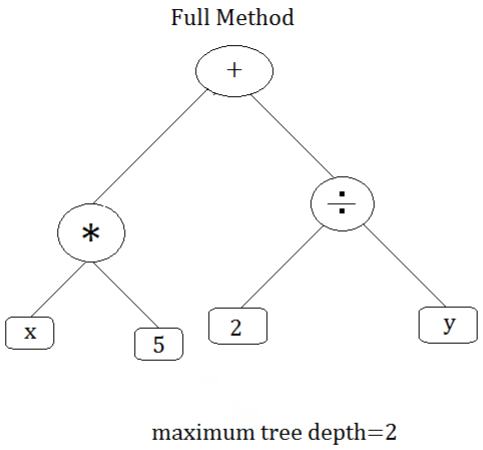

Figure 3.2: Full Method

3.1.1.3 Grow Method

Where the full method generates trees of specific size and shape,the grow method allows for the creation of trees of varying size and shape. Here nodes are selected from the whole primitive set (i.e. functions and terminals) until the depth limit is reached, below which only terminals may be chosen.

The size and shapes of the trees generated via grow method are highly sensitive to the sizes of the primitive sets (ie. Sizes of functions and terminals).If one has significantly more terminals than functions, the grow method will almost always generate very short trees regardless of the depth limit. Similarly, if the number of functions is considerably greater than the number of terminals, then the grow method will behave quite similar to the full method.Figure 3.3 illustrates grow method

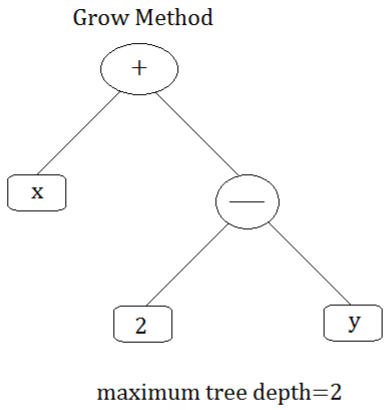

Figure 3.3: Grow Method

3.1.1.4 Ramped Half-and-Half

For each depth between two and the maximum depth, half of the trees are initialized with grow method and the other half with full. The process uses a range of depth limits in order to ensure the creation of programs with various sizes and shapes. This method has become one of the most frequently used GP initialization techniques recently.

3.1.1.5 Fitness Function (Measure)

The fitness measure is derived from the high- level statement of the problem. The first two preparatory steps define the primitive set for GP; and therefore, indirectly define the search space GP will explore. This includes all the programs that can be constructed by composing the primitives in all possible ways. However at this stage, we still do not know which elements or regions of this search space are good. (i.e. programs that solve or approximately solve the problem). This is the task of the fitness measure, which effectively (even though implicitly) specifies the desired goal of the search process. Typically fitness measures for symbolic regression problem are:

- Absolute differences between expected and obtained results, summed for all samples of the data set.

- Root mean square error (RMSE)

  When selecting randomly a tree to perform any genetic operation, the lexicography tournament selection method is used in this thesis. This method specifies the proba-

bility of selection on the basis of the fitness of the solution. The fitness of a solution shall reflect:

1. The quality of approximation of the experimental data by a current expression represented by a tree.
2. The length of the tree in order to obtain more compact expressions.

In problems of empirical model building, the most obvious choice for the estimation of the quality of the model is the sum of squares of the difference between the simplified model output and the results of the runs of the original model over some chosen plan (design) of experiments. Generally there can be two sources of error: incorrect structure and inaccurate tuning parameters. In order to separate these, the measure of quality (better) $B(S_i)$ is only calculated for the tuned approximation.In a dimensionless form this measure of quality of the solution can be presented as follows:

$$B(S_i) = \frac{\sum_{p=1}^{p}(F_p - \tilde{F}_p)^2}{\sum_{p=1}^{p} F_p^2}$$

If $B(S_i)$ is the measure of quality of the solution S_i, B_u is an upper limit value of the quality for all N_t members of the population, $ntpi$ is the number of tuning parameters contained in the solution S_i and c is a coefficient penalizing the excessive length of the expression, the fitness function $\omega(S_i)$ can be expressed as:

$$\omega(S_i) = B_u - B(S_i) - c * ntpi^2 \rightarrow max$$

Programs with greater fitness values $\omega(S_i)$ have a greater chance of being selected in a subsequent genetic action. Highly fit programs live and reproduce, and less fit programs die.

In terms of computer implementation of the GP paradigm, it is more convenient and more efficient to make the best value of the fitness 0, i.e. to solve a minimization problem. Another possible choice is the statistical concept of correlation, which determines and quantifies whether a relationship between two data sets exists.To evaluate the goodness-of-fit, the standard root mean square error (RMSE) will be used in this thesis according to the following expression:

$$RMSE = \sqrt{\frac{\sum_{p=1}^{p}(F_p - \tilde{F}_p)^2}{p}}$$

where,

F_p = actual values observed and $\tilde{F}_p$ = predicted values.

3.1.2 Breeding

It is in this process, the search process creates new and hopefully fitter individuals. The breeding cycle consists of three steps:

1. Selecting parents.

2. Crossing the parents to create new individuals (offspring or children).

3. Replacing old individuals in the population with the new ones.

3.1.2.1 Selection

Selection is the process of choosing two parents from the population for crossing. The purpose of selection in GP is to provide the mechanism which ensures that the survival of the fittest principles is respected. This means that, typically, the probability of an individual to have offspring which inherit its genetic information (i.e. traits or characteristics) is proportional to its fitness (i.e. its respective level of desirable characteristics): the better an individual's fitness value, the higher the probability its genetic material will survive within the individuals of the next generation. The various selection methods are discussed as follows:

1. Roulette Wheel Selection

   Roulette process can be explained as follows: The expected value of an individual is that fitness divided by the actual fitness of the population. Each individual is assigned a slice of the roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation.

2. Random Selection

   This technique randomly selects a parent from the population. In terms of disruption of genetic codes, random selection is a little more disruptive, on average, than roulette wheel selection.

3. Rank Selection

   Rank Selection ranks the population and every program receives fitness from the ranking. The worst has fitness 1 and the best has fitness N. It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent.

4. Tournament Selection

   Unlike, the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among Nu individuals. The best individual from the tournament is the one with the highest fitness, which is the winner of Nu. Tournament competitions and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GP to improve the fitness of the succeeding genes. This method is more efficient and leads to an optimal solution. However Lexicographic tournament selection optimizes both the fitness level and the size of the tree.

3.1.3 Genetic Operators

3.1.3.1 Crossover (Recombination)

Crossover is the process of combining the attributes of two chromosomes to produce two new chromosomes that inherit some (or all) of their attributes from one (or both) of their parents. To determine which individuals undergo crossover, there are two possibilities. The first and more common method is to select both parents based on fitness, ensuring the genetic material from both parents is a component of fit individuals. An alternative, which does have merit is to select the first individual based on fitness, and the second at random. This tends to increase the diversity of a population, while maintaining some selective pressure. Crossover is a recombination operator that proceeds in three steps:

- The reproduction operator selects at random a pair of two individual trees for the mating.
- A cross site is selected at random along the trees.
- Finally, the position values are swapped between the two trees following the cross site.

The various crossover techniques are discussed as follows:

1. Single Point Crossover (Sub tree crossover)

   The traditional GP uses single point crossover, where the two mating trees are cut once at corresponding points and the sections after the cuts exchanged. A random

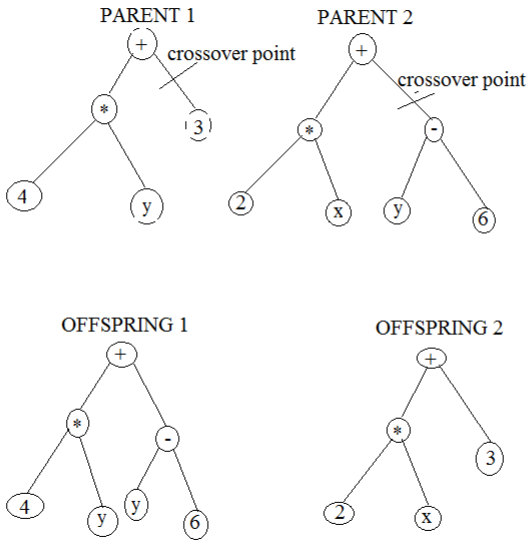

Figure 3.4: Crossover illustration

sub tree is selected from both the father and the mother. The father's sub tree is replaced by the mother's sub tree and the new individual is the offspring.

2. Two Point High level Crossover

   In two point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents.

3.1.3.2 Mutation

After crossover, the programs are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly disturbing genetic information. In the case of GP there are two main forms of mutation:

1. subtree mutation

   In this type of a mutation, a mutation point (node) is randomly selected and the subtree rooted in that node is either replaced by a terminal or by a randomly generated subtree.

2. Point mutation

   In this type of mutation, the function stored in a randomly selected function node (mutation point) is replaced by a different randomly chosen function with the same arity (if no other function with the same arity exists, the function node can be turned into a terminal node, or the mutation operation on that node can be canceled)

When subtree mutation is applied, exactly one subtree per selected individual must be modified. However, point mutation is applied on per-node basis, which means that during one application of point mutation on a selected individual, each node in the tree is considered for mutation with a given probability. Figure 3.5 illustrates mutation in a tree form. The choice of which genetic operators should be used to create an offspring is probabilistic. Also, in this case, genetic operators are mutually exclusive (unlike in most evolutionary techniques where offspring are obtained through a composition of operators). Their probabilities of application are called genetic operator rates. Usually, crossover is applied with a very high rate of 90% and above, where as the mutation rate is rather small, typically in the neighborhood of 50%.

When the rates of crossover and mutation sum up to a rate of p which is smaller than 100%, another genetic operator called reproduction (elitism) must be used with a rate of

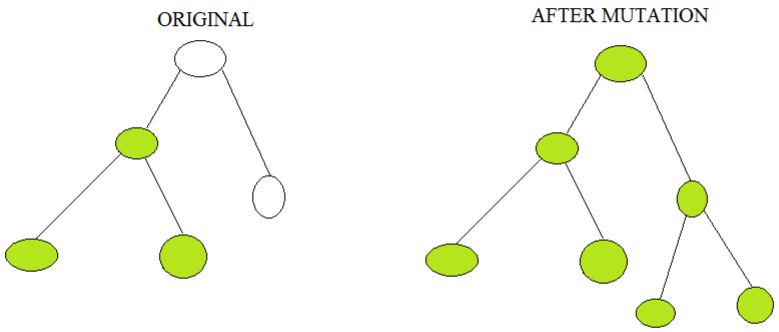

Figure 3.5: Mutation illustration

$1-p$. Reproduction consists of simply selecting an individual based on fitness and inserting a copy of it in the next generation.

