ADAPTIVE LENGTH GENE EXPRESSION PROGRAMMING

Dissertation

By:

SAMSUL AMAR

17/420462/STK/00658

MECHANICAL ENGINEERING DOCTORAL PROGRAM

DEPARTMENT OF MECHANICAL AND INDUSTRIAL ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2024

GENE EXPRESSION PROGRAMMING DENGAN PANJANG ADAPTIF

Disertasi untuk memperoleh gelar Doktor Ilmu Teknik Mesin

SAMSUL AMAR

17/420462/STK/00658

PROGRAM S3 ILMU TEKNIK MESIN

DEPARTMENT TEKNIK MESIN DAN INDUSTRI

FAKULTAS TEKNIK

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2024

DISSERTATION REPORT APPROVAL

Adaptive Length Gene Expression Programming

By:

Samsul Amar

17/420462/STK/00658

Has been approved for the final examination

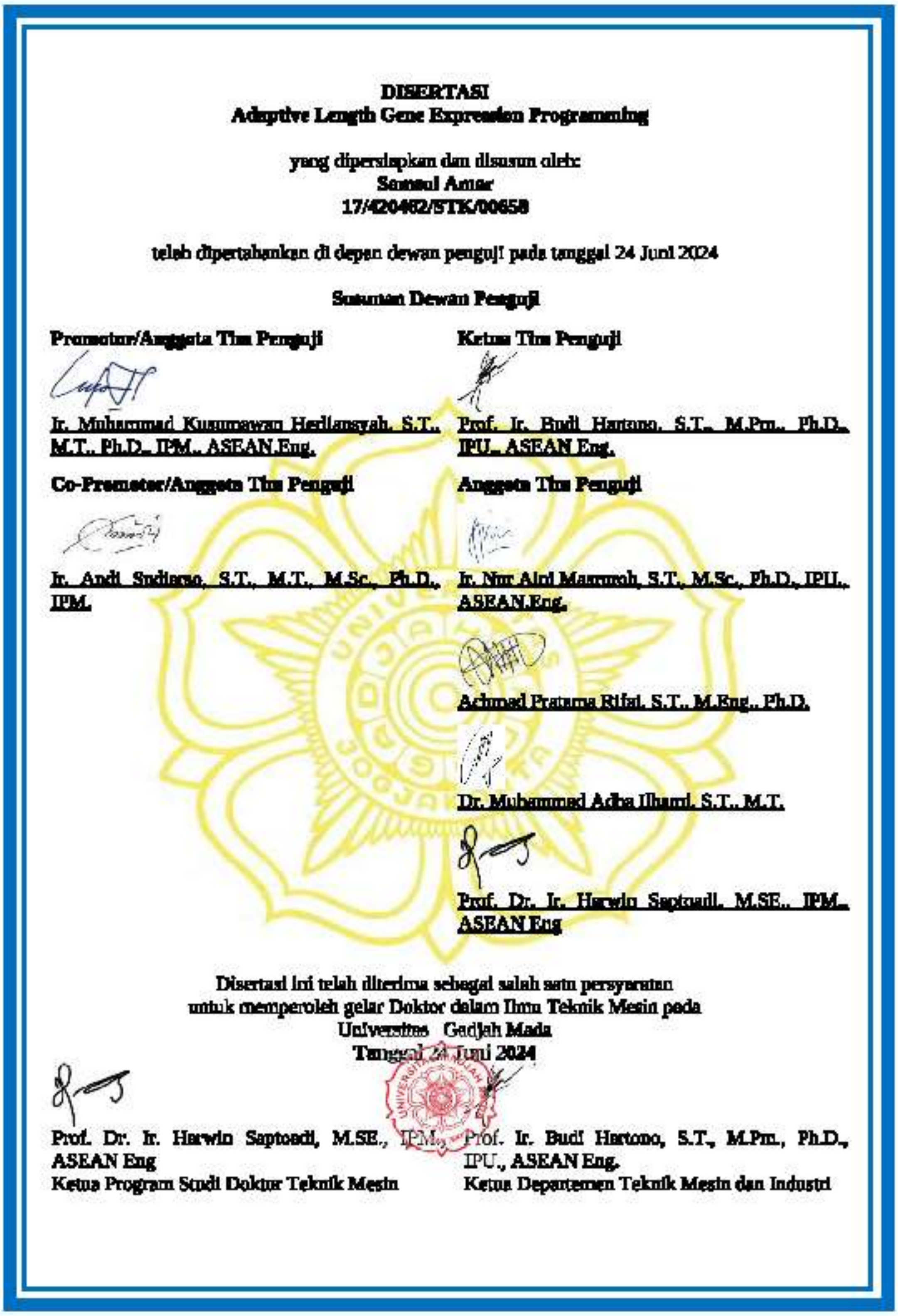
Approved by:

Muhammad K. Herliansyah, S.T., M.T., Ph.D. [illegible] on 19 Februari 2024

Promotor

Andi Sudiarso, S.T., M.T., M.Sc., Ph.D. [illegible] on 19 February 2024

Co-Promotor



DISERTASI
Adaptive Length Gene Expression Programming

yang dipersiapkan dan disusun oleh:
Samsul Amar
17/420482/STK/00658

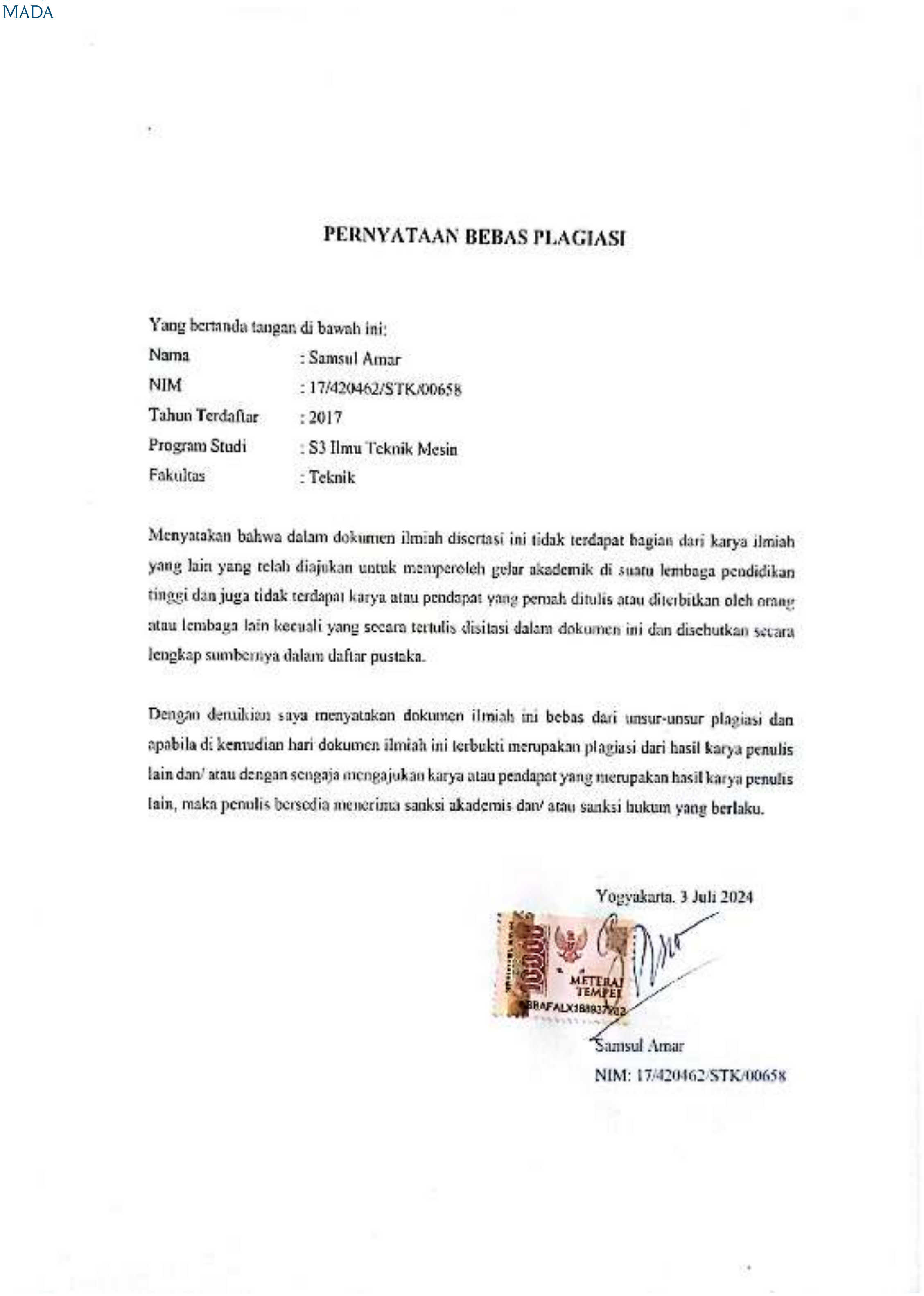
telah dipertahankan di depan dewan penguji pada tanggal 24 Juni 2024

Susunan Dewan Penguji

Promotor/Anggota Tim Penguji	**Ketua Tim Penguji**
[illegible]	[illegible]
Ir. Muhammad Kusumawan Herliansyah, S.T., M.T., Ph.D., IPM., ASEAN.Eng.	Prof. Ir. Budi Hartono, S.T., M.Pm., Ph.D., IPU., ASEAN.Eng.
Co-Promotor/Anggota Tim Penguji	**Anggota Tim Penguji**
[illegible]	[illegible]
Ir. Andi Sudiarso, S.T., M.T., M.Sc., Ph.D., IPM.	Ir. Nur Aini Masruroh, S.T., M.Sc., Ph.D., IPU., ASEAN.Eng.
	[illegible]
	Achmad Pratama Rifai, S.T., M.Eng., Ph.D.
	[illegible]
	Dr. Muhammad Adha Ilhami, S.T., M.T.
	[illegible]
	Prof. Dr. Ir. Harwin Saptoadi, M.SE., IPM., ASEAN Eng

Disertasi ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar Doktor dalam Ilmu Teknik Mesin pada
Universitas Gadjah Mada
Tanggal 24 Juni 2024

[illegible]	[illegible]
Prof. Dr. Ir. Harwin Saptoadi, M.SE., IPM., ASEAN Eng	Prof. Ir. Budi Hartono, S.T., M.Pm., Ph.D., IPU., ASEAN Eng.
Ketua Program Studi Doktor Teknik Mesin	Ketua Departemen Teknik Mesin dan Industri



PERNYATAAN BEBAS PLAGIASI

Yang bertanda tangan di bawah ini:

Nama	: Samsul Amar
NIM	: 17/420462/STK/00658
Tahun Terdaftar	: 2017
Program Studi	: S3 Ilmu Teknik Mesin
Fakultas	: Teknik

Menyatakan bahwa dalam dokumen ilmiah disertasi ini tidak terdapat bagian dari karya ilmiah yang lain yang telah diajukan untuk memperoleh gelar akademik di suatu lembaga pendidikan tinggi dan juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang atau lembaga lain kecuali yang secara tertulis disitasi dalam dokumen ini dan disebutkan secara lengkap sumbernya dalam daftar pustaka.