3.1.3.3 Replacement

Replacement is the last stage of any breeding cycle. Two parents are drawn from a fixed size population, they breed two children, but not all four can return to the population, so two must be replaced i.e., once off springs are produced, a method must determine which of the current members of the population, if any, should be replaced by the new solutions. Basically, there are two kinds of methods for maintaining the population; generational updates and steady state updates.

The basic generational update scheme consists in producing N children from a population of size N to form the population at the next time step (generation), and this new population of

children completely replaces the parent selection. In a steady state update, new individuals are inserted in the population as soon as they are created, as opposed to the generational update where an entire new generation is produced at each time step. The insertion of a new individual usually necessitates the replacement of another population member. The individual to be deleted can be chosen as the worst member of the population. (it leads to a very strong selection pressure), or as the oldest member of the population.

3.1.4 Elitism

An additional operator, elite transfer, is used to allow a relatively small number of the fittest programs, called the elite, to be transferred unchanged to a next generation, in order to keep the best solutions found so far. As a result, a new population of trees of the same size as the original one is created, but it has a higher average fitness value.
To run a GP system to solve a problem, a small number of ingredients, often termed as preparatory steps, need to be specified:

1. The terminal set
2. The function set
3. The fitness measure
4. Certain parameters for controlling the run
5. The termination criterion and the method for designating the result of the run

3.1.5 Terminal Set

The terminal set may consist of:

- The programs external inputs, typically taking the form of named variables (say x,y)
- Functions with no arguments, which are, therefore interesting either because they return different values in different invocations (e.g. The function rand () that returns random numbers) or because they produce side effects (such as go-left ())
- Constants, which can either be pre-specified or randomly generated as part of the tree creation process.

The terminal set contains nodes that provide an input to the GP system.

3.1.6 Function Set

The set contains nodes that process values already in the system. The function set used in GP is typically driven by the nature of the problem domain. In a simple numeric problem, for example, the function may consist of merely the arithmetic functions $(+, -, *, /)$. However all sorts of other functions and constructs typically encountered in computer programs can be used. All the functions and terminals must be compatible in order to faultlessly pass information between each other (closure property). The sufficiency property requires the identification of functions and terminals so that their combination can yield a valid approximation in the solution domain, e.g. including sufficient number of variables in the terminal set to describe the optimization problem.

3.1.7 Parameters for controlling the run, termination criterion and solution designation

The fourth and fifth preparatory steps entail specifying the control parameters for the run. The most important control parameter is the population size. Other control parameters include the probabilities of performing the genetic operations, the maximum size for programs and other details of the run.

A general, domain independent, termination criterion for a GP process is to monitor the number of generations and terminate the algorithm as soon as given limit is reached. Another widely used termination criterion is to stop the algorithm after the fitness values of several successive best-of-generation individuals appear to have reached a plateau (i.e. evolution has reached a phase where no new progress seems possible) Sreeter et al.,(2003). Domain specific termination criteria are also very frequently used. the algorithm is terminated as soon as a problem specific success predicate is fulfilled. In practice it is quite common to use a combination of both domain independent and domain specific termination criteria. After the algorithm terminates, it is time to choose the result the algorithm will return. Typically, the single best-so-far individual is harvested and designated as the result of the run, Sreeter et al.,(2003). In some cases, it might be necessary to return additional individuals or data depending on the problem domain. There are some applications, like data based structure identification, in which returning the best-so-far individual produced by the GP process during training is not the optimal strategy. A far better approach is to have two disjoint data sets; one that is used for GP training, and another that is used for

validation. The programs produced by GP process are eventually tested on the validation data set and the best performing individual on validation data is returned as the result of the run.

3.1.8 Symbolic regression

One of the key elements in solving a problem using genetic programming is finding an appropriate fitness function. In some problems, only the side effects of executing the programs present interest and as such, the role of the fitness function is to compare the side effects of the execution of different programs in a given environment. Though in many other problems, the side effects of running programs present no interest, the goal is just to find a program whose output has some desired property (i.e. the output must match some target values). In this case the fitness function is usually a measure of the difference between the output of the program and the ideal (i.e. the desired output values). When taking into account the fact that when using an appropriate function base, the programs produced by GP are in fact mathematical formulas, the main task becomes that of inducing accurate mathematical expressions from given data. These resulting mathematical expressions are nothing more than models that try to explain the given data. This is generally known as symbolic regression problem. Regression can be considered as one of the simplest forms of inductive learning. Regression can be described as finding the coefficients of a predefined function (i.e. finding a model based on that function) such that the resulting expression best fits some given data. Poli et al,(2008) presents some of the issues related with the

classical approach to regression analysis. Firstly, if the fit is not good, the experimenter has to keep trying different types of functions by hand until a good model is found. This is very laborious and the results depend very much on the skill and inventiveness of the experimenter. Again, another problem is related to the fact that even expert users tend to have a strong mental bias when choosing the type of functions used for regression (e.g. linear, quadratic, exponential, etc).

Symbolic regression is an attempt to go beyond the limitations imposed by the classical regression. When GP is used to build an empirical mathematical model of data acquired from a process or system, the GP is often known to be performing a symbolic regression. This is a type of regression analysis that searches the space of mathematical expressions to find the model that best fit a given data set, both in terms of accuracy and simplicity. Unlike the classical regression analysis, no particular model is provided as a starting point to the algorithm and no priori assumptions are imposed. Both model structures and parameters are determined automatically and hence has larger search space to search.Searson,(2009),

A multigene individual consists of one or more genes, each of which is a traditional GP tree. Genes are acquired incrementally by individuals in order to improve fitness (e.g. to reduce a model's sum of squared errors on a data set). The overall model is a weighted linear combination of each gene. The optimal weights for the genes are obtained (using ordinary least squares to regress the genes against the output data). The resulting pseudo-linear model can capture non-linear behavior.

The structure of multigene symbolic regression models is illustrated in Figure 3.6

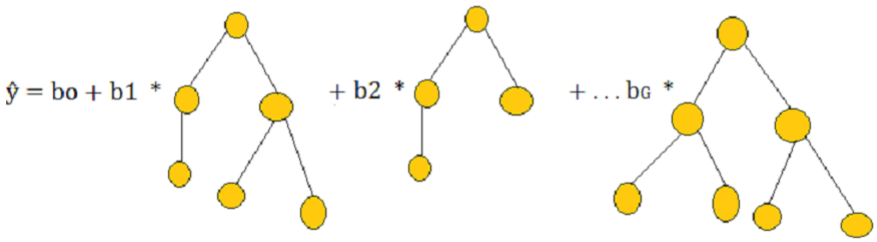

Figure 3.6: Multigene symbolic regression: The prediction of the response data $\hat{y}$ is the vector output of G trees modified by bias term b_o and scaling parameters $b_1, \ldots, b_G$.

3.1.9 Estimation of the coefficient of the Regression parameters

The method of least squares is typically used to estimate the regression coefficients in the regression model. The prediction of the y training data is given by:

$$\hat{y} = b_0 + b_1 t_1 + b_2 t_2 + \cdots + b_G t_G \tag{3.1}$$

where t_i is the $(N \times 1)$ vector of outputs from the ith tree(gene) comprising a multigene individual.

Next, define G as a $(N \times (G+1))$ gene response matrix as follows in $G = [\mathbf{1}t_1, \ldots, t_G]$ where the $\mathbf{1}$ refers to a $(N \times 1)$ column of ones used as a bias(offset) input. Now (3.1) can

be rewritten as:

$$\hat{y} = Gb$$

The least squares estimate of the coefficients $b_0, b_1, b_2, \ldots, b_G$ formulated as a $((G+1) \times 1)$ vector can again be computed from the training data as

$$b = (G^T G)^{-1} G^T \tag{3.2}$$

We note that the optimality of the estimates of b is only true if a number of assumptions are met such as independence of the columns of G and normally distributed errors. In practice the assumptions are rarely strictly met, and the columns of the gene response matrix G may be collinear i.e. linearly dependent (e.g. due to duplicate genes in an individual) and so the Moore-Penrose pseudo-inverse through singular value decomposition (SVD) is used in (3.2) instead of the standard matrix inverse.

Singular Value Decomposition (SVD)

Generally, the SVD finds application in problems involving large matrices, with dimensions that can reach into the thousands as it can turn a singular problem into a non-singular one. SVD is based on a theorem from linear algebra which says that a rectangular matrix A can be broken down into the products of three matrices - an orthogonal matrix U, a diagonal matrix S, and the transpose of an orthogonal matrix V.

Theorem 1 ***SVD:*** *Let A be a real valued $m \times n$ matrix, where $m \geq n$. Then A can*

be decomposed as follows: $A = USV^T$*, where* $U^TU = I$*,* $V^TV = I$*. The columns of* U *are orthonormal eigenvectors of* AA^T*, the columns of* V *are orthonormal eigenvectors of* A^TA *and* S *is diagonal matrix containing the square roots of eigenvalues from* U *or* V *in descending order. Kirk,(2013)*

Now (3.2) becomes

$$b = US^{-1}V^T \tag{3.3}$$

The following example applies this definition to a 2x3 matrix in order to compute its SVD.

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

In order to find U, we have to find AA^T. The transpose of A is

$$A^T = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix}$$

therefore

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$$

Next, we find the eigenvalues and the corresponding eigenvectors of AA^T. We know that eigenvectors are defined by the equation $A\vec{v} = \lambda\vec{v}$, and applying this to AA^T gives us

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We then write this as systems of equations

$$11x_1 + x_2 = \lambda x_1$$

$$x_1 + 11x_2 = \lambda x_2$$

We then rearrange to get

$$(11 - \lambda)x_1 + x_2 = 0$$

$$x_1 + (11 - \lambda)x_2 = 0$$

We solve for λ by setting the determinant of the coefficient matrix to zero,

$$\begin{vmatrix} (11 - \lambda) & 1 \\ 1 & (11 - \lambda) \end{vmatrix} = 0$$

which works out as

$$(11-\lambda)(11-\lambda)-1\cdot 1=0$$

$$(\lambda-10)(\lambda-12)=0$$

$$\lambda=10,\lambda=12$$

to give us our two eigenvalues $\lambda=10,\lambda=12$. Plugging λ back in to the original equations gives us our eigenvectors. For $\lambda=10$ we get

$$(11-10)x_1+x_2=0$$

$$x_1=-x_2$$

$$\Rightarrow x_1=1,x_2=-1$$

Thus we have the eigenvector $[1,-1]$ corresponding to the eigenvalue $\lambda=10$. For $\lambda=12$, we have

$$(11-12)x_1+x_2=0$$

$$x_1=x_2$$

$$\Rightarrow x_1=1,x_2=1$$

Thus we have the eigenvector $[1, 1]$ corresponding to the eigenvalue $\lambda = 12$

These eigenvectors become column vectors in a matrix ordered by the size of the corresponding eigenvalue. Hence in the matrix below, the eigenvector for $\lambda = 12$ is column one, and the eigenvector for $\lambda = 10$ is column two

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Finally, we have to convert this matrix into an orthogonal matrix by applying the Gram-Schmidt orthonormilization process to the column vectors. We begin by normalizing $\vec{v_1}$.

$$\vec{u_1} = \frac{\vec{v_1}}{|\vec{v_1}|} = \frac{[1,1]}{\sqrt{1^2+1^2}} = \frac{1,1}{\sqrt{2}} = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$$

We the compute

$$\begin{aligned} \vec{w_2} &= \vec{v_2} - \vec{u_1} \cdot \vec{v_2} * \vec{u_1} \\ &= [1,-1] - [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] \cdot [1,-1] * [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] \\ &= [1.-1] - 0 * [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] \\ &= [1,-1] - [0,0] = [1,-1] \end{aligned}$$

and normalize

$$\vec{u_2} = \frac{\vec{w_2}}{|\vec{w_2}|} = [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]$$

to give

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

The calculation of V is similar. V is based on $A^T A$, and so we have

$$A^T A = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 2 \\ -0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

Finding the eigenvalues of $A^T A$ by

$$\begin{bmatrix} 10 & 0 & 2 \\ -0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

which represents the systems of equations

$$10x_1 + 2x_3 = \lambda x_1$$

$$10x_2 + 4x_3 = \lambda x_2$$

$$2x_1 + 4x_2 + 2x_3 = \lambda x_3$$

which becomes

$$(10-\lambda)x_1 + 2x_3 = 0$$

$$(10-\lambda)x_2 + 4x_3 = 0$$

$$2x_1 + 4x_2 + (2-\lambda)x_3 = 0$$

which are solved by setting

$$\begin{vmatrix} (10-\lambda) & 0 & 2 \\ 0 & (10-\lambda) & 4 \\ 2 & 4 & (2-\lambda) \end{vmatrix} = 0$$

continuing, we have

$$(10-\lambda)\begin{vmatrix} (10-\lambda) & 4 \\ 4 & (10-\lambda) \end{vmatrix} + 2\begin{bmatrix} 0 & (10-\lambda) \\ 2 & 4 \end{bmatrix} =$$

$$(10-\lambda)[(10-\lambda)(2-\lambda) - 16] + 2[0 - (20 - 2\lambda)] =$$

$$\lambda(\lambda - 10)(\lambda - 12) = 0$$

Therefore $\lambda = 0, \lambda = 10, \lambda = 12$ are the eigenvalues for $A^T A$. Substituting it back into the original equations to find the corresponding eigenvectors yields for $\lambda = 12$

$$(10-12)x_1 + 2x_3 = -2x_1 + 2x_3 = 0$$

$$x_1 = 1, x_3 = 1$$

$$(10 - 12)x_2 + 4x_3 = -2x_2 + 4x_3 = 0$$

$$x_2 = 2x_3$$

$$x_2 = 2$$

So for $\lambda = 12, \vec{v_1} = [1, 2, 1]$. For $\lambda = 10$ we have

$$(10 - 10)x_1 + 2x_3 = 2x_3 = 0$$

$$x_3 = 0$$

$$2x_1 + 4x_2 = 0$$

$$x_1 = -2x_2$$

$$x_1 = 2, x_2 = -1$$

which means for $\lambda = 10, \vec{v_2} = [2, -1, 0]$. For $\lambda = 0$ we have

$$10x_1 + 2x_3 = 0$$

$$x_3 = -5$$

$$10x_1 - 20 = 0$$

$$x_2 = 2$$

$$2x_1 + 8 - 10 = 0$$

$$x_1 = 1$$

which means for $\lambda = 0, \vec{v_3} = [1, 2, -5]$. Ordering $\vec{v_1}, \vec{v_2}$ and $\vec{v_3}$ as column vectors in a matrix according to the size of the eigenvalue, we get

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 2 \\ 1 & 0 & -5 \end{bmatrix}$$

We continue by using the Gram-Schmidt orthonormilization process to convert the matrix to an orthonomal matrix.

$$\vec{u_1} = \frac{\vec{v_1}}{|\vec{v_1}|} = [\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}]$$

$$\vec{w_2} = \vec{v_2} - \vec{u_1} \cdot \vec{v_2} * \vec{u_1} = [2, -1, 0]$$

$$\vec{u_2} = \frac{\vec{w_2}}{|\vec{w_2}|} = [\frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}}, 0]$$

$$\vec{w_3} = \vec{v_3} - \vec{u_1} \cdot \vec{v_3} * \vec{u_1} - \vec{u_2} \cdot \vec{v_3} * \vec{u_2} = [\frac{1}{\sqrt{30}}, \frac{2}{\sqrt{30}}, \frac{-5}{\sqrt{30}}]$$

Which gives

$$V = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

and so V^T becomes

$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

For S we take the square roots of the non-zero eigenvalues and populate the diagonal with them, putting the largest in s_{11}, the next in s_{22} and so on until the smallest value ends up in s_{mm}. We add a zero column vector to S so that it is of proper dimensions for multiplication between U and V since we have done full SVD. Hence

$$S = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix}$$

3.1.10 Goodness-of-fit of the Model

1. The coefficient of multiple determination, R^2

   The coefficient of multiple determinations R^2 is a measure of the amount of reduction in the variability of Y obtained by using the regressor variables. $X_1, X_2..., X_k$. As in the simple linear regression case, we must have $0 \leq R^2 \leq 1$. R^2 is the fundamental model comparison statistic for comparing models with the same number of predictor.

However, a large value of R^2 does not necessarily imply that the regression model is a good one. Adding a variable to the model will always increase R^2, regardless of whether or not the additional variable is statistically significant. There is misconception about R^2. In general, R^2 does not measure the magnitude of the slope of regression line. A large value of R^2 does not imply a steep slope. Hence we resort to the adjusted R^2.

2. The adjusted R^2

   The adjusted R^2 statistics is simply an adjustment of R^2 that allows comparison to be made between models with different number of regressor variables. In general, the adjusted R^2 statistic will not always increase as variables are added to the model. When comparing two models, the model with the smaller RMS error has the larger adjusted R^2.

The R^2 is estimated as follows:

$$\sum_{i=1}^{n} y_i^2 = y^T y = (\hat{y} + y - \hat{y})$$

$$\epsilon = (y - \hat{y})$$

$$y^T y = (\hat{y} + \hat{\epsilon})^T (\hat{y} + \hat{\epsilon})$$

$$y^T y = \hat{y}^T \hat{y} + \hat{y}^T \hat{\epsilon} + \hat{\epsilon}^T \hat{y} + \hat{\epsilon}^T \hat{\epsilon}$$

But $\hat{y}^T\hat{\epsilon} = 0$

$$\therefore y^T y = (\hat{y} + \hat{\epsilon})^T(\hat{y} + \hat{\epsilon})$$

$$\Rightarrow y^T y = \hat{y}^T\hat{y} + \hat{\epsilon}^T\hat{\epsilon}$$

subtracting ny from both sides,

$$y^T y - ny = \hat{y}^T\hat{y} - n\hat{y} + \hat{\epsilon}^T\hat{\epsilon}$$

which is the same as

$$\sum_{i=1}^{n}(y_i - \overline{y})^2 = \sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Dividing through by $\sum_{i=1}^{n}(y_i - \overline{y})^2$ we get

$$\frac{\sum_{i=1}^{n}(y_i - \overline{y})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} + \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

$$1 = R^2 + \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

$$R^2 = 1 - \frac{SSE}{SST}$$

Where:

$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ =residual sum of squares

$SST = \sum_{i=1}^{n}(y_i - \overline{y})^2$ =total corrected sum of squares of y.

For better understanding of how GP works in the context of symbolic regression, an example is provided below:

Let us assume that we have made observations of a certain phenomenon. For simplicity, we shall also assume that the gathered data are error free. Each observation focused on recording only two variables pertinent to the experiment: X and Y (as shown in Table 3.1). Let Y be the response variable and X be the predictor variable. As such the goal is to find

Table 3.1: Input data for example experiment

X	-4	-3	-2	-1	0	1	2	3	4
Y	14.2	5.8	-0.2	-3.8	-5	-3.8	-0.2	5.8	14.2

a model that attempts to explain the data of Y based on the data of X. This example is based on synthetic data, meaning that the values of the set Y have been obtained from the values of the set X by simply using the formula:

$$y = f_{target}(x) = 1.2 * x^2 - 5 \tag{3.4}$$

which is also the target for our symbolic regression problem. In GP based symbolic regression, solution candidates are evaluated by applying them to X, thus obtaining the estimated values E. Finally, in order to asses the fitness of the candidates, the estimated values are compared to the known original (target) values Y. We shall consider formula (3.5) and (3.6) as solution candidates (initial population of solutions) in the GP process that is aimed at solving our regression problem. Here the function set = $\{*, -\}$ and the terminal set = $\{x\}$.

$$f_1(x) = 1.4 * x - 7 \tag{3.5}$$

$$f_2(x) = x^2 - 9 \tag{3.6}$$

Furthermore, by considering these two formulas as parents and performing crossover (as shown in Figure 3.6) we shall obtain the offspring formula:

$$f_3(x) = 1.4 * x^2 - 7 \tag{3.7}$$

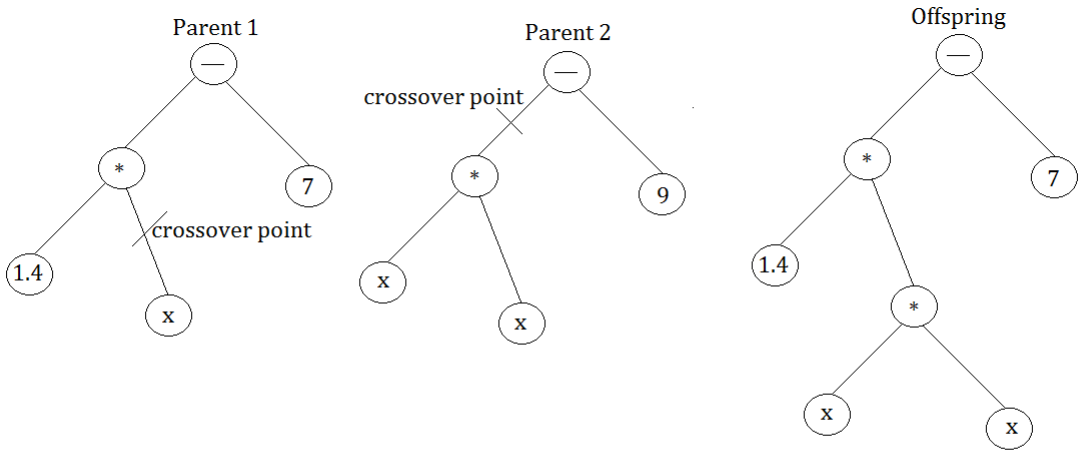

Figure 3.7: GP crossover of example regression formula

When applying f_1,f_2 and f_3 on X, each will generate a set of estimates: E_1, E_2 and E_3 respectively. These estimate values are shown in Table 3.2