Dengan demikian saya menyatakan dokumen ilmiah ini bebas dari unsur-unsur plagiasi dan apabila di kemudian hari dokumen ilmiah ini terbukti merupakan plagiasi dari hasil karya penulis lain dan/ atau dengan sengaja mengajukan karya atau pendapat yang merupakan hasil karya penulis lain, maka penulis bersedia menerima sanksi akademis dan/ atau sanksi hukum yang berlaku.

Yogyakarta, 3 Juli 2024

10000
METERAI TEMPEL
BBAFALX181937702

[illegible]

Samsul Amar
NIM: 17/420462/STK/00658

Table of Contents

List of Tables

Table of Figures

List of Abbreviations

Ada-GEP	Adaptive Gene Expression Programming
ADF	Auto Defined Function
ALGEP	Adaptive Length Gene Expression Programming
ANN	Artificial Neural Network
BBRI	Code for Bank BRI stock in Indonesian Stock Market
BP	Back Propagation
DNA	Deoxyribo Nucleic Acid
EA	Evolutionary Algorithm
EMH	Efficient Market Hypothesis
GA	Genetic Algorithm
GEP	Gene Expression Programming
GP	Genetic Programming
IDX	Indonesian Stock Exchange
IHSG	Indeks Harga Saham Gabungan (Indonesian Composite Index)
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
NMSE	Normalized Mean Square Error
P-GEP	Prefix Gene Expression Programming
RGEP	Robust Gene Expression Programming
RNA	Ribo Nucleic Acid
RNC	Random Numerical Constants
RW	Roulette wheel
SCM	Supply Chain Management
SL-GEP	Self-Learning Gene Expression Programming
SRP	Symbolic Regression Problem

ABSTRACT

Gene expression programming (GEP) is capable of solving many prediction, classification, and optimization problem effectively. It uses a fixed-length chromosome representing a set of equations. A chromosome may contains one or more genes and the gene and chromosome length significantly affects the algorithm’s performance. Different problems may require varied gene/chromosome lengths to achieve good results and user have to do trial and error to get best length. There is no study found to overcome gene length problem in GEP. Therefore, this study aimed to develop an adaptive GEP to find proper gene length during the evolutionary process, called adaptive length GEP (ALGEP).

In ALGEP, the gene length may be varied for each individual instead of using the same length. The evolutionary process would adjust the gene length and the chromosome with proper gene length will tend to survive. Furthermore, the study proposed a length adjustment operator that could delete or insert an allele in the chromosome to make it short or extended. This operator is expected to adjust the gene length to its optimal. A special slice crossover was also proposed to accommodate the crossover between parents with different gene lengths. The proposed algorithms' performance was investigated by solving three symbolic regression problems and the performance was compared to related previous gene expression programming algorithms which are original GEP and Robust GEP (RGEP). A constant creation method and a constant mutation operator were also proposed to deal with complex constants those may occur in the problems. In addition, the algorithm will be applied to solve a complex real problem that is stock market prediction problem. The performance of the algorithm in predicting Indonesian Stock Exchange Composite Index and an individual stock price will be compared with another well-known algorithm, deep learning artificial neural network.

The result show that ALGEP performs better than GEP and RGEP for solving the SRPs. Furthermore, the use of slice crossover in GEP make the algorithm perform significantly better for solving the SRP’s; however, the use of length adjustment operator does not have significant impact. Finally the performance of ALGEP is significantly better for predicting a stock index and a stock price compared to artificial neural network.

Keyword: *gene expression programming, parameter setting, adaptive length, stock market prediction, artificial neural network.*

ABSTRAK

Gene expression programming (GEP) banyak digunakan untuk melakukan prediksi, klasifikasi, dan optimasi secara efektif. GEP biasanya menggunakan kromosom dengan panjang tertentu Dimana kromosom tersebut mewakili suatu program atau persamaan tertentu. Satu kromosom bisa terdiri atas satu atau lebih gen, Dimana panjang gen secara signifikan mempengaruhi kinerja algoritma. Masalah yang berbeda mungkin memerlukan panjang kromosom yang berbeda untuk mencapai hasil yang baik dan pengguna harus melakukan trial and error untuk mendapatkan panjang terbaik. Belum ditemukan penelitian yang dilakukan untuk menangani masalah panjang gen pada GEP tersebut. Oleh karena itu, penelitian ini bertujuan untuk mengembangkan GEP dengan panjang gen yang adaptif, dimana panjang gen yang tepat akan secara otomatis didapatkan selama proses evolusi.

Untuk mengatasi masalah panjang gen/ kromosom, penelitian ini mengusulkan panjang gen yang bervariasi untuk setiap individu dalam satu populasi. Proses evolusi akan menyesuaikan panjang gen dan kromosom dengan panjang gen yang sesuai akan cenderung bertahan. Lebih lanjut, penelitian ini mengusulkan *length adjustment operator* yang dapat menghapus atau menambah satu alel (bit) ke dalam gen untuk menjadikan lebih pendek atau lebih panjang. Operator ini diharapkan dapat mengatur panjang gen hingga tercapai panjang optimal. Sebuah operator *crossover* yaitu *slice crossover* juga diusulkan untuk mengakomodasi persilangan antara dua individu dengan panjang gen/ kromosom yang berbeda. Kinerja algoritma yang diusulkan akan diuji untuk menyelesaikan tiga *symbolic regression problem* lalu kinerjanya akan dibandingkan dengan algoritma *gene expression programming* yang terkait yaitu GEP dan Robust GEP (RGEP). Sebuah metode pembangkitan konstanta dan operator mutasi konstanta juga diusulkan untuk menangani konstanta yang lebih kompleks yang mungkin diperlukan dalam permasalahan. Selain itu, algoritma ini juga akan diterapkan untuk menyelesaikan permasalahan nyata yang kompleks yaitu permasalahan prediksi pasar saham. Kinerja algoritma dalam memprediksi Indeks Harga Saham Gabungan (IHSG) dan harga saham individu akan dibandingkan dengan algoritma terkenal lainnya, yaitu jaringan syaraf tiruan.

Hasil penelitian menunjukkan bahwa ALGEP berkinerja lebih baik dibandingkan GEP dan RGEP dalam menyelesaikan SRP. Selain itu, penggunaan *slice crossover* di GEP membuat algoritme bekerja lebih baik secara signifikan dalam menyelesaikan SRP; Namun penggunaan *length adjustment operator* tidak memberikan dampak yang signifikan. Terakhir, kinerja ALGEP secara signifikan lebih baik dalam memprediksi indeks saham dan harga saham dibandingkan dengan jaringan syaraf tiruan.

Kata kunci: *gene expression programming, setting parameter, panjang gene adaptif, prediksi pasar saham, jaringan syaraf tiruan.*

1. INTRODUCTION

1.1. Background

Prediction and classification are two areas that play an important role in industries. Some of the activities in the prediction area include demand forecasting, machine breakdown prediction, market share prediction, raw material price prediction, stock price prediction and market trend prediction. Product recommendations to consumers and fraud prediction are also examples of prediction areas. In the classification area, identification of defective products, consumer grouping, image recognition, and part classification are often used in industries. Popular machine learning methods have been applied for prediction and classification, including artificial neural networks, k-nearest neighbors, decision trees, and ada-boost. Although these methods could produce good prediction accuracy, they create a black box problem (Bathaee, 2018; Guidotti et al., 2018). The methods could probably provide a better prediction or classification than humans, but they cannot communicate or explain the reason underlying that decision. This could make the decision maker reject applying machine learning or cause tracing difficulties. Also, it could lead to law problems regarding the decision guided by machine learning methods.

The black box problem could be avoided using an algorithm called gene expression programming (GEP). This algorithm has been applied successfully to solve many real problems effectively (Candida Ferreira, 2006; Zhong et al., 2017). GEP is a search algorithm inspired by the natural or biological mechanism of adaptation and survival. In contrast of many machine learning methods, GEP produces explicit mathematical equations or model explanations that enables users to get the answer and know how the variables are interconnected.

GEP adopts the way genes work in organisms in an organism. It uses and denotes a fixed-length chromosome into an expression tree representing a set of equations or a specific model. A chromosome consists of one or more genes. Fix length string chromosome is easy to manipulate using various evolutionary operators. The GEP procedure is similar to the genetic algorithm processes, including initialization, fitness evaluation, selection and replication, mutation, and crossover. The chromosome design makes the decoding process always produce valid equations. This is one advantage of GEP compared with genetic programming, where an invalid function is probably produced in a generic operation. Also, the GEP's expression tree representation is better than the genetic algorithm in solving complex models.

GEP was developed by Fereira in 1999 and has been implemented or improved by many studies (Zhong et al., 2017). For instance, Mwaura and Keedwell (2009) developed a heuristic method to adjust the mutation and crossover rate during the evolutionary process to avoid an unfit rate-setting problem. Some studies combine GEP with other metaheuristic methods to increase the algorithm's performance. Zhang and Xiao (2010) and Zuo et al. (2004) combined GEP with differential evolution and found that their proposed algorithm performed better than the original GEP. Yang and Ma (2016) used an orthogonal design for a multiple-parent crossover operator in chromosome reproduction. The study also introduced an evolutionary stable strategy to maintain population diversity during evolution. Furthermore, Fajfar and Tuma (2018) developed a special numeric crossover operator to improve the constant creation ability of RGEP. Mehr (2018) combined GEP with GA for streamflow forecasting. The study refined the best individual from the GEP algorithm by GA to get better fitness. Similarly, Deng et al. (2018) combined GEP with an artificial fish swarm algorithm and found that the combination performed better than the original GEP. Yang et al. (2018) developed a hybrid Kalman filter GEP to reconstruct drawing and handwriting.

GEP, with its developments, are capable of solving various problems. However, some problems still need to be undertaken (Zhong et al., 2017), such as the encoding design. The basic encoding of GEP adopts K-expression to represent the computer program in a fixed-length string. One of the K-expression disadvantages is that a good building block could be easily destroyed in the genetic operation. Efforts have been made to overcome this problem, including prefix GEP (P-GEP) (Li et al., 2005), robust GEP (RGEP) (Ryan and Hibler, 2011), and self-learning GEP (SL-GEP) (Zhong et al., 2016). P-GEP applied depth-first decoding instead of width-first decoding as in the original K-expression. Experiments showed that this decoding method could protect the good structure more than the original GEP (Li et al., 2005). RGEP implements the P-GEP with certain development to simplify the gene structure and avoid invalid syntax. SL-GEP proposes a representation of sub-function as more efficient for complex problems.

Another problem of GEP is gene/ chromosome length. According to Ferreira (2006), the algorithm performs better when the gene/chromosome length expands to a certain threshold. The performance then decreases significantly because of the large solution space to be investigated. To get best performance, and user have to do trial and error to get best gen/chromosome length. Therefore, Bautu et al. (2007) developed AdaGEP that could adaptively adjust the active genes in the chromosome through genemap, a binary string representing gene activation. The gene is activated when the value of a correspondent gene in the genemap is 1. Otherwise, the gene is deactivated or ignored in the decoding process. The study showed that AdaGEP could perform better than the original GEP even when the number of genes surpasses the threshold. Using AdaGEP, the performance decrease after the gene length threshold is not as sharp as the original GEP. However, the user must still decide the number of genes in the chromosome and also the length of the genes.