E1	-12.6	-11.2	-9.8	-8.4	-7	-5.6	-4.2	-2.8	-1.4
E2	7	0	-5	-8	-9	-8	-5	0	7
E3	15.4	5.6	-1.4	-5.6	-7	-5.6	-1.4	5.6	15.4

Table 3.2: Estimations produced by GP models for example experiment

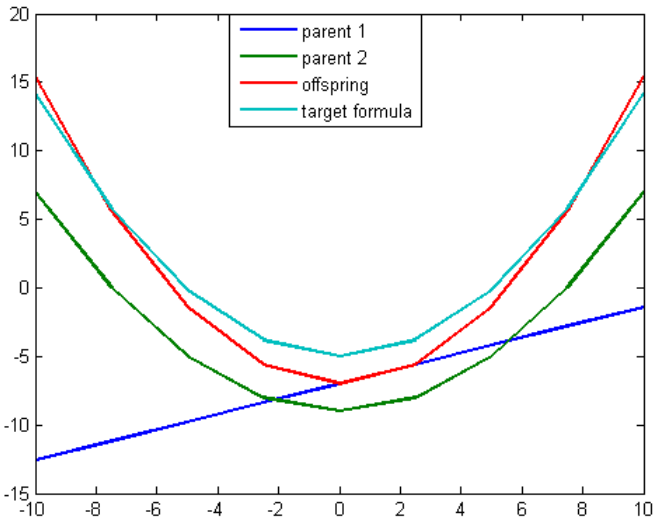

Figure 3.8: Plot of regression formula used as an example

The task of genetic programming in symbolic regression is to find the combination of primitives, input variables and coefficients that can provide estimation values that are as close as possible to the desired target values. Hence we use root mean square error (RMSE) function as our fitness function. The root mean squared error of two vectors A and B, each containing n values is given by

$$RMSE(A,B) = \sqrt{\frac{\sum_{i=1}^{n}(A_i - B_i)^2}{n}}$$

Obviously, when using RMSE as a fitness estimator in GP symbolic regression, the goal is to find a solution candidate with a fitness value as small as possible. When calculating the fitness of the considered candidate solutions, f_1, f_2 and f_3 as RMSE(Y,E_1), RMSE(Y,E_2) and RMSE(Y,E_3) respectively, we obtained:

$$RMSE(Y, E_1) = 12.74$$

$$RMSE(Y, E_2) = 5.85$$

$$RMSE(Y, E_3) = 1.62$$

As one might have expected by analyzing the plot in figure 3.8, from the three possible candidates, function f_3(the offspring formula) is the most precise estimator of the target regression function.

3.1.11 Multigene Genetic Programming (MGGP)

In this section it is outlined how multigene individuals are created and iteratively evolved by the MGGP algorithm.MGGP algorithm is similar to a 'standard' GP algorithm except for modifications made to facilitate the crossover and mutation of multigene individuals.
In the first generation of the MGGP algorithm, a population of random individuals is generated. For each new individual, a tree representing each gene is randomly generated (subject to depth constraints), using the user's specified palette of building block functions

and the available M input variables $X_1, \ldots, X_M$ as well as ephemeral random constants (ERCs) which is optional. In the first generation the MGGP algorithm attempts to maximize diversity by ensuring that no individual contains duplicate genes. However, due to computational expense, this is not enforced for subsequent generations of evolved individuals.

Each individual is specified to contain (randomly) between 1 and G_{max} (Maximum genes). G_{max} is a parameter set by the user. When using MGGP for regression, a high G_{max} may capture more non-linear behavior but there is the risk of over-fitting the training data and creating models that contain complex terms that contribute little or nothing to the model's predictive performance. Conversely, setting G_{max} to 1 is equivalent to performing scaled symbolic regression(i.e. containing only a bias(offset) term b_o and a weighting/scaling term b_1).

Like 'standard' GP, at each generation, individuals are selected probabilistically for breeding (using regular or Pareto tournaments or mixture of both) as discussed earlier. Each tournament results in an individual being selected based on either its fitness or for Pareto tournaments, its fitness and its complexity.

In order to further illustrate the coding procedure and the genetic operators used for GP, a symbolic regression example will be used. Consider the problem of predicting the numeric value of an output variable, y, from two input variables x_1 and x_2. One possible symbolic

representation for y in terms of x_1 and x_2 would be

$$y = \frac{x_1 + x_2}{5}$$

.

Figure 3.9 demonstrates how this expression may be represented as a tree structure. With

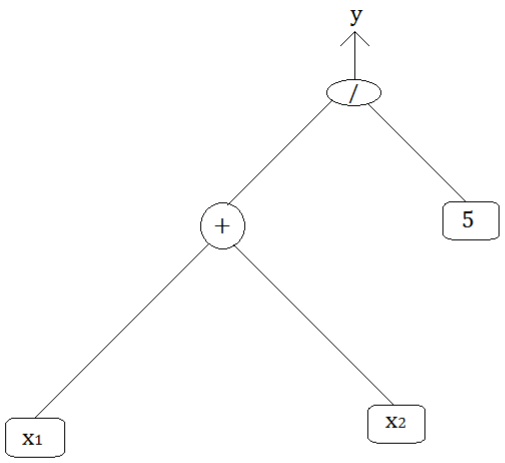

Figure 3.9: Representation of a numeric expression using tree structure

this tree representation, the genetic operators of crossover and mutation must be posed in a manner that allows the syntax of resulting expressions to be preserved. Figure 3.10 also illustrates a valid crossover operation where the two parent expressions are given in equations 3.8 and 3.9. Parent 1(y_1) and Parent 2 (y_2) are presented in equations 3.8 and 3.9. The developed offspring 1(y_3) and offspring 2 (y_4) are presented in equations 3.10 and

3.11 respectively.

$$y_1 = \frac{x_1 + x_2}{5} \tag{3.8}$$

$$y_2 = (x_3 - x_2) * (x_1 + x_3) \tag{3.9}$$

$$y_3 = \frac{x_1 + x_2}{x_3 - x_2} \tag{3.10}$$

$$y_4 = 5 * (x_1 + x_3) \tag{3.11}$$

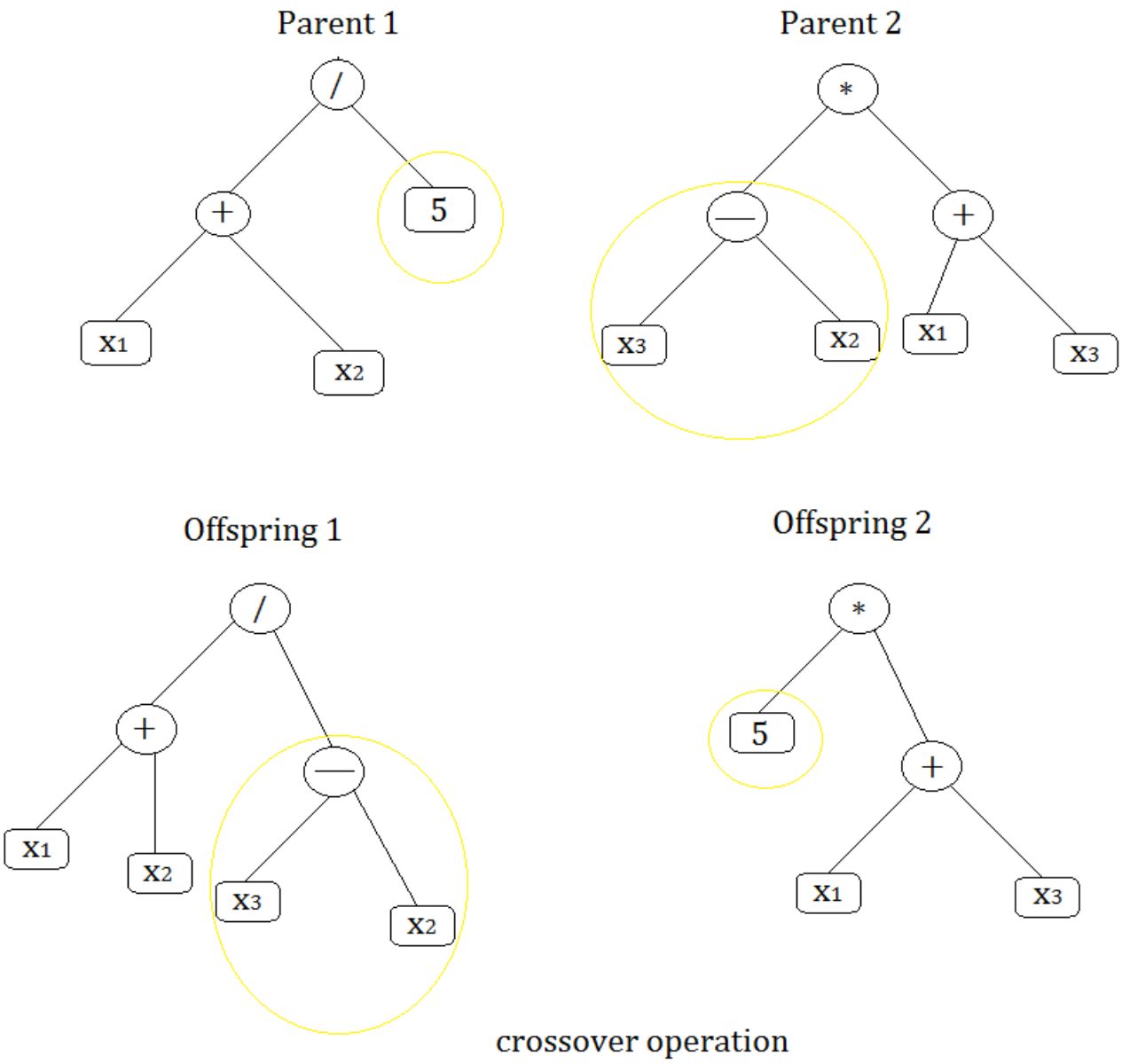

Figure 3.10: Valid crossover representation

Chapter 4

Data Collection and Analysis

In this chapter, we shall model the thyroid problem as a symbolic regression problem and consider the application of the GP to the problem. A life data from the UCI machine learning repository shall be examined.

Mathematically, our objective here is to design the classifier using GP ,i.e. given a set of training samples $X = \{x_1, x_2, x_3, , x_n\} \in S^n$ and its associated labels $Y = \{y_1, y_2, y_3\} =$ {normal, hyperthyroidism, hypothyroidism} then the objective can be described as finding the mapping $y = C(x)$ or $C : X \rightarrow Y$ using Genetic Programming. Where, C is the classifier that maps search space to label vectors, S^n is the 'n' dimensional search space.

4.0.12 Data Collection

The data used in this thesis was retrieved from the UCI Machine learning Repository.The data has 7200 instances and consists of twenty one (21) input variables and one (1) output variable which has a value '1', '2' or '3', where '1' means the person is normal, '2' implies the person is suffering from hyperthyroidism and '3' means the person is suffering from hypothyroidism.This database is owned by Ross Quinlan from the Garavan Institute in Sydney, Australia.Some of the data set was used as training set and some as test set to

validate our model. An example of the data is given in Figure 4.1.

GP input variables (terminal set) are given by Table 4.1 below. These consists of the information and tests conducted on a patient before the patient is diagnosed as being normal or suffering from thyroid disorder.

Table 4.1: GP terminal set

x_1	Age
x_2	Sex
x_3	On thyroxine
x_4	Query on thyroxine
x_5	On antithyroid medication
x_6	Sick
x_7	Pregnant
x_8	Thyroid surgery
x_9	I131 treatment
x_{10}	Query on hypothyroid
x_{11}	Query on hyperthyroid
x_{12}	Lithium
x_{13}	Goiter
x_{14}	Tumor
x_{15}	Hypopituitary
x_{16}	Psych
x_{17}	TSH
x_{18}	T3
x_{19}	TT4
x_{20}	T4U
x_{21}	FTI

Some of the inputs are thyroid function tests which help to check the function of the thyroid gland. They are mainly used to detect hypothyroidism (under active thyroid) and hyperthyroidism (overactive thyroid). These are: TSH (X_{17}), T3 (X_{18}), TT4 (X_{19}), T4U (X_{20}), FTI (X_{21}).

Total thyroxine (T T4) is generally elevated in hyperthyroidism and decreased in hypothy-

Figure 4.1: An example of the thyroid data

X1	0.73	0.5	0.65
X2	0	0	1
X3	1	1	0
X4	0	0	0
X5	0	0	0
X6	0	0	0
X7	0	0	0
X8	1	1	0
X9	0	0	0
X10	0	0	0
X11	0	0	0
X12	0	0	0
X13	0	0	0
X14	0	0	0
X15	0	0	0
X16	0	0	0
X17	0.006	0.061	0.01479
X18	0.015	0.0096	0.015
X19	0.12	0.013	0.061
X20	0.82	0.116	0.085
X21	0.146	0.011	0.072
y	3	2	1

roidism and is measured to see the bound and unbound levels of T4.The Free Thyroxine Index (FTI) is obtained by multiplying the total T4 with T uptake. FTI is elevated in hyperthyroidism and decreased in hypothyroidism. T3 is elevated in hyperthyroidism and depressed in hypothyroidism. The Thyroid Stimulating hormone(TSH) regulates the hormonal output from the thyroid. TSH rises whenever the thyroid gland fails to produce

sufficient thyroid hormones . Whenever there is too much thyroid hormones present in the serum, the brain responds by lowering TSH.

4.0.13 The Multigene Genetic programming (MGGP) Algorithm to the Thyroid Disorder

1. (Start) We input data file containing m x n matrix of input (predictor) variables, and m x 1 matrix of output (response) variables.

2. An initial population of candidate solutions are generated based on step one by using Ramped-half-and -half method (e.g. If maximum tree depth is 5 and population is 100 then, on average, 20 will be generated at each depth (1/2 using grow and 1/2 using full) after we had chosen our function and terminal sets. The maximum depth of each gene is then set to 7 to control the complexity of the model.

3. Also the number of genes(Gmax) is set to 8 as a high Gmax may capture more non-linear behavior but there is the risk of over-fitting the training data and creating models that contain complex terms that contribute little or nothing to the model's predictive performance. Conversely, setting Gmax to 1 is equivalent to performing scaled symbolic regression (where we have only the bias term b_0 and a scaling term b_1)

4. (Fitness) Our fitness measure is to minimize the error between the predicted values and the actual observed values , hence we then evaluated the fitness of each individual

solutions in the population by using:

$$RMSE = \sqrt{\frac{\sum_{p=1}^{p}(F_p - \tilde{F}_p)^2}{p}}$$

where,

F_p = actual values observed and $\tilde{F}_p$ = predicted values.

5. (New population) We create a new population by repeating the following steps until the new population is complete.

   (a) (Selection) We set a tournament size of 7 and apply Lexicographic Tournament pressure as our selection method to select the more fitter solutions into the next generation, i.e

$$\text{Lexicographic Tournament selection} = \text{pick}max\{F(G_1), F(G_2), \ldots, F(G_7)\}$$

$$\text{if} F(G_1) = F(G_2) = \cdots = F(G_7)$$

$$\text{pick}min\{S(G_1), S(G_2), \ldots, S(G_7)\}$$

   Where,

$$F(G_n) = \text{Fitness level of gene n}$$

$$S(G_n) = \text{Size of gene n}$$

(b) (Crossover) We then apply two point high level crossover with a probability of 0.85 to the selected individual solutions.

Consider two individuals P_1 and P_2 consisting of respective genes $[G_{1,1}, G_{2,1}, G_{3,1}]$ and $[G_{1,2}, G_{2,2}, G_{3,2}, G_{4,2}]$.

Two randomly selected crossover sections are shown below. The gene sections enclosed by the crossover points are denoted by $\{\dots\}$

$$P_1 = [G_{1,1}, \{G_{2,1}\}, G_{3,1}]$$

$$P_2 = [G_{1,2}, \{G_{2,2}, G_{3,2}, G_{4,2}\}]$$

The two crossover sections are exchanged resulting in the offspring O_1 and O_2

$$O_1 = [G_{1,1}, G_{2,2}, G_{3,2}, G_{4,2}, G_{3,1}]$$

$$O_2 = [G_{1,2}, G_{2,1}]$$

(c) (Mutation) A subtree mutation is applied with the probability of 0.1

(d) (Elitism) Also we apply the direct reproduction operator (elitism) with a probability of 0.05. i.e. the entire individual is simply copied to the next generation without modification.

(e) (Accepting) Place new offspring in the new population.

6. (Replace) We then use the new generated population for a further sum of the algorithm.

7. (Test) If the end condition is satisfied, stop and return the best solution in current population.

8. (Loop) Go to step 4 for fitness evaluation.

The two major control parameters in GP are the population size and the maximum number of generations to be run when no individual reaches the termination criterion. These two parameters depend on the difficulty of the problem to be solved. Generally, populations of 500 or more trees give better chances of finding a global optimum. For a small number of design variables, a starting population of 100 has proven to be sufficient. The maximum number of generations has been chosen to be 250 for this thesis and the initial population of size 1500.

4.0.14 The Developed Multigene Genetic Programming Model

GPTIPS, a Genetic Programming toolbox for use with MATLAB, framework was used to apply GP in the experiment designed in this research. The data set as shown in Figure 4.1 was loaded into GPTIPS framework ,Searson D.,(2009), then a symbolic regression via GP was applied with parameter set (preparatory steps) shown in Table 4.2. The cross validation was tuned to 60% for training and 40% for testing.

Initially the function pool that was used is

$$\{+, -, *, /, \sqrt{}, square, \sin, \cos, \arcsin, \arccos, log, abs, reciprocal, \tanh\}$$

After several runs with different combination of function sets, the function set that achieved best solution is shown in Table 4.2. And the best eight genes together with their weights are given in Figure 4.2 through to Figure 4.8 below.

Table 4.2: GP Preparatory steps

Population size	1500
Number of generation	250
Selection Method	Lexicographic tournament selection
Tournament size	7
Termination criteria	0
Maximum depth of each tree	7
Maximum Number of Genes	8
Function set	{+,-,*,tanh,cos,}
Crossover probability	0.85
Mutation probability	0.1
Elite probability	0.05