AdaGEP has tried to overcome number of genes in the chromosome problem. However, the research to overcome the gene length problem is not exist yet. This study aimed to contribute reducing the gene length problem by proposing an adaptive length gene expression programming (ALGEP). It adopted RGEP encoding to enhance its performance with some developments. First, the study proposed varying the gene length for each individual instead of using the same length in the population. The gene length is randomly chosen from a specified range when generating the initial population. The evolutionary process adjusts the length to ensure that the chromosome with the proper gene length survives. Second, the study proposed a length adjustment operator that could delete or insert an allele in the chromosome to make it short or extended. This operator is expected to adjust the gene length to its optimal. A special slice crossover was also proposed to accommodate the crossover between parents with different gene lengths. Additionally, this research also develops a simple direct constant creation method and constant mutation operator to deal with complex constant.

The performance was investigated by applying the proposed algorithm to solve three non-linear symbolic regression problems (SRP). The result was compared with the standard GEP and RGEP, the previous related algorithms. Finally, the proposed algorithm will be applied to solve a complex real problem that is stock market prediction problem. By its nature, stock price is complex, nonlinear, and volatile, and therefore predicting stock price is a difficult task. Stock market prediction is still a challenge for researchers and practitioners (Agrawal et al., 2013). The performance of the proposed algorithm in stock market prediction will be compared with a well-known machine learning method, deep learning ANN.

1.2. Problems Statement

One issue of GEP is the parameters setting. It significantly affects the performance of the algorithm and different problem may require different parameters setting to achieve good result. One of the parameter setting issue is gene/ chromosome length. Too short or too long gene/ chromosome length will reduce the performance of GEP algorithm significantly. AdaGEP have tried to overcome the problem of deciding the number of genes is the chromosome by adaptively activate or deactivated some genes in the chromosome according to genemap, however there is no research found so far to overcome the problem of gene length. The problem addressed in this research is how to develop a better adaptive gene length in the GEP algorithm that produce a better performance.

1.3. Limitations

To make the research more focus and manageable in the time limit, the research will focus on adaptive gene length and use RGEP encoding scheme. The proposed adaptive length GEP will be evaluated for solving three SRPs, predicting monthly Indonesian Stock Exchange (IDX) Composite index, and predicting a monthly stock price. To evaluate the adaptive gene length scheme, this research will only use one gene in the chromosome for solving SRPs. For stock market prediction, this research uses available data from yahoo finance.

1.4. Objective of the Research

The main objective of this research is to develop an adaptive length GEP algorithm that produces better performance. To achieve the main objective, some specific objectives need to be determined. *Firstly*, this research aims to develop an algorithm that can adaptively find a proper gene length to achieve a good performance. *Secondly*, this research will also develop a length adjustment operator that can adjust the length of chromosome during evolutionary process, develop a crossover operator that is suitable for the proposed encoding, and develop a simple constant creation method and a constant mutation operator. *Thirdly*, the performance of the proposed algorithm for solving SRPs will be analyzed by comparing with the previous related GEP algorithms. *Finaly*, the proposed algorithm will be applied for stock market prediction and compare the performance of the proposed algorithm with deep learning ANN algorithm.

1.5. Benefit of the Research

The successful of this research will make a contribution on artificial intelligence field, especially on GEP algorithm development. The outcome of this research will also be useful for other researchers and practitioners in many areas that need an effective and efficient tool for predicting or classifying that can give an understanding about how the decision is made in the model, or how the input variables are interconnected to build the output.

2. LITERATURE REVIEW

This section reviews the previous studies related to the development of GEP, the application of GEP, and stock market prediction.

2.1. Machine Learning and Black Box Problem

Machine learning is a class of artificial intelligent that has automatic learning ability from given data sets to predict (or classify) the out of sample data or the future without programming the relationship between data input and data output explicitly. For instance, in stock market prediction, the relationship between historical data and future price is not known exactly. In this situation, machine learning can be applied using available historical data to automatically learn and give the prediction.

There are popular machine learning methods which have been applied for stock market prediction including ANN, k-nearest neighbor, decision tree, naïve Bayesian, and Ada-boost. Those methods have been applied successfully in many areas (Mohammed et al., 2016). Deep learning is a subfield of ML especially form ANN method that has the ability of learning from raw data without pre-processing that raw data. This ability is obtained from the existence of more layers in the ANN. The main different between traditional ML and deep learning is in how features are extracted. Special treatment for feature extraction is commonly required in traditional ML before applying the learning algorithms (Alom et al., 2019). In deep learning, the features are learned automatically and are represented hierarchically in multiple levels. Figure 2.1 shows the relationship between AI, ML and deep learning.

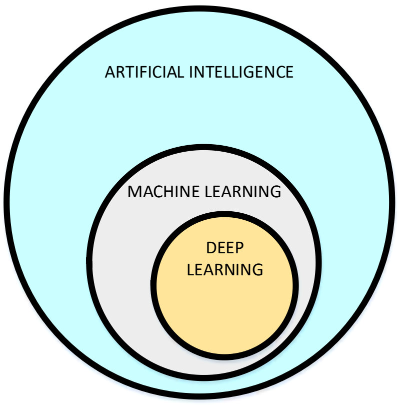


Figure 2.1 AI, ML and deep learning (Chollet, 2018)

There is a problem in ML, called black box problem (Bathaee, 2018; Guidotti et al., 2018). ML methods can probably do a good prediction or classification as well as or better than human do. However, they cannot communicate or explain the reason underlying that decision. This problem can lead to the rejection from the decision maker to apply ML, the difficulty of problem tracing, and law problem regarding the decision guided by ML methods.

Recently, some methods from evolutionary algorithm can overcome the black box problem. Genetic Programming (GP) and its variants is a class of evolutionary algorithm that can produce programs, explicit mathematical equations, or model explanations. Users do not only get the answer from a black box, they can also get the model that explains how the variables are interconnected. Some variants of GP including traditional GP itself have ramified structures of different sizes and shapes similar to the parse trees. Those ramified structure make the process of evolution difficult. The genetic operators must be chosen carefully to make sure producing valid structure. This condition can prevent the finding of good solutions. Another problem using GP is what called bloating problem. During the evolution process, the size of the expression trees may grow quickly, which results in a very big structure and the search for better solution halts as the function become too large. GEP encodes those ramified structures into a linear fix length chromosome. The linear fix length chromosome make GEP gets all benefit and simplicity of GA evolutionary process (Candida Ferreira, 2006; Nedjah et al., 2006). The fix length chromosome of GEP also avoid the bloating problem (Zhong et al., 2017). The next sub-chapter will discuss the development of GEP.

2.2. Development of GEP

Since invented by Ferreira in 1999, GEP has attracted many researchers to develop and apply it. Zhong et al., (2017) make a good review on the developments and applications of GEP. They divide the development work of GEP on several areas i.e. encoding design, adaptation, evolutionary mechanism, cooperative co-evolutionary, constant creation, parallel design, theoretical study, and the applications of GEP. The development of GEP in encoding design, self-adaptation and application areas will be discussed.

2.2.1. Encoding Design

The basic encoding of GEP adopts K-expression to represent the computer program in a fix length string. K-expression use a width first representation. The K-expression has a disadvantage that is a good building block can be easily destroyed in the genetic operations. Some efforts have been done to overcome this shortcoming including P-GEP (Li et al., 2005), RGEP (Ryan and Hibler, 2011), auto defined function (ADF) (Ferreira, 2006), SL-GEP (Zhong et al., 2016), and AdaGEP (Bautu et al., 2007).

P-GEP adopt a new linear genotype representation in prefix notation and a resulted different mapping mechanism between its genotype and phenotype. P-GEP applied depth-first decoding instead of width-first decoding as in original K-expression. The experiments show that this method of decoding can more protect the good structure than the original GEP (Li et al., 2005).

RGEP implement P-GEP encoding scheme with some development to make the gene structure simpler and avoid invalid syntax. RGEP does not use separate head and tail sections in a gene, nor does it re-create and reject invalid individuals. RGEP simply ignores function strings that don’t have a sufficient number of arguments (functions or terminal strings). More explanation about P-GEP and RGEP will be presented in Theoretical Background Section.

ADF is proposed by Koza in GP and then adopted in GEP (Ferreira, 2006). ADF is an automated sub-function that is represented in specific genes. ADFs are then called in the main function and treated as terminals. Using ADF will make the algorithm able to dig more complex functions.

SL-GEP proposes a representation of sub function to be more efficient for complex problem. In SL-GEP, an ADF can be called in the main program as a function. Zhong et al. (2016) report that SL-GEP performs better performance than the current state of the art of GP and GEP variance.

2.2.2. Self-Adaptation

Many parameters should be set in GEP algorithm including length of head, number of genes, population size, mutation rate, transposition rate, crossover rate, etc. The parameters setting significantly affect the performance of the algorithm (Wolpert and Macready, 1997). Different problem may require different parameters setting to achieve good result. Typically, the parameters are defined through trial and error. This is time consuming and may not lead to a good performance. In contrast with other EA method such as GA, only few works have been conducted to deal with parameter setting problem in GEP (Zhong et al., 2017).

One of parameters that has to be determined carefully is gene length. In the original GEP as reported by Ferreira (2006), the performance of the algorithm is increased as the gene length expanded until a certain threshold. After that, the performance will decrease significantly because of the large solution space to be investigated. Figure 2.2 shows the typical relationship between gene length and success rate.

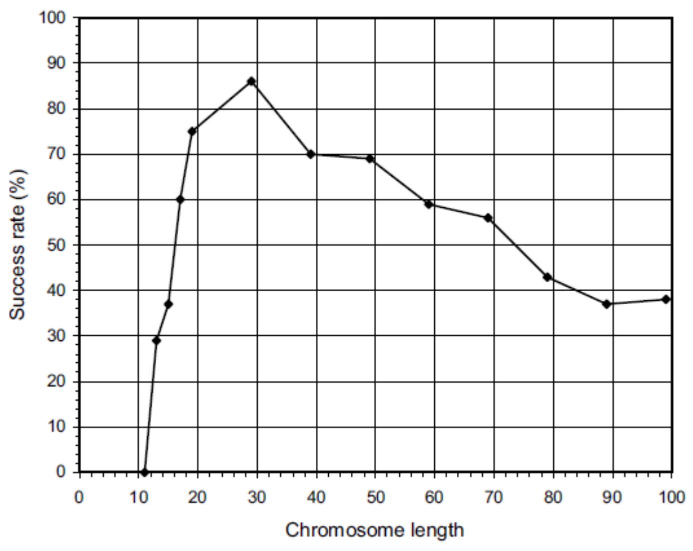


Figure 2.2 Effect of gene length on success rate (Ferreira, 2006)

Bautu et al. (2007) develop AdaGEP that can adaptively adjust the number of active genes in the chromosome through genemap. Genemap is a binary string that represent the activation of the genes in a chromosome. If the value of a correspondent gene in the genemap is 1, the gene is activated, otherwise the gene is deactivated or ignored in the decoding process. The research shows that AdaGEP can still achieve a good performance compared to original GEP even if the number of genes surpass the threshold. Using AdaGEP, the performance decreasing after gene length threshold is not as sharp as the original GEP. However, the user still has to decide the number of genes and the performance is still influenced by the number of genes and the length of head.

Another work is done by Mwaura and Keedwell (2009) to adjust the mutation and crossover rate using a simple heuristic. If the number of better offspring, compared to the parents, produced by an operator (mutation or crossover) exceeded the number of worst offspring in a generation, the rate is increased. If the offspring produced is dominated by worst offspring, the rate is decreased. In Mwaura and Keedwell (2009), the rate of adjustment is fix at 20% of the previous rate. The research reports good results compared with the original GEP in term of success rate and average number of generations.