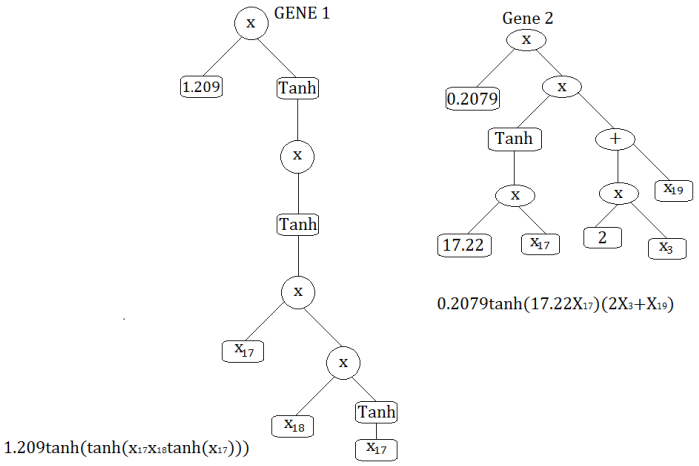

Figure 4.2: Gene 1 & 2

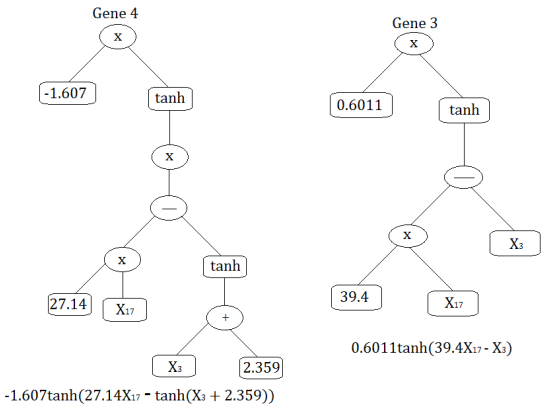

Figure 4.3: Gene 3 & 4

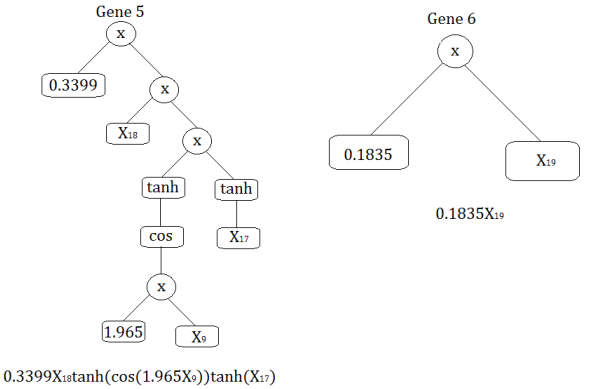

Figure 4.4: Gene 5 & 6

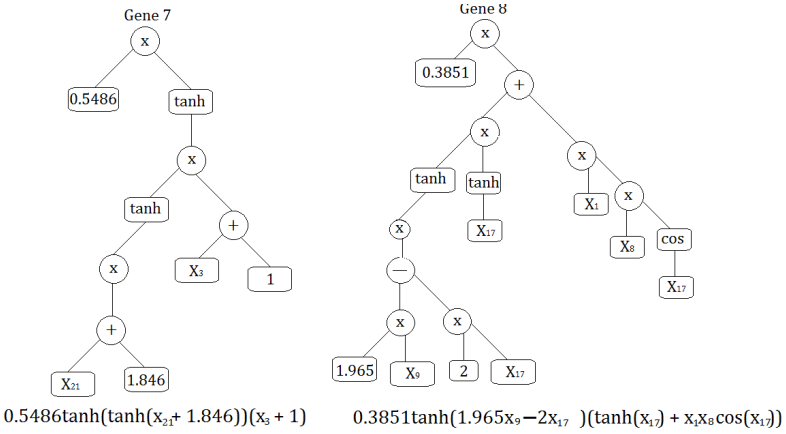

Figure 4.5: Gene 7 & 8

4.0.15 The Complete GP Model

After linear combination of all the eight (8) genes together with their weights, the over all simplified multigene symbolic regression model was harvested as follows:

$$
\begin{aligned}
y &= 0.1835x_{19} - 1.607tanh(27.14x_{17} - tanh(x_3 + 2.359)) \\
&+ 1.209tanh(tanh(x_{17}x_{18}tanh(x_{17}))) + 0.6011tanh(39.4x_{17} - x_3) \\
&+ 0.5486tanh(tanh(x_{21} + 1.846))(x_3 + tanh(x_{21} + 7.51)) \\
&+ 0.2079tanh(17.22x_{17})(2x_3 + x_{19}) \\
&+ 0.3851tanh(1.965x_9 - 2x_{17})(tanh(x_{17}) + x_1x_8cos(x_{17})) \\
&+ 0.3399x_{18}tanh(cos(1.965x_9))tanh(x_{17}) - 1.131
\end{aligned}
$$

4.0.16 Analysis of Results

We did statistical analysis of the developed GP model to test the goodness of fit of our model. From Figure 4.6, the RMSE for the training data set was 0.1254 and had 88.232% variation explained. The RMSE for the test data set was 0.1684 and had 79.1642% variation explained.

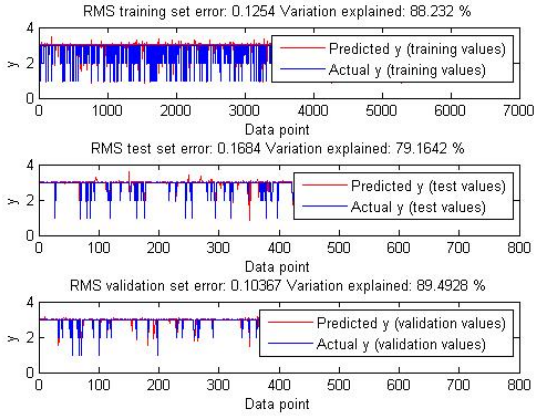

Figure 4.6: RMS error

4.0.17 Gene Weights, R^2 and Adjusted R^2

The model gave the coefficient of determination (R^2) = 0.88248 and the Adjusted R^2 = 0.88233. That is, 88% of variation in the response variable y are explained by the variation of the predictor variables x_i. This shows that there is a good correlation between the variable of interest (y) and the independent variables (x_i). The regression coefficients were estimated by the method of least squares described in chapter three and are shown in Figure 4.7 as Gene weights.

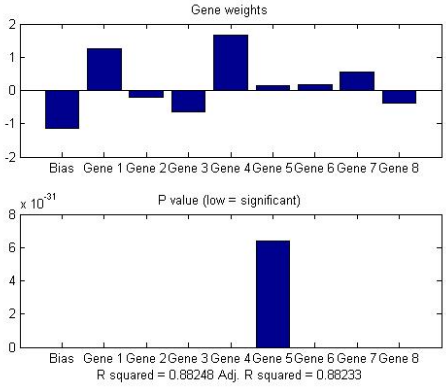

Figure 4.7: $R^2$and adjusted R^2

4.0.18 Scatter Diagram

We also plotted a scatter diagram to have a visual indication of the degree of the association between the response variables, that is the predicted ($\hat{y}$) and the actual (y). The line drawn passed through the points plotted for the output(i.e. Normal(1), Hyperthyroidism(2) and Hypothyroidism(3)) as shown in figure 4.8. This means that there is a positive association between the (y) and ($\hat{y}$).

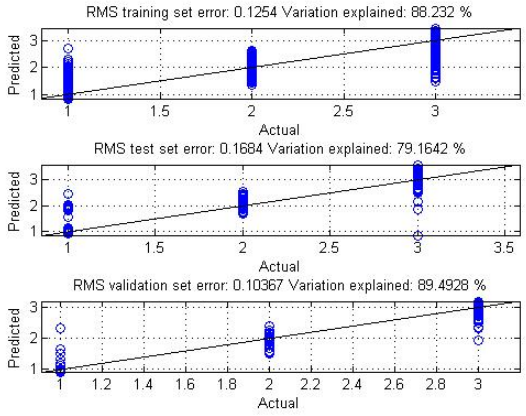

Figure 4.8: scatter diagram

4.0.19 Important attributes for thyroid disorder detection

The developed GP model used only eight (8) attributes out of the twenty one (21) to detect the thyroid disorder and these are x_1, x_3, x_8, x_9, x_{17}, x_{18}, x_{19}, x_{21}. Out of these TSH(x_{17}) was the most influential factor on the response variable followed by x_3(On thyroxine), x_{18}(T3), x_{21}(FTI), x_{19}(TT4), x_8(Thyroid surgery), x_9(I131 treatment) in that order. These are shown in figure 4.9 as histogram. The height of the bars corresponds to the most

influential attribute to the thyroid disorder.

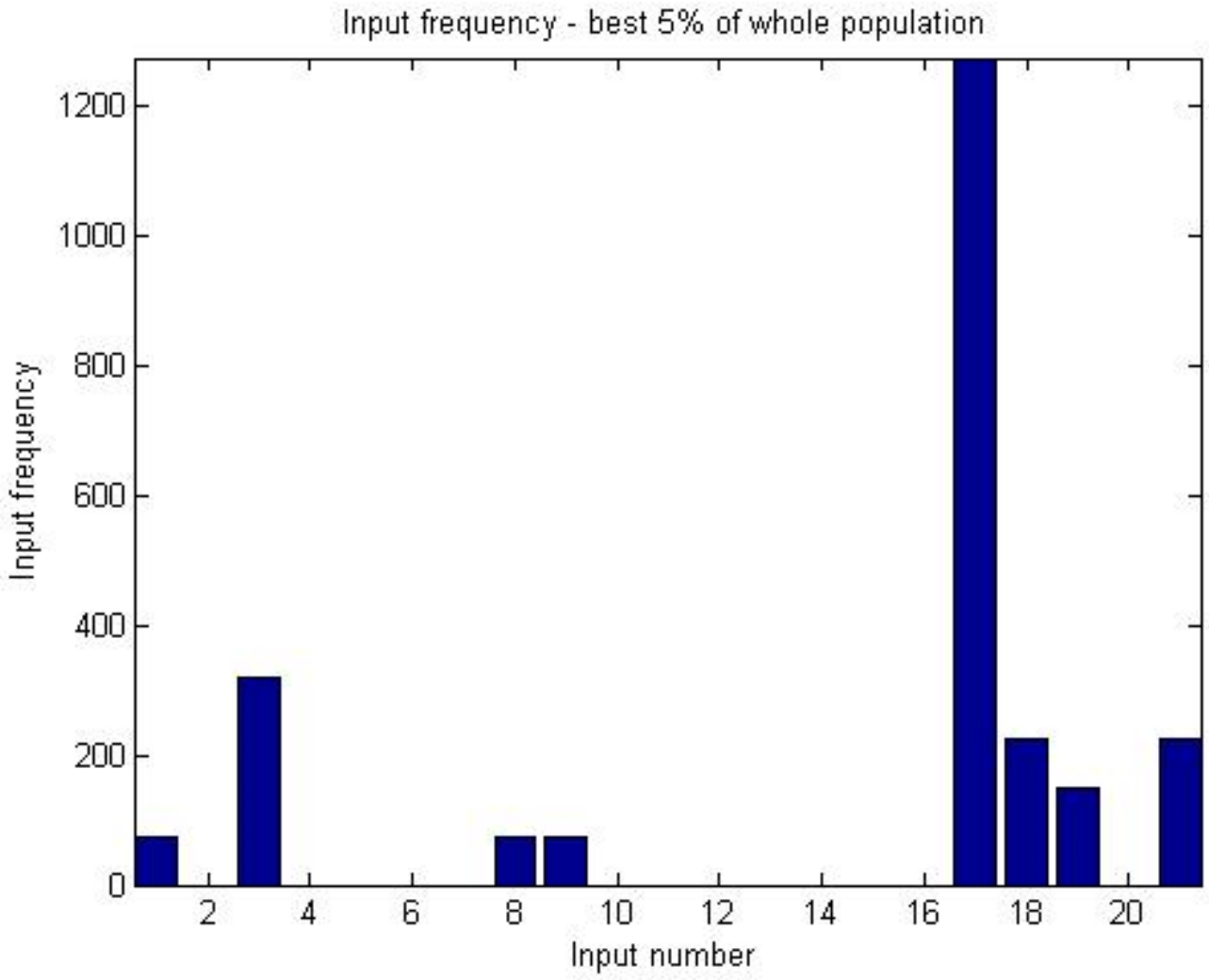

Figure 4.9: Best eight inputs

Chapter 5

Conclusion, and Recommendation

5.1 Conclusion

In this work, a MGGP mathematical model was developed to provide a solution to the thyroid problem.

The developed model is given as:

$$
\begin{aligned}
y &= 0.1835x_{19} - 1.607tanh(27.14x_{17} - tanh(x_3 + 2.359)) \\
&+ 1.209tanh(tanh(x_{17}x_{18}tanh(x_{17}))) + 0.6011tanh(39.4x_{17} - x_3) \\
&+ 0.5486tanh(tanh(x_{21} + 1.846))(x_3 + tanh(x_{21} + 7.51)) \\
&+ 0.2079tanh(17.22x_{17})(2x_3 + x_{19}) \\
&+ 0.3851tanh(1.965x_9 - 2x_{17})(tanh(x_{17}) + x_1x_8cos(x_{17})) \\
&+ 0.3399x_{18}tanh(cos(1.965x_9))tanh(x_{17}) - 1.131
\end{aligned}
$$

The MGGP used the most important attributes for the thyroid disorder detection and

these are x_{17}(TSH), x_{18}(T3), x_{21}(FTI), x_{19}(TT4), x_8(Thyroid surgery), x_9(I131 treatment). Root mean square error (RMSE), R^2 and Adjusted R^2, scatter diagram and histogram were used to ascertain the goodness of fit of the Model. Based on the statistical analysis, the model is found to be good for the thyroid disorder detection and is able to classify patient as normal (1), suffering from Hyperthyroidism (2) or Hypothyroidism (3). The developed MGGP model was able to reduce the attributes used in the classification. That's the MGGP model used eight (8) most important attributes instead of the whole 21 for the thyroid disorder detection.

5.1.1 Recommendation

The traditional method for detecting thyroid disorder, that is, the information and clinical tests conducted on the patient, is very cumbersome as it involves about twenty-one (21) attributes. Our MGGP model we developed uses only eight (8) attributes for prediction and it is very effective. We therefore recommend this MGGP model to all physicians for the detection of thyroid problems since it is economical in terms of time and money as compared to the traditional method.

References

Allen, F., and Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. in Journal of financial economics , 51, p. 245-271, 1999 .

Anderson, B., Svensson, P., Nordahl, M. and Nordin, P. (2000). On-line Evolution of Control for a Four-Legged Robot Using Genetic Programming In: Real World Applications of Evolutionary Computing(S. Cagnoni, et al., Eds), pp 319-326.Springer, Berlin, Germany.

Andre, D. (1994). Evolution of map making ability: Strategies for the evolution of learning,planning, and memory using genetic programming. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, volume 1. IEEE Press.

Andre, D. (1994). Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In KennethE. Kinnear,Jr., editor, Advances in Genetic Programming, chapter 23. MIT Press.

Andre, D. (1994). Learning and upgrading rules for an OCR system using genetic programming.In Proceedings of the 1994 IEEE World Congress on Computational Intelligence. IEEE Press.

Andrews, M., and Richard, P. (1994). Genetic programming for the acquisition of double auction market strategies. In Kenneth E. Kinnear, Jr., editor, Advances in Genetic Programming, chapter 16, pages 355-368. MIT Press.

Banzhaf, W. (1993). Genetic programming for pedestrians In proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93tr93-03.

Basim, A., H., Alaa Al-Afeef, and J. T. (2012). Symbolic regression of crop pest forecasting using genetic programming. Electrical Engineer and Computer Science,Vol.20,No.Sup.2.

Burke, E.K., M. H., and Kendall, G. (2006). Evolving bin packing with genetic programming.Proceedings of PPSN IX. Sprinfer-Verlag, pp.860-869 .

Carlos, M.V, C. R., Alberto, C., and Ventura, S. (2012). Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data. Springer Science+Business Media, LLC.

Cramer, N.L. (1985), A Representation for the Adaptive Generation of Simple Sequential Programs, Proceedings of an international conference on genetic algorithms and their applications July 24-26,1985 at Carnegie-Mellon University Pittsburgh, PA

Drake J. H, K. I., Mathew, H., and O'Zcan, E. (2009). A genetic programming hyperheuristic for the multidimensional knapsack problem.

Dufy, J., and Warnick J. E. (2010). Using symbolic regression to infer strategies from experimental data. University of Pittsburgh, Pittsburgh, PA 15260 USA.

Faris, H. (2013). A symbolic regression approach for modeling the temperature of metal cutting tool. International Journal of Control and Automation Vol. 6, No. 4.

Flanigan, J. (2004). Genetic programming, a general approach to solving specific problems.cis 203.

Guo, H. and Nandi, A. K. (2006). Breast cancer diagnosis using genetic programming generated feature. Pattern Recognition, vol. 39, pp. 980-987 .

Handley, S. (1993) Automatic learning of a detector for alpha-helices in protein sequences via genetic programming. In Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, pages 271 to 278. Morgan Kaufmann.

Iba, H., and Sasaki, T. (1999). Using genetic programming to predict financial data. CEC 99. Special session on time series prediction .

Jinjun, H. (2007). Finding an approximate analytic solution to differential equation by seed selection genetic programming. Vol.36, NO. 3.

Jonsson H., Madjid, P., and Mordal, N. (1997). Evolution of trading rules for fx market or how to make money out of gp. TRITA-PDC Report, ISRN KTH/PDC/R-97/5-SE. ISSN 1401-2731.

Karlsson R., Nodin, P., and Nordahl, M. (2000). Sound Localization for a Humanoid Robot by Means of Genetic Programming. In: Real World Applications of Evolutionary Computing pp 65-76. Springer, Berlin, Germany.

Kotanchek,M. E., E. Y. V., and Smits, G. F. (2012). Symbolic regression via gp as a discovery engine: Insights on outliers and prototypes.

Koza, J. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press Cambridge, MA, USA.

Kirk B,(2013). Singular value decomposition tutorial.

Koza, J. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs.

MIT Press, Cambridge, MA

Koza, J. R. (2007). Retrieved from www.geneticprogramming.com-HomePage

Koza,R. J. (1990). A genetic approach to econometric modeling. published at Sixth world congress of the Econometric Society, Barcelona, Spain.

Kumar, R., K. K. B., Joshi, A. H. and Rockett, P. I. (2008). Evolution of hyperheuristics for the biobjective 0/1knapsack problem by multiobjective genetic programming. In proceedings of GECCO 2008, 2008 PP. 1227-1234 .

Kulkarni, S.N., and Karwankar, A. R. (2012).Thyroid Disease Detection using Modified Fuzzy Hyperline Segment Clustering Neural Network. International Journal of Computers Volume 3 No. 3.

Langdon, W. B. (1995). Evolving data structures using genetic programming. Research Note RN/1/95, UCL, Gower Street, London, WC1E 6BT. Accepted for presentation at ICGA-95.

Luke, S., Hohn, C., Farris, F., Jackson, G., and Hendler, J. (1998). Co-evolving Soccer Softbot Team Coordination with Genetic Programming. In: RoboCup-97: Robot Soccer World Cup I (Lecture Notes in Artificial Intelligence No. 1395) (H. Kitano,ed).Pp 398-411. Springer-Verlag. Berlin, Germany.

Montana, D. J. (1994). Strongly typed genetic programming. BBN Technical report No. 7866, Bolt Beranek and Newman, Inc, 10 Moulton Street, Cambridge, MA 02138, USA.

Muhammed, A. P, Abdul, R. (2010). Design of classifier for detection of diabetes using

genetic programming. 18th European Signal Processing Conference(EUSIPCO).

Neshatian, K. (2010). Feature manipulation with genetic programming. Unpublished doctoral dissertation, Victoria University of Wellington.

Oakley, H. (1994). Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In Kenneth E. Kinnear, Jr., editor, Advances in Genetic Programming, chapter 17. MIT Press.

O'Neill M., R. C., Brabazon A., and J.J., C. (2001). Evolving market index trading rules using gramatical evolution. Lecture notes in computer science 2037, Genetic Programming, pp.343-351 .

Ozyimalz, L. and Yildirim, T. (2002). Diagnosis of thyroid disease using artificial neural network methodsin proceedings of iconip 9th international conference on neural information processing (singapore: Orchid country club,2002)pp.2033-2036.

Perry, J. E. (1994). The effect of population enrichment in genetic programming. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, pages 456 - 461.IEEE Press.

Poli R., Langdon W. B., and McPhee, N. F. A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

Poli, R. and Bader-El-Den M.(2007). '"generating sat local search heuristics using a gp hyper-heuristic framework in artificial evolution."'

Ready, J. M. (1997). Profits from technical trading rules. working paper University of

Wisconsin- Madison, Finance, Investment and Banking.

Rosca, J. P. and Ballard, D. H. (1994). Learning by adapting representations in genetic programming. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence. IEEE Press.

Santini, M., and Tettamanzi, A. (2001). Genetic programming for financial time series prediction. In EuroGP,01 Proceedings, Lecture Notes in Computer Science 2038, Genetic Programming, p. 361-371.

Searson, D. (2009). Gptips:genetic programming and symbolic regression for matlab. School of Computing Science, Newcastle University, UK.

Searson, D. P. (2009). Gptips giude:an open source software platform for symbolic data mining. School of Computing Science, Newcastle University, UK.

Sharief, A. A. and Sheta, A. (2014). Developing a mathematical model to detect diabetes using multigene genetic programming. (IJARAI) International Journal of Advanced Research in Artificial Intelligence, Vol. 3, No. 10, 2014 .

Sivananda, S.N and Deepa, S.N. (2008). Introduction to genetic algorithm. ISBN 978-3-540-73189-4 Springer Berlin Heidelberg New York.

Spencer, G.F. (1993). Automatic generation of programs for crawling and walking. In Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93,page 654, Stanford. Morgan Kaufmann.

Streeter, M. A., Koza,J. R., Keane,J. R., Mydlowec,M. J., Yu, J., and Lanza,G. (2003). Genetic Programming IV: Routine Human- Competitive Machine Intelligence. Kluwer

Academic Publisher.

Tackett, W. A. (1993). Genetic programming for feature discovery and image discrimination.In Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93. Morgan Kaufmann.

UCI. (1987). Thyroid disease data set. Retrieved from http://archive.ics.uci.edu/ml/dataset/thyroid+Disease.

Yoshihara, I., A. T., and M. Y. (2000). Financial application of time series prediction based on genetic programming. in Proceedings of the Genetic and Evolutionary Computation Conference p. 537 .

Zhang, C. (2012). genetic programming for symbolic regression. Department of Electrical Engineering and Computer Science, University of Tennesse, Knoxville, TN 37996, USA.

Zhao, K. and Wang, J. (2000). Multi-robot Cooperation and Competition with Genetic Programming In: EuroGP 2000 Proceedings. Pp. 50- 359. Springer-Verlag.Berlin, Germany.