On important issue in GEP is gen/ chromosome length. AdaGEP have tried to overcome the problem of deciding the number of genes is the chromosome by adaptively activate or deactivated some genes in the chromosome according to genemap, in other word, AdaGEP solved the problem of determining the number of gene in chromosome, but not the gene/ chromosome length and there is no research found so far to overcome the problem of gene length. The problem addressed in this research is how to develop a better adaptive gene length in the GEP algorithm that produce a better performance.

2.2.3. Combination with Other Methods for Better Performance

Some researchers combine GEP with other methods, mainly other metaheuristic methods, to increase the performance of the algorithm. GEP algorithm, by its nature, has large search space and therefore, has limitation in finding appropriate numerical constant for better precision. Combining GEP with other searching algorithm is reported producing better performance than the original GEP.

Zhang et al. (2007) and Zuo et al., (2004) combine GEP with differential evolution and find that their proposed algorithm result better than the original GEP. Yang and Ma (2016) use orthogonal design for multiple-parent crossover operator in the chromosome reproduction. They also introduce an evolutionary stable strategy to maintain the population diversity during the evolution. Fajfar and Tuma (2018) develop a special numeric crossover operator to improve the constant creation ability of RGEP. Mehr (2018) combines GEP with GA for stream flow forecasting. The best individual from the GEP algorithm is then refined by GA to get a better fitness. Deng et al. (2018) combine GEP with artificial fish swarm algorithm, a swarm algorithm, and find that the combination produce better performance than original GEP. Yang et al. (2018) develop a hybrid Kalman filter GEP to reconstruct drawing and handwriting.

2.2.4. The proposed development

This research tries to develop an adaptive length GEP (ALGEP) algorithm that has less impact from gene length setting and produce better performance. *Firstly*, this research proposed an adaptive length chromosome. Instead of using the same gene length in the population, the length is varied for each individual. When generating initial population, the length of a chromosome is randomly chosen from a specified range. The evolutionary process will adjust the gene length and the chromosome with good length will tend to survive. *Secondly*, this research proposes a length adjustment operator. The length adjustment operator can delete a bit in the chromosome to make it shorten or insert a bit to make it extend. This operator is expected to adjust the gene length to its optimal length. *Thirdly*, this research develops a special slice crossover to accommodate the crossover between parents with different gene length. *Finally*, this research develops a simple constant creation method and a constant mutation operator. The comparison of this research with other researches can be seen in Table 2.1.

2.2.5. Applications of GEP

GEP has been reported being applied to many problems. Application of GEP in SRPs are the most widely reported. SRP is the problem on finding mathematical equation for a given data set. Most of prediction problems, including stock market prediction, is some example of SRPs. GEP has an advantages compared to traditional regression techniques such us linear or quadratic regression. Traditional regression techniques usually require determining the equation model first and then estimating the coefficients, while GEP can automatically produce a good model with its coefficients. Although GEP can be used for various types of predictions, GEP has advantages over other machine learning methods for types of data that have causal relationships.

Zhong et al. (2017) shows the applications of GEP in various areas. In the SRP, the application of GEP range widely including the field of agriculture, engineering, production, stock market, and environment. More detail about the use of GEP in stock market will be describe in the next sub-section.

Beside has been applied for SRPs, GEP also has been applied in other area including classification problems, combinatorial optimization problems, automatic model designs, and real parameter optimization problems. An interesting application is in the automated model design problem. The capability of GEP to produce a mathematical equation or a model construction is an advantage of the algorithm compare to other black-box tool like ANN and most of the machine learning tools. GEP even can be applied to construct automatic taxonomy creation of ANN and decision tree (Ferreira, 2006). GEP is also reported being applied for automatic circuit design (Janeiro and Ramos, 2012; Janeiro et al., 2013; Xue-song Yan et al., 2006), line production design (Baykasoğlu, 2008), and robot behavior design (Mwaura and Keedwell, 2015).

Table 2.1. GEP development research map

Authors	**Encoding scheme**			**Adaptive chromosome/ gene length**	**Specific constant generator**	**Adaptive rate of genetic operators**	**Novel Crossover Operator**	**Novel Mutation Operator**	**Combination**			**Application**
	K-expression	**Prefix encoding**	**Single section gene**						**Heuristic**	**Other EAs**	**Others**	
(Zuo et al., 2004)	√				√					Differential Equation	DMI	Time series prediction
(Li et al., 2005)		√										A SRP and 4 testing datasets.
(Bautu et al., 2007)	√			√								3 SRPs
(Mwaura and Keedwell, 2009)	√					√			Simple feedback			2 SRPs
(Zhang et al., 2007)	√				√					Differential Equation		2 SRPs
(Ryan and Hibler, 2011)		√	√									3 SRPs and 5 Boolean decision trees
(Gao et al., 2013)	√											Optimization strategy
(Peng et al., 2014)	√				√							2 SRPs
(Zuo et al., 2004)	√				√						Orthogonal design	Electricity demand prediction, estimation of plastic rotation capacity, and time-series sunspot forecasting

Table 2.1. GEP development research map (Continued 1)

Authors	**Encoding scheme**			**Adaptive chromosome/ gene length**	**Specific constant generator**	**Adaptive rate of genetic operators**	**Novel Crossover Operator**	**Novel Mutation Operator**	**Combination**			**Application**
	Standard K-expression	**Prefix encoding**	**Single section gene**						**Heuristic**	**Other EAs**	**Others**	
(Li et al., 2005)		√										15 SRPs and 6 even-parity problems
(Bautu et al., 2007)	√			√								2 SRPs
(Mwaura and Keedwell, 2009)	√					√						partial differential equation inverse problem
(Fajfar and Tuma, 2018)		√	√		√				A special numeric crossover operator			21 SRPs
(Mehr, 2018)	√									GA		Stream flow forecasting
(Deng et al., 2018)	√									Artificial fish swarm		electricity load forecasting
(Yang et al., 2018)	√										Kalman Filter	Drawing Trace Reconstruction
This research		√	√	√	√		√	√				3 SRPs and stock market prediction

Table 2.2. Applications of GEP

Problem Class	**Specific Problem**	**Articles**
Symbolic Regression Problem (SRP)	Evapotranspiration prediction	(Traore and Guven, 2013; Yassin et al., 2016)
	Velocity field prediction	(Gholami et al., 2015)
	Construction and demolition waste forecasting	(Wu et al., 2015)
	Bus dwell time prediction	(Rashidi and Ranjitkar, 2015)
	Stock market prediction and trading	(Alghieth, 2016; Alghieth et al., 2015; Barbulescu and Bautu, 2012; Bautu et al., 2010; Chen et al., 2014; Huang et al., 2013; Karatahansopoulos et al., 2014; Karathanasopoulos, 2017; Lee et al., 2014; Sermpinis et al., 2013; Visoiu, 2011; Yang et al., 2019) (Ghahtarani et al., 2019) (Bărbulescu and Dumitriu, 2021)
	Caspian sea level change forecasting	(Imani et al., 2014)
	River friction factor prediction	(Md. Azamathulla, 2013)
	Critical velocity estimation for slurry transport via pipeline	(Azamathulla and Ahmad, 2013)
	Performance estimation of a cooling system	(Dikmen, 2015)

Table 2.2 Applications of GEP (Continued)

Problem Class	**Specific Problem**	**Articles**
Symbolic Regression Problem	Estimation of flash point temperature	(Gharagheizi et al., 2012)
	Short-term load forecasting of power system	(Sadat Hosseini and Gandomi, 2012)
	Time-series prediction	(Zuo et al., 2004)
	Estimate flowtime of jobs in a multi-stage job shop	(Baykasoğlu and Göçken, 2009)
	Evaluation of the normalized shear modulus and damping ratio of sand	(Keshavarz and Mehramiri, 2015)
	Prediction performance of PEM fuel cells	(Nazari, 2012)
	Developing a parametric scheme of flow duration curve (FDC) regionalization	(Hashmi and Shamseldin, 2014)
	Electricity demand prediction	(Mousavi et al., 2014)
Classification Problem	Web music emotion recognition	(Zhang and Sun, 2013)
	Multi-label classification	(J. L. Ávila et al., 2011)
	Event selection in high energy physics	(Teodorescu, 2006)
	Benchmark classification	(Chi Zhou et al., 2003)
Automated Model Design Problem	Circuit design	(Janeiro and Ramos, 2012; Janeiro et al., 2013; Xue-song Yan et al., 2006)
	ANN design	(Ferreira, 2006)
	Production line design	(Baykasoğlu, 2008)
	Robot sub-behavior design	(Mwaura and Keedwell, 2015)
Combinatorial Optimization Problem	Automatic design of a hyper-heuristic framework	(Sabar et al., 2015)
	Dynamic scheduling problem	(Nie et al., 2013, 2010)
	TSP and task scheduling	(Candida Ferreira, 2006)
Real Paraneter Optimization Problem	Optimal constant problem	(Ferreira, 2006; Xu et al., 2009)

2.3. GEP for Stock Market Prediction and stock selection

The work on stock market prediction and stock selection using GEP is not as much as other previous popular prediction problem such as ANN and traditional methods. The preliminary search, exploring any type of publications in any time, can only get publication of 5 proceedings, 7 journals and 1 dissertation (see Table 2.3). From those publications, only 2 journals have Q level on Scopus index those are Journal of Forecasting and Mathematics. This indicates that the level of interest from researchers in using GEP for this area is still low, even though the articles collected report that the performance of GEP is superior compared to other methods. The research area in this field is still unexplored well. The nature of stock market which is too much noise and the lack of need for explicit equation probably are the main reason behind the low interest of using GEP for stock price prediction.

The stock market place in the collected publications are Taiwan, USA, and top stocks in worldwide, only three of them placed in the developing or emerging countries. According to Yavas and Dedi (2016), the volatility of stock price in emerging countries are significantly higher than in developed countries. Therefore, using GEP for stock market prediction in Indonesian Stock Exchange (IDX) as an emerging market is interesting and worth doing.

The published articles use GEP to some areas in stock market, i.e. prediction, classification, selection, and trading strategy. Predicting the direction of stock price can be considered as classification. Some play with individual stock price and others with index. In the published articles, GEP may be used purely (Sermpinis et al., 2013; Chen et al., 2014), while others develop modification of GEP (Yang et al., 2019), combine GEP with other method (Karatahansopoulos et al., 2014; Yang and Ma, 2016) or use GEP and other method separately (Alghieth et al., 2015). Some methods are used for comparison with GEP. The most common method for comparison is ANN and its variance, Moving Average Convergence Divergence (MACD) and Autoregressive Moving Average ARMA. Table 2.4 shows the methods used, methods for comparison, and the performance measures.

Table 2.3. Publications of GEP application for stock market

Publication	**Stock Market Place**			
	Taiwan	**USA**	**Other**	**Worldwide**
Dissertation				
Dissertation at De Montfort University				1
Journal				
Applied Soft Computing	1			
Hindawi Computational Intelligence and Neuroscience				1
Journal of Forecasting			1	
Syst. in Accounting, Finance and Management				1
The International Arab Journal of Information Technology				1
Scientia Iranica			1	
Mathematics			1	
Proceeding				
Artificial Intelligence Applications and Innovations (AIAI)		1		
International Conference on Applied Mathematics and Computational Methods in Engineering	1			
International Conference on Complex, Intelligent and Software Intensive Systems		1		
International Symposium on Innovations				1
Procedia Computer Science	1			
Grand Total	**3**	**2**	**3**	**5**

Huang et al. (2013) apply GEP to find profitable strategies for a Taiwan Stock Market Index trading. They find that the strategy obtained from the algorithm produces higher cumulative return and annualized return than the buy and hold strategy. Karathanasopoulos (2017) uses the GEP Trader Tool for forecasting and trading the future contracts of FTSE100, DAX30 and S&P500 daily closing prices from 2000 to 2015. He finds that the proposed approach outperforms all the others approach including Random Walk (RW), MACD, ARMA, GP, ANN, and Recurrent Neural Network (RNN) in the in-sample and out-of-sample periods.

Lee et al. (2014) use GEP for determining trading strategies to 100 top Taiwan stocks. Using two simulation experiments, both experiment shows that the strategy outperforms buy and hold strategy. Sermpinis et al. (2013) apply GEP for forecasting and trading the SPDR Down Jones Industrial Average (DIA), the SPDR S&P 500 (SPY) and the Powershares Qqq Trust Series 1 (QQQ) exchange traded funds (ETFs). They find that the GEP model outperforms all other models under consideration. Chen et al. (2014) forecast a group of stock and then performing some portfolio selection strategies. The experimental results show that the method yields better annualized return then the benchmark method. Karathanasopoulos et.al (2014) uses GEP and compares with GP and other methods to predict and trade ASE 20 Greek index and find that the GEP algorithm perform better than the existing well-known method.

Ghahtarani et al., (2019) used GEP to predict market value of Teheran Stock Market and then used the prediction to produce scenario for portfolio selection. The result shows that GEP outperform other methods for the prediction. Bărbulescu and Dumitriu, (2021) study the performances of GEP for modeling monthly and weekly financial series that present trend and/or seasonality and found that GEP perform better than ARIMA models.

Bautu et al. (2010) predict the direction of change of stock price indices using AdaGEP and compare the prediction performance with Naïve Bayes, Support Vector Machines, ANN, Decision Tree and Random Forrest. The experiment shows that the proposed algorithm is competitive to the previous popular classification methods. Yang et al. (2019) develop a Restricted GEP encode and optimize the structure of the S-system, one of the nonlinear differential equation models, to predict five stock indexes those are Dow Jones Index, Hang Seng Index, NASDAQ Index, Shanghai Stock Exchange Composite Index, and SZSE Component Index. The result shows that the proposed model outperforms deep recurrent neural network (DRNN), flexible neural tree (FNT), radial basis function (RBF), backpropagation (BP) neural network, and ARIMA for predicting 1-week-ahead and 1-month-ahead indexes.

Alghieth (2016) and Alghieth et al. (2015) apply GEP and GP the modelling and prediction of short-term and medium-term stock price and report that GEP result in good performance. Lastly, Barbulescu and Baututu (2012) combine ARMA and GEP for time-series forecasting. They use ARMA to model the linear component and then to derive a model for the residuals using GEP. They find an improvement in the accuracy of hybrid method over pure ARMA and GEP method.

Table 2.4. Methods and performance measures in GEP for stock prediction

Method	**Authors**	**Output**	**Comparison Methods**	**Performance Measure**						
				MAPE	**MSE**	**RMSE**	**MAE**	**Direction accuracy**	**Return**	**Others**
GEP	(Huang et al., 2013)	Buy, Sell, Wait	Buy and Hold						√	
	(Karathanasopoulos, 2017)	Return	RW, MACD, ARMA, GP, ANN, RNN	√		√	√		√	√
	(Lee et al., 2014)	Buy, Sell, Wait							√	
	(Sermpinis et al., 2013)	Daily Return	RW, MACD, GP, ANN, RNN, Gaussian Mixture Neural Network (GM)			√	√			√
	(Chen et al., 2014)	Buy/Sell	GA						√	
	(Karatahansopoulos et al., 2014)	Buy/Sell	ANN, ARMA, MACD, Naïve, GP		√				√	
	(Ghahtarani et al., 2019)	Price	Adaptive Neuro-Fuzzy Inference System (ANFIS), FDA, decision tree, and random forest			√				√
	(Bărbulescu and Dumitriu, 2021)	Price	ARIMA	√	√	√	√			√
AdaGEP	(Bautu et al., 2010)	Price Direction	Naïve Bayes, Support Vector Machines, ANN, Decision Tree and Random Forrest					√		
Restricted RGEP	(Yang et al., 2019)	1 week and 1 month price ahead	Deep recurrent neural network (DRNN), flexible neural tree (FNT), radial basis function (RBF), backpropagation (BP) neural network, and ARIMA			√	√			√
GEP and GP	(Alghieth, 2016)	5 days price ahead	ANFIS	√						
	(Alghieth et al., 2015)	5 days and 56 days ahead of price direction						√		
ARMA and GEP	(Barbulescu and Bautu, 2012)	Monthly index	ARMA and GEP		√					

3. Theoretical Background

EAs are searching algorithm that inspired from the biological evolution in nature. The algorithms usually use random guided evolution and population based. The better individual (solution) will have the bigger chance to survive and produce the next individual. GAs is a class in EAs. The plural use in "GAs" is to differentiate with a popular variant of GAs that is genetic algorithm (GA). GAs adopt the concept of Darwinian theory of evolution. Three popular GAs variants are GA, GP and GEP. Before describing GEP, the main concern in this research, this section will briefly introduce the biological perspective as the model base of GEP, and the predecessor of GEP those are GA and GP.

3.1. Biological Perspective

Expression of genetic information in a cell is very complex and involves hundreds of molecules. The main players are DNA (Deoxyribo Nucleic Acid), RNA (Ribo Nucleic Acid) and protein. DNA is a long string double helix arranged from four nucleotides which are represented by character "A", "T", "C" AND "G" (see Figure 3.1). The sequence of these four nucleotides represents the genetic information of an individual. The genetic information is the blueprint of all organism in the earth. The color of hair and skin, the type of blood, the risk of certain genetic disease etc. are represented to the certain sequence of nucleotides. A mammal genome consists approximately $5x10^9$ base letters. The representation of DNA can constructs about 300,000 proteins that built the organism (Ferreira, 2006).

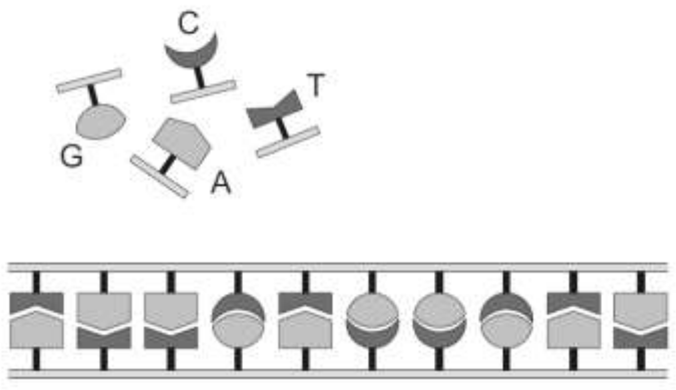
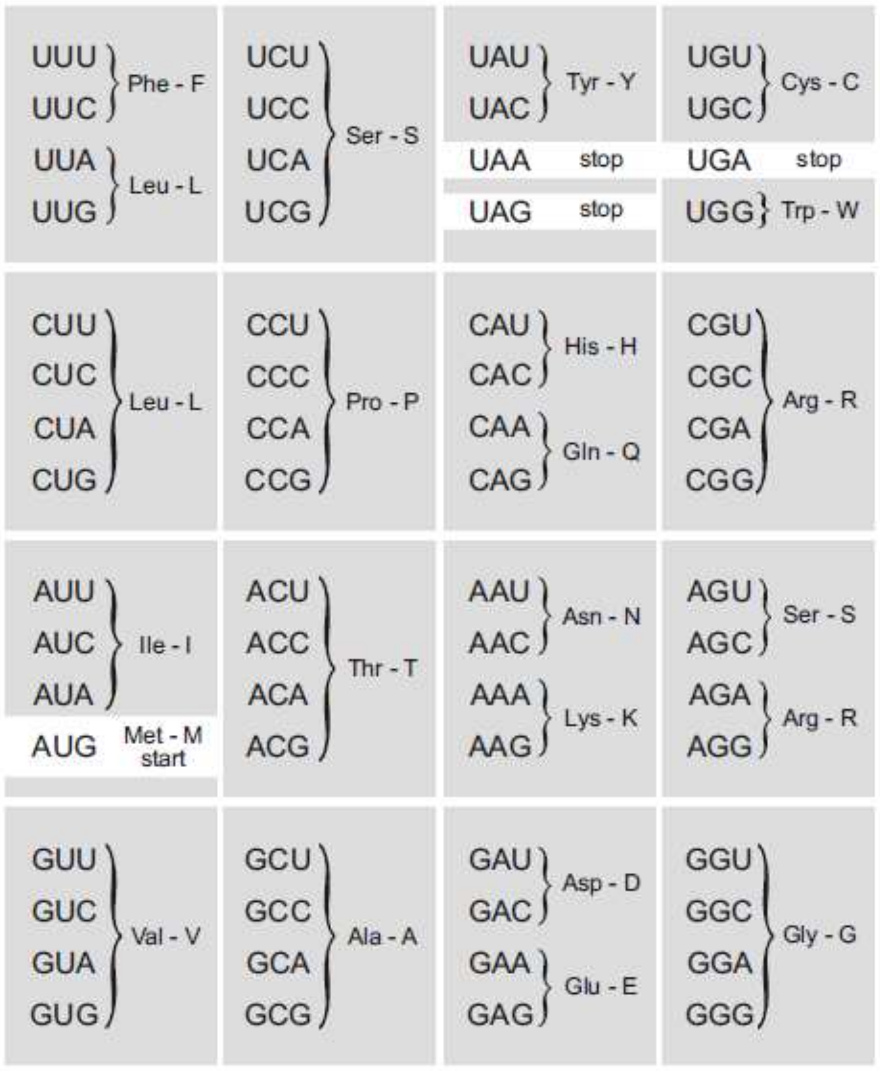


Figure 3.1. Double helix structure of DNA (Ferreira, 2006)

RNA is the working copy of DNA in the protein synthesis. The different between DNA and RNA is that RNA string mostly formed as single-stranded. The nucleotides of

RNA are symbolized as “A”, “U”, “C” AND “G”. RNA can be divided into mRNA (messenger RNA) and tRNA (transfer RNA). mRNA copies the structure of DNA and tRNA transfers the structure into nucleic acid. A combinations of nucleic acid will form a protein. Tree certain sequence of RNA code, called codon, will form a certain type of nucleic acid. There are exist 20 type of nucleic acid. Since the number of codon is $4^3 = 64$, a type of nucleic acid may be represented by more than one codons. Figure 3.2 shows the codon representation of nucleic acid in RNA.



UUU UUC Phe - F UUA UUG Leu - L	UCU UCC UCA UCG Ser - S	UAU UAC Tyr - Y UAA stop UAG stop	UGU UGC Cys - C UGA stop UGG Trp - W
CUU CUC CUA CUG Leu - L	CCU CCC CCA CCG Pro - P	CAU CAC His - H CAA CAG Gln - Q	CGU CGC CGA CGG Arg - R
AUU AUC AUA Ile - I AUG Met - M start	ACU ACC ACA ACG Thr - T	AAU AAC Asn - N AAA AAG Lys - K	AGU AGC Ser - S AGA AGG Arg - R
GUU GUC GUA GUG Val - V	GCU GCC GCA GCG Ala - A	GAU GAC Asp - D GAA GAG Glu - E	GGU GGC GGA GGG Gly - G

Figure 3.2. RNA representation of nucleic acid (Ferreira, 2006)

As can be seen from Figure 3.3, there are exist start and stop signal. The protein synthesis process will only decode the codon within start and stop signal and neglect the codon other than that. There are several start and stop signals in a long RNA string. Figure 3.3 illustrate the start and stop signal. However, note that the length of string in the real RNA between start and stop signal is much longer than what illustrated in Figure 3.3.

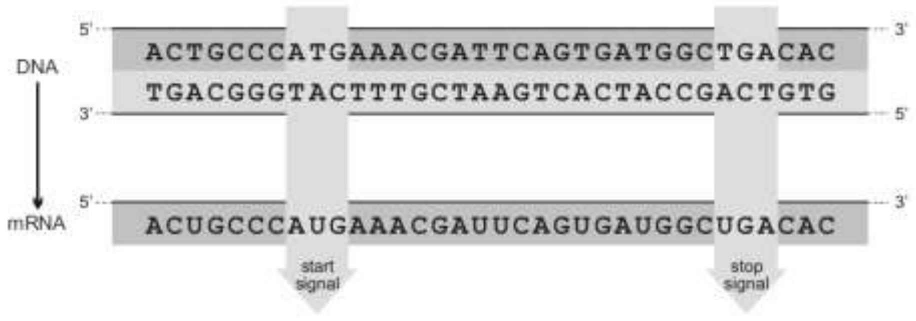

5'
3'
DNA
ACTGCCCATGAAACGATTCAGTGATGGCTGACAC
TGACGGGTACTTTGCTAAGTCACTACCGACTGTG
3'
5'
5'
3'
mRNA
ACUGCCCAUGAAACGAUUCAGUGAUGGCUGACAC
start signal
stop signal

Figure 3.3. Start and stop signal in DNA and RNA (Ferreira, 2006)

In the nature, there are genome restructuring through the process of evolution. The genome restructuring could be the result of mating, adapting the environment, or just random case. The genome restructuring process includes mutation, recombination or crossover, transposition, and duplication. *Mutation* is the small change in a nucleotide. The form of mutations is including: *substitution*, a type of nucleotide (a letter) is substituted by another letter; *deletion*, a letter is simply deleted; or *insertion*, a letter is inserted between two letter. Figure 3.4 illustrated the mutation type on DNA. Mutation process rate in the nature is very low.

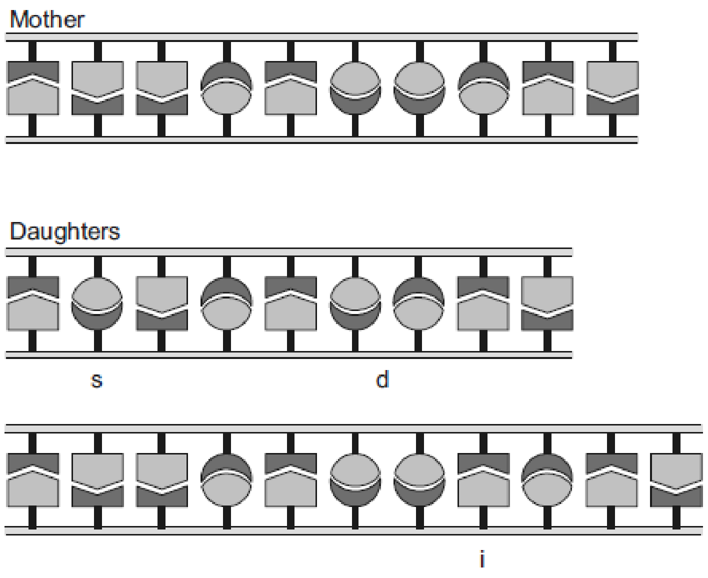


Figure 3.4. Mutation type in DNA
Substitution (s), deletion (d) and insertion (i)
(Ferreira, 2006)

Recombination or crossover is the process of combining some DNA parts of an individual with some other DNA parts of another individual. The process of recombination is commonly the result of mating. *Transposition* is the proses of moving some genes to other position in a genome. The effect of transposition in the living organism is very drastic and the rate is very low. *Gene duplication* sometimes occurs in an organism. The process may create a new protein.

According to Darwinian theory, the genome restructuring provides organism diversity, however, not all new produced organisms will survive. The only organisms those fit with their environments will tend to be survive. This principle is known as “survival of the fittest”. This principle is adopted by most of evolutionary algorithms including GAs where the good solutions tend to be survive in the next generation and then adapt to produce better solution.

3.2. Genetic Algorithm

GA is invented by John Holland in 1960. GA is a metaheuristic optimization algorithm based on the principle of genetic and evolution in nature. In GA, a solution (phenotype) is encoded in a chromosome in the form of a fix length string. In the basic GA, the string is binary that contains 0 and 1 only. Recent developments of GA use other types of encoding including real number, permutation, hexadecimal, etc. The typical process of GA, as shown in Figure 3.5, start with encoding and then generating initial random population, fitness evaluation, selection, crossover, mutation and replacement.

Encoding is the process of translating the real solution to the gene. The process of mutation and crossover are done in the form of gene. Figure 3.6 Show the encoding and decoding scheme. The basic encoding is binary encoding. In binary encoding, the gene only contains the combination of 0 and 1. A chromosome may contain one or more genes. Figure 3.7 shows examples of binary encoding with lower bound -5.12 and upper bound 5.12.

A population is the collection of many chromosomes. The number of chromosomes in the population is called population size. The initial population is usually generated randomly. The population size should be determined properly. If the population size is too small, a good solution might not be found; if the population size is too big, the computation time required is probably too long. A different problem may have a different proper population size.

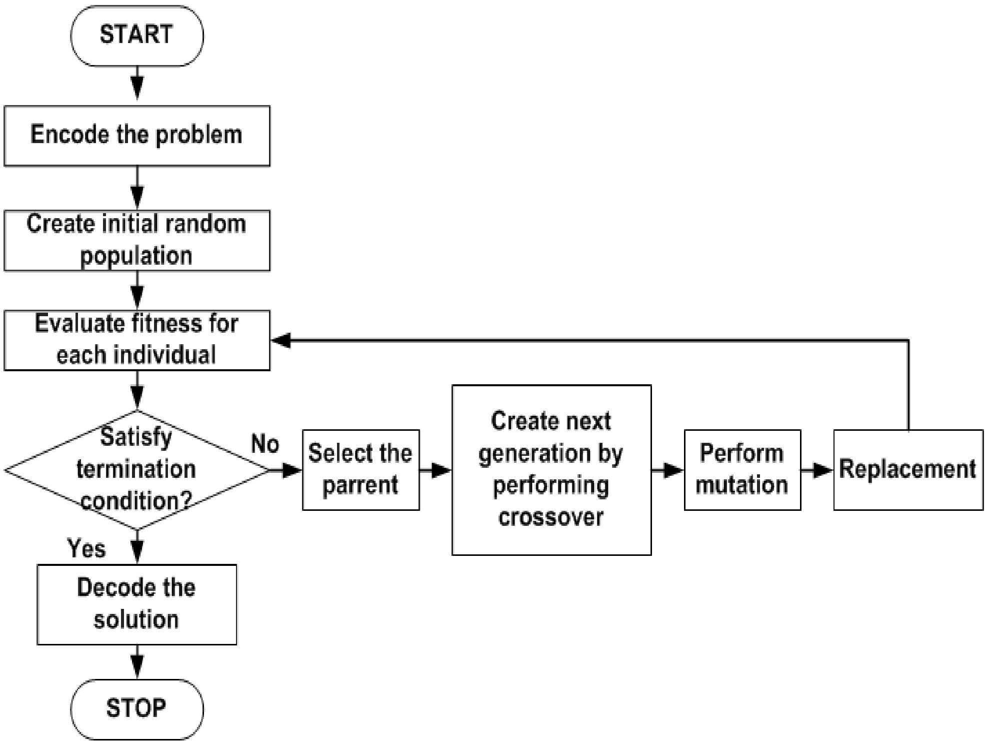

START
Encode the problem
Create initial random population
Evaluate fitness for each individual
Satisfy termination condition?
No
Select the parrent
Create next generation by performing crossover
Perform mutation
Replacement
Yes
Decode the solution
STOP

Figure 3.5. The typical process of GA

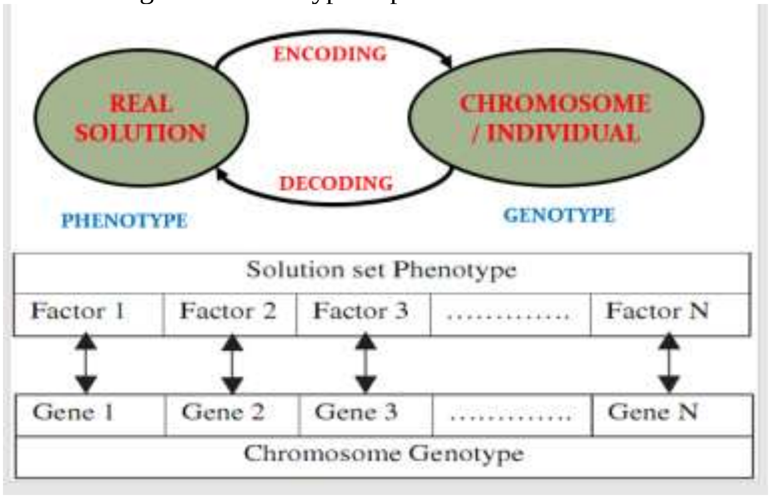
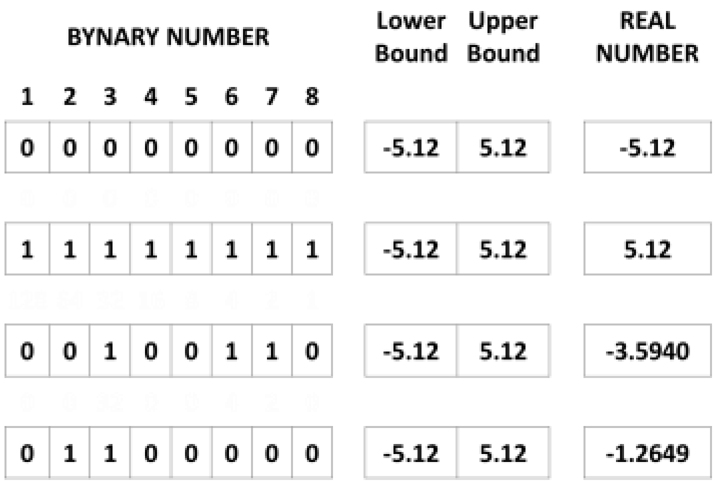


Figure 3.6. Encoding and decoding

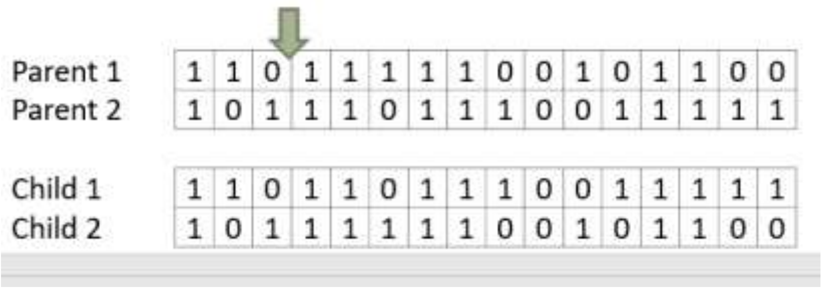
Selection is the process of choosing chromosomes for mating to produce children. The fittest chromosome will have the biggest probability to be chosen. The common method for selection in roulette wheel selection. This method gives a probability for a chromosome to be chosen linear with the fitness of the chromosome.



BYNARY NUMBER								Lower Bound	Upper Bound	REAL NUMBER
1	2	3	4	5	6	7	8			
0	0	0	0	0	0	0	0	-5.12	5.12	-5.12
0	0	0	0	0	0	0	0			
1	1	1	1	1	1	1	1	-5.12	5.12	5.12
128	64	32	16	8	4	2	1			
0	0	1	0	0	1	1	0	-5.12	5.12	-3.5940
0	0	32	0	0	4	2	0			
0	1	1	0	0	0	0	0	-5.12	5.12	-1.2649

Figure 3.7. Examples of binary encoding

After a number of parents are chosen, those parents will be mated. Typically, two consecutive parent will perform crossover to produce two children. Several methods of crossover have been developed including one-point crossover and two-point crossover. One-point crossover cut the two parents at a random point. The first child will have the left side gene, from that point, of first parent and the right side of the second parent. In opposite, the second child will be inherited the left side of the second parent and the right side of the first parent. Figure 3.8 illustrates the process of one-point crossover. Two-point crossover, as illustrated in Figure 3.9, uses two random point to perform the crossover.



Parent 1	1	1	0	1	1	1	1	1	0	0	1	0	1	1	0	0
Parent 2	1	0	1	1	1	0	1	1	1	0	0	1	1	1	1	1
Child 1	1	1	0	1	1	0	1	1	1	0	0	1	1	1	1	1
Child 2	1	0	1	1	1	1	1	1	0	0	1	0	1	1	0	0

Figure 3.8. One-point crossover

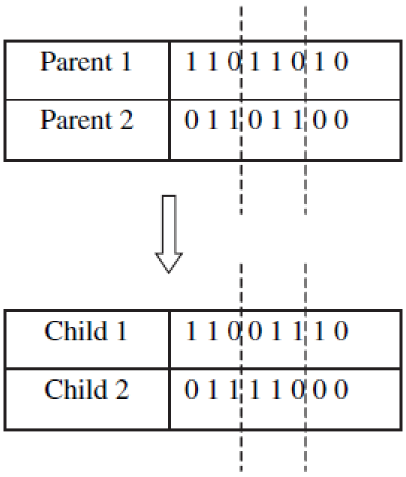


Figure 3.9. Two-point crossover (Sivanandam and Deepa, 2007)

Mutation is an important process of GA; it prevents the algorithm trapped in local solution. However, mutation can destroy a good solution even though it occasionally produces a good solution. Therefore, the mutation rate is typically kept in low level. Substitution is a common method for mutation. Using this method, a randomly chosen code in a bit in a gene is substituted by another code. In binary encoding, 0 is substituted with 0 and vice versa as shown in Figure 3.10.

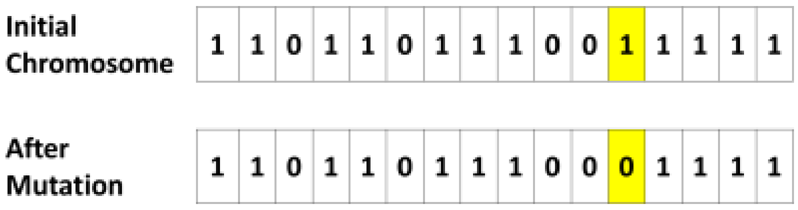


Figure 3.10. Substitution or reverse mutation

The last step is replacement that is the step of replacing the old generation with the new chosen generation. One of the method for replacement is generation update. In this method, the children replace the entire old population. Other methods include roulette wheel, random replacement, weak parent replacement and both parent replacement.

The GA algorithm is iterated until one termination condition is reached. The termination conditions may include: predefined maximum generation is reached, maximum time is passed, no change in the best fitness for specific number of generation and reach the optimal solution (if known).

3.3. Genetic Programming

GP is a class of Genetic Algorithms that can produce a computer program (or function) through an evolutionary process. The process is similar with the evolutionary process of GA. The main different is that, in GP, the gene is represented in an expression tree. The tree contains sub-tree, function, and terminal. The evolutionary operators are adjusted to the gene representation. Below are a programming language and prefix notation of the expression tree in Figure 3.11.

Programming language:

```
IF (Y>X OR Y<4) THEN i:=(i+1), ELSE i:=0.
```

Prefix notation:

```
IF(OR(>(Y,X),<(Y,4)),:=(i,+(i,1)),:=(i,0)).
```

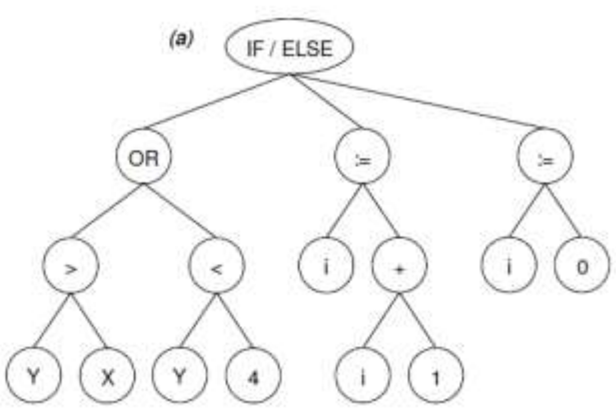


Figure 3.11. Expression tree example (Ferreira, 2006)

A common crossover method for GP is one-point sub-tree crossover (see Figure 3.12). By this method, a random piece of sub-tree in first parent is replaced by a random sub-tree of another parent to produce child 1 and vice versa for child 2. Mutation can be performed by deleting or replacing a subtree with a random generated sub-tree, by replacing a function with another function or a terminal, and other methods. Figure 3.13 shows examples of mutation process in GP.

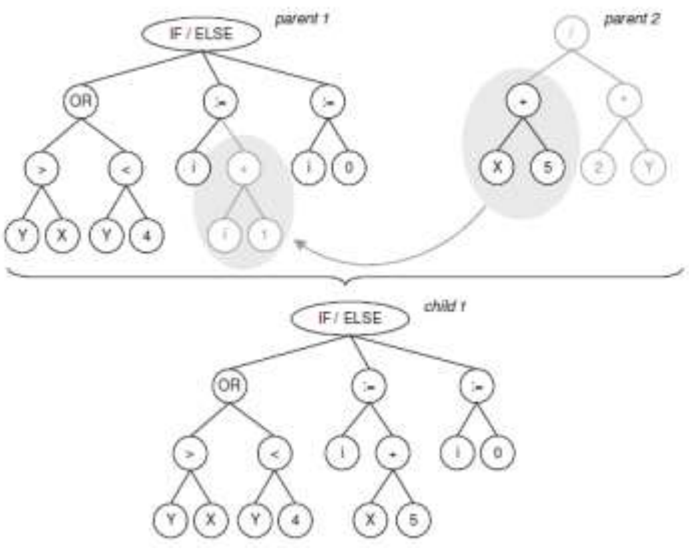


Figure 3.12. One-Point sub-tree crossover (Ferreira, 2006)

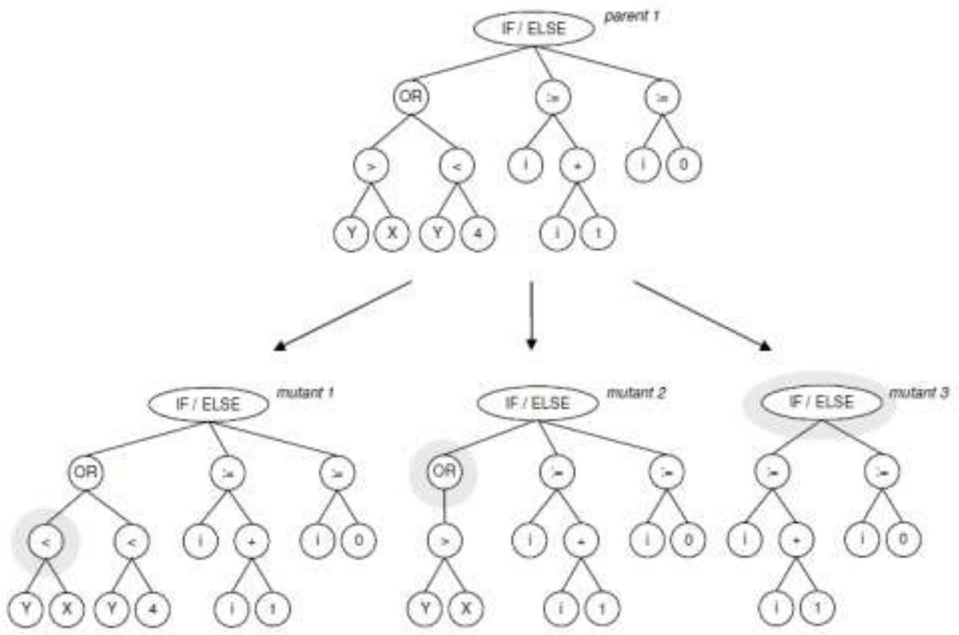


Figure 3.13. Examples of mutation in GP (Ferreira, 2006)

3.4. Gene Expression Programming

GEP was invented by Ferreira in 1999 (Candida Ferreira, 2006). GEP uses a fixed-length string to represent and decode the gene to an expression tree to produce a mathematical equation. The solution is expressed using the Karva language by adopting a width-first scheme to translate a gene into an expression tree. In GEP, a gene contains a head and a tail section. The head could contain function and terminal string, while the tail has only the terminal. Function and terminal strings are defined by the user. Examples of function strings are *, /, +, -, sqrt, sin, and cos, while terminal strings could include constants and variables. The length of a tail depends on the length of the head, calculated using (3.1)

$$t = h(n_{max} - 1) + 1 \tag{3.1}$$

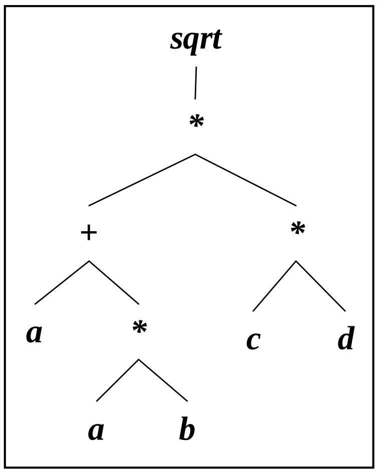
Where t is the length of the tail, h is the length of the head, and nmax is the maximum number of arguments in the functions. For instance, sqrt (square root) has one input argument while * (multiplying operator) has two input arguments. Using this scheme, all generated genes and the result of all generic operators are translated into valid functions. This is one advantage of GEP, where there is no need to iteratively find a valid syntax after genetic manipulation of a gene, as in genetic programming. Figure 3.14 shows the gene decoding process, where a fixed-length string containing a head and tail is decoded into an expression tree and translated into a function. In the example, the head length is six, and the maximum number of arguments in the function is two. Therefore, the tail length (t) is:

$t = 6(2\text{-}1)+1 = 7$

Gene:

Head						Tail						
sqrt	*	+	*	*a*	*	*c*	*d*	*a*	*b*	*c*	*b*	*b*

Expression Tree



Equation

$$\sqrt{\big(a + (a * b)\big) * (c * d)}$$

Figure 3.14. Example of a GEP Decoding

Figure 3.15 shows the typical GEP process, where the initial population is created randomly. The number of chromosomes in the population, called population size, is predefined. Each chromosome is then expressed or translated into a mathematical function or computer program. The program is executed to obtain the fitness function of each chromosome.

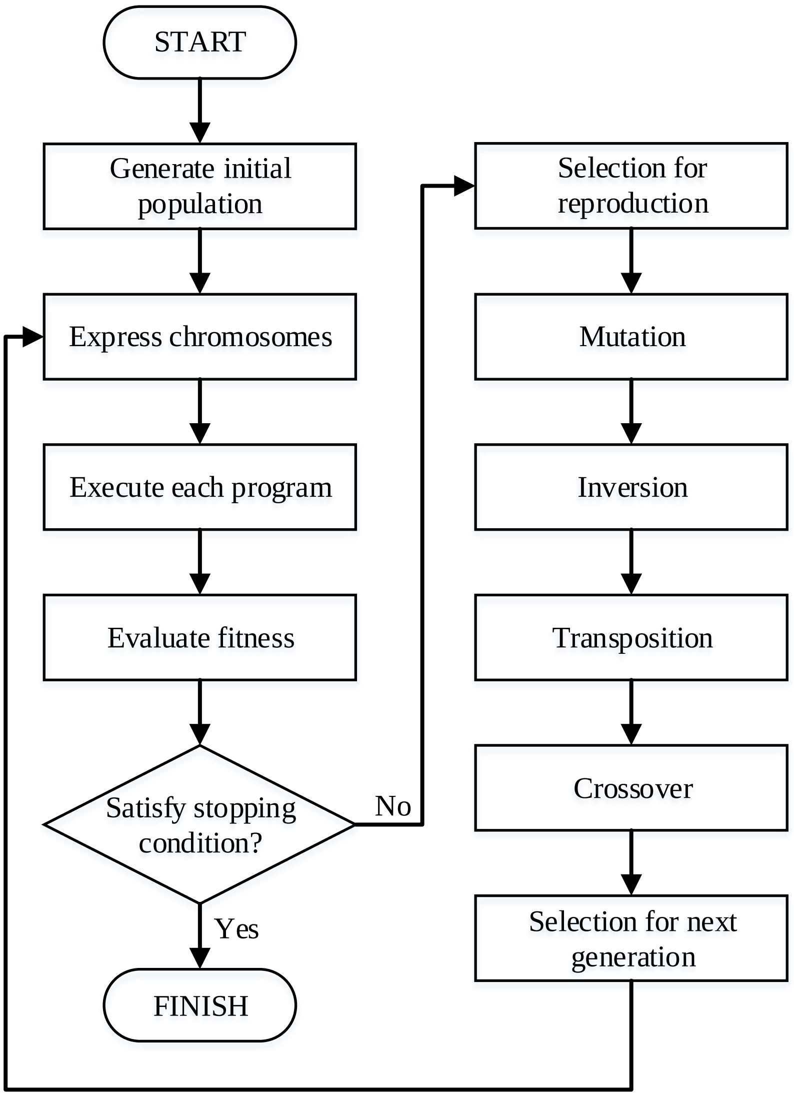
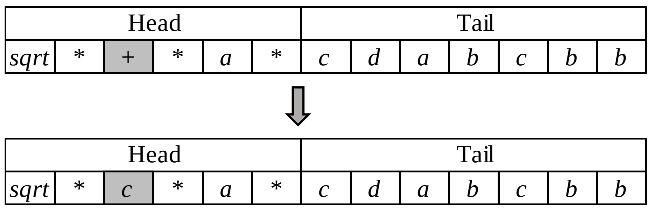
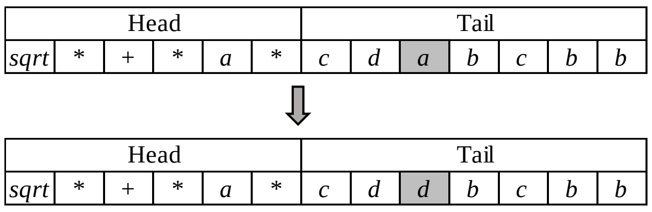


Figure 3.15. Typical GEP process

The selection scheme is similar to genetic algorithm, where the roulette wheel is the commonly used operator for selection. A mutation is the most effective evolutionary operator (Candida Ferreira, 2006), though it could be constructive or destructive. The mutation rate is usually low because a rate close to 1 would make the algorithm similar to a random search. Ferreira proposed a guide to determine the mutation rate (pm) equal to 2/ (gene length). For instance, the rate is 0.2 when the gene length is 10. Mutation could occur in both the head and tail and the process replaces an allele with another allele. When the mutation occurs in the head, a function or terminal is replaced by another function or terminal. In contrast, the replacement is only for the terminal when the mutation occurs in the tail. Figure 3.16 a) and b) show a mutation in the head and tail sections, respectively.



a)



b)

Figure 3.16. Example of mutation: a) in the head. b) In the tail

Inversion is the process of inverting a sequence part of a gene at a rate usually less than 0.1. Figure 3.17 shows an example of an inversion process that drastically changes fitness. Transposition is copying and inserting a chromosome fragment into another position, as shown in Figure 3.17.

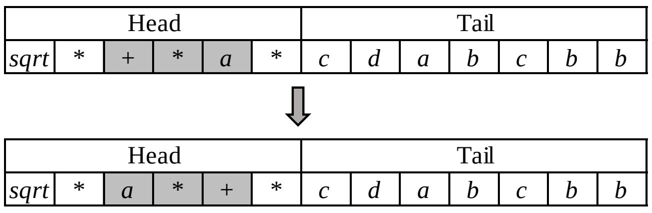


Figure 4. Example of an inversion

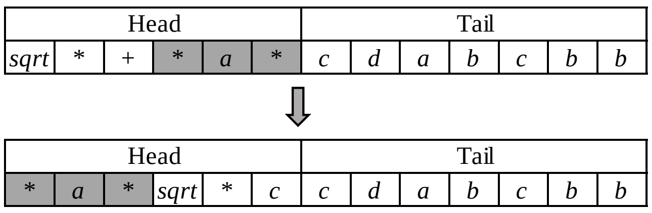


Figure 3.17. Example of a transposition

GEP crossover is performed similarly to crossover in genetic algorithm. One-point crossover slices two parents in a point and exchanges each section to produce two children. The two-point crossover is the same process but with two slice points. Figure 3.18 and Figure 3.19 show examples of one-point and two-point crossover.

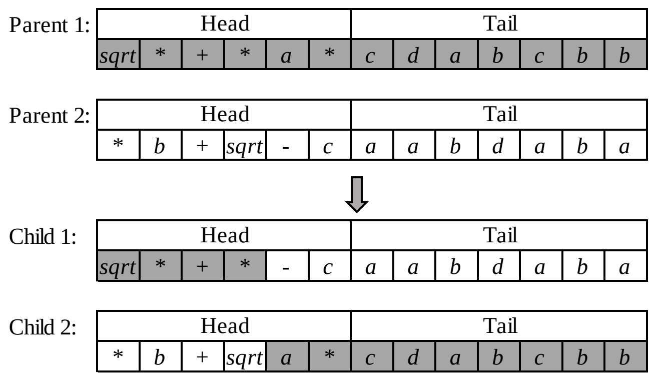

Parent 1:
Head
Tail
sqrt * + * a * c d a b c b b
Parent 2:
Head
Tail
* b + sqrt - c a a b d a b a
Child 1:
Head
Tail
sqrt * + * - c a a b d a b a
Child 2:
Head
Tail
* b + sqrt a * c d a b c b b

Figure 3.18. Example of a one-point crossover

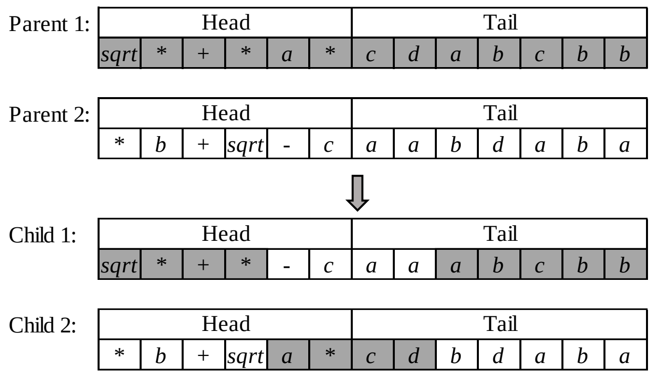

Parent 1:
Head
Tail
sqrt * + * a * c d a b c b b
Parent 2:
Head
Tail
* b + sqrt - c a a b d a b a
Child 1:
Head
Tail
sqrt * + * - c a a a b c b b
Child 2:
Head
Tail
* b + sqrt a * c d b d a b a

Figure 3.19. Example of a two-point crossover

The process is iterated to satisfy one of the termination conditions, including reaching a predefined maximum generation, passing the maximum time, no change in the best fitness for a specific number of generations, and reaching an optimal solution (if known).

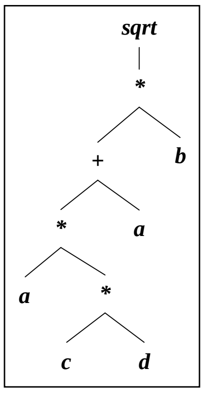
3.5. P-GEP

P-GEP adopts a linear genotype representation in prefix notation and a different mapping mechanism between its genotype and phenotype. It applies depth-first decoding instead of width-first decoding as in the original K-expression. The experiments showed that this decoding method could protect the good structure more than the original GEP (Li et al., 2005). Figure 3.20 shows an example of P-GEP decoding.

Gene:

Head						Tail						
sqrt	*	+	*	*a*	*	*c*	*d*	*a*	*b*	*c*	*b*	*b*

Expression Tree



Equation $\sqrt{((a*(c*d))+a)*b}$

Figure 3.20. Example of a P-GEP encoding

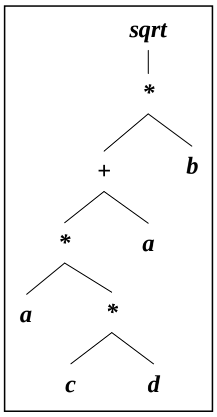
3.6. RGEP

RGEP implements the P-GEP encoding scheme with some development to simplify the gene structure and avoid invalid syntax. It does not use separate head and tail sections in a gene and does not re-create and reject invalid individuals. The method ignores functions with no sufficient functions or terminal strings. Figure 3.21 shows an example of RGEP decoding. In the example, the grey strings "*" and "-" have insufficient input arguments and are ignored in the decoding process.

Gene:

sqrt	*	+	*	*a*	*	*c*	*d*	*a*	*	-	*b*

Expression Tree



Equation $\sqrt{((a*(c*d))+a)*b}$

Figure 3.21. Example of a RGEP encoding

3.7. Prediction/ Forecasting Accuracy

The accuracy of a model can be represented in error prediction. The lower the error, the better the model is. The error of the prediction is simply the different between the actual value and the forecast. Accuracy can be described as how close the prediction