Appendix

```
function [ gp ] = mygp()

function [gp] = func(gp)

gp.runcontrol.pop_size=1500;    % Population size

gp.runcontrol.num_gen=250;

gp.runcontrol.verbose=5;


% Selection method options

% -------------------------

gp.selection.tournament.size=7;

gp.selection.tournament.lex_pressure=true;


% Fitness function specification

%--------------------------------


gp.fitness.fitfun=@regressmulti_fitfun;

gp.fitness.minimisation=true;

gp.fitness.terminate=true;

gp.fitness.terminate_value=0 ;


% Set up user data
```

```
% ----------------

c = load('thyroid101.mat');


%allocate to train, validation and test groups


thyroid101tra = dat.thyroid101tra;

thyroid101tst = dat.thyroid101tst;


%allocate to train, validation and test groups

gp.userdata.xtrain=thyroid101tra(:,1:21);

gp.userdata.ytrain=thyroid101tra(:,22);

gp.userdata.xtest=thyroid101tst(:,1:21);

gp.userdata.ytest=thyroid101tst(:,22);


%scale data to zero mean and unit variance

% gp=gpscale(gp);

%-------------------------------------------------------------------

gp.userdata.scale = true;

%get data

xtrain=gp.userdata.xtrain;

ytrain = gp.userdata.ytrain;
```

```
% check that test data is present
if (~isfield(gp.userdata,'xtest')) || (~isfield(gp.userdata,'ytest')) || ...
isempty(gp.userdata.xtest) || isempty(gp.userdata.ytest)
usetest=false;
else
usetest=true;
xtest = gp.userdata.xtest;
ytest = gp.userdata.ytest;
end
% check that validation data is present
if (~isfield(gp.userdata,'xval')) || (~isfield(gp.userdata,'yval')) || ...
isempty(gp.userdata.xval) || isempty(gp.userdata.yval)
useval=false;
else
useval=true;
xval = gp.userdata.xval;
yval = gp.userdata.yval;
end
% First try to scale input training data
[xtrainS,mux,sigmax]=scalematrix(xtrain);
```

```
%check & get rid of any columns with zero variance/sdev
zv_ind=find(sigmax==0);
xtrain(:,zv_ind)=[];
if usetest
xtest(:,zv_ind)=[];
end
if useval
xval(:,zv_ind)=[];
end

[gp.userdata.xtrainS,mux,sigmax]=scalematrix(xtrain);
if usetest
gp.userdata.xtestS = (xtest-repmat(mux,size(xtest,1),1))./repmat(sigmax,size(xtest,1),1)
end
if useval
gp.userdata.xvalS = (xval-repmat(mux,size(xval,1),1))./repmat(sigmax,size(xval,1),1);
end

%Scale outputs
[gp.userdata.ytrainS,muy,sigmay]=scalematrix(ytrain);
if usetest
```

```
gp.userdata.ytestS = (ytest-repmat(muy,size(ytest,1),1))./repmat(sigmay,size(ytest,1),1)
end
if useval
gp.userdata.yvalS = (yval-repmat(muy,size(yval,1),1))./repmat(sigmay,size(yval,1),1);
end


gp.userdata.mux = mux;
gp.userdata.sigmax = sigmax;
gp.userdata.muy = muy;
gp.userdata.sigmay = sigmay;


if ~isempty(zv_ind)
disp('GPSCALE warning: The following columns of the scaled input data
have been removed because they have zero variance: ');
disp(num2str(zv_ind));
gp.userdata.removed_scaled_xcolumns=zv_ind;
else
gp.userdata.removed_scaled_xcolumns=[];
end


function [z,mu,sigma]=scalematrix(x)
```

```
mu =mean(x);

sigma=std(x);


z = x-repmat(mu,size(x,1),1);

z=   z ./(repmat(sigma,size(x,1),1));

end

%--------------------------------------------------------------------


gp.userdata.datasampling = true;

gp.userdata.user_fcn=@regressmulti_fitfun_validate;


% Input configuration

% -------------------


gp.nodes.inputs.num_inp=size(gp.userdata.xtrain,2);


% Tree build options

% ------------------

gp.treedef.max_depth=5;                % Maximum depth of trees
```

```
% Multiple gene settings
% ----------------------
gp.genes.multigene=true;               % Set to true to use
multigene individuals and false to use ordinary single gene individuals.

gp.genes.max_genes=8;            % The absolute maximum number
of genes allowed in an individual.

% Define functions
% ----------------
%
%   (Below are some definitions of functions that have been used
for symbolic regression problems)
%
%         Function name                              Number of arguments
%   (must be an mfile on the path)

gp.nodes.functions.name{1}='times'      ;
gp.nodes.functions.name{2}='minus'      ;
gp.nodes.functions.name{3}='plus'       ;
gp.nodes.functions.name{4}='rdivide'    ; % unprotected divide (may cause NaNs)
```

```
gp.nodes.functions.name{5}='psqroot'    ; % protected sqrt

gp.nodes.functions.name{6}='plog'       ; % protected natural log

gp.nodes.functions.name{7}='square'     ; % .^2 square

gp.nodes.functions.name{8}='tanh'       ; % tanh function

gp.nodes.functions.name{9}='pdivide'   ;  % protected divide function

gp.nodes.functions.name{10}='iflte'     ; % IF-THEN-ELSE function

gp.nodes.functions.name{11}='sin'       ;

gp.nodes.functions.name{12}='cos'       ;

gp.nodes.functions.name{13}='exp'       ;


% Active functions

% ----------------

% Manually setting a function node to inactive allows you to

exclude a function node in a particular run.

gp.nodes.functions.active(1)=1;

gp.nodes.functions.active(2)=1;

gp.nodes.functions.active(3)=1;

gp.nodes.functions.active(4)=0;

gp.nodes.functions.active(5)=0;

gp.nodes.functions.active(6)=0;

gp.nodes.functions.active(7)=0;
```

```
gp.nodes.functions.active(8)=1;

gp.nodes.functions.active(9)=0;

gp.nodes.functions.active(10)=0;

gp.nodes.functions.active(11)=0;

gp.nodes.functions.active(12)=1;

gp.nodes.functions.active(13)=0;

end

%----------------------- gp = rungp('thyroiddemo.m')

% if nargin<1

%     disp('Cannot execute. To run GPTIPS a configuration m file must be ');

%     disp('specified, e.g. gp=rungp(''quartic_poly'') ');

%     return;

% end

% generate prepared data structure with some default parameter values

% gp=gpdefaults();

gp.runcontrol.about='Main run control parameters';

gp.runcontrol.pop_size=1500;

gp.runcontrol.num_gen=250;

gp.runcontrol.verbose=5;    % The generation frequency with which

 results are printed to CLI

gp.runcontrol.savefreq=50;
```

```
gp.runcontrol=orderfields(gp.runcontrol);

gp.runcontrol.quiet=false; %if true, then GPTIPS runs with no

CLI output

gp.selection.about='Selection method parameters';

gp.selection.method='tour';

gp.selection.tournament.size=2;

gp.selection.tournament.lex_pressure=true; % Set to true to use

Sean Luke's lexographic selection pressure during tournament selection

gp.selection.elite_fraction=0.05;

gp.selection=orderfields(gp.selection);

gp.fitness.minimisation=true;                % Set to true if

you want to minimise the fitness function (if zero it is maximised).

gp.fitness.fitfun='A handle to your fitness function should go here';

gp.fitness.about='Fitness/objective function configuration';

gp.fitness.terminate=false;

gp.fitness.terminate_value=0;

gp.fitness=orderfields(gp.fitness);

gp.userdata=[];

gp.userdata.datasampling=false;

gp.userdata.scale=false;

gp.userdata.user_fcn=[];
```

```
gp.treedef.about='Tree building parameters and constraints';

gp.treedef.max_depth=6;

gp.treedef.max_mutate_depth=6;

gp.treedef.build_method=3;   %ramped half and half

gp.treedef.max_nodes=Inf;

gp.treedef=orderfields(gp.treedef);

gp.operators.about='Genetic operator configuration';

gp.operators.mutation.p_mutate=0.1;

gp.operators.crossover.p_cross=0.85;

gp.operators.directrepro.p_direct=0.05;

gp.operators.mutation.mutate_par=[0.9 0.05 0.05 0 0 0];

gp.operators.mutation.gaussian.std_dev=0.1;  % if mutate_type 3 (constant perturbation)

gp.operators.mutation=orderfields(gp.operators.mutation);

gp.operators=orderfields(gp.operators);

gp.nodes.about='Node configuration parameters';

gp.nodes.functions.about='Function node configuration';

gp.nodes.functions.name={'A cell array containing the names of your function nodes shoul

gp.nodes.functions.arity=[];

gp.nodes.functions.active=[];

gp.nodes.functions=orderfields(gp.nodes.functions);
```

```
gp.nodes.const.about='Ephemeral random constant configuration';

gp.nodes.const.use_matlab_format=false;

gp.nodes.const.num_dec_places=6;

gp.nodes.const.range=[-10 10];

gp.nodes.const.p_ERC=0.2;

gp.nodes.const=orderfields(gp.nodes.const);


gp.nodes.inputs.num_inp=[];

gp.nodes=orderfields(gp.nodes);


gp.genes.about='Multigene configuration';

gp.genes.multigene=true;

gp.genes.max_genes=8;

gp.genes.operators.p_cross_hi=0.2;

gp.genes=orderfields(gp.genes);


% ************************************************************************


% run user configuration file

gp=feval(@func,gp);

% perform error checks
```

```
gp=gpcheck(gp);

% perform initialisation

gp=gpinit(gp);

% Set and store the the PRNG seed

gp.info.PRNGseed=sum(100*clock);

rand('twister',gp.info.PRNGseed);

% main generation loop

for count= 1:gp.runcontrol.num_gen

if count==1;

% generate the initial population

gp=initbuild(gp);

% calculate fitnesses of population members

gp=evalfitness(gp);

% update run statistics

gp=updatestats(gp);

% call user defined function (if defined)

gp=gp_userfcn(gp);

% display current stats on screen

displaystats(gp);

else

% use crossover, mutation etc. to generate a new population
```

```
gp=popbuild(gp);

% calculate fitnesses of population members

gp=evalfitness(gp);

% update run statistics

gp=updatestats(gp);

% call user defined function

gp=gp_userfcn(gp);

% display current stats on screen

displaystats(gp);

end;

%save gp structure

if  gp.runcontrol.savefreq && ~mod(gp.state.count-1,gp.runcontrol.savefreq)

save gptips_tmp gp

end

% break out of generation loop if termination required

if gp.state.terminate

disp('Fitness criterion met. Terminating run.');

break;

end

end;  %end generation loop

% finalise the run
```

```
gp=gpfinalise(gp);

disp('Evaluating the best holdout validation individual of');

disp('the run on the fitness function using:');

disp('>>runtree(gp,''valbest'');');

runtree(gp,'valbest');

if license('test','symbolic_toolbox')

disp(' ');

disp('using the the GPPRETTY command on the best holdout validation individual: ');

disp(' ');

gppretty(gp,'valbest');

end

disp('frequency distribution of variables in the top 5\% of');

disp('the final population use: ');

gppopvars(gp);

end
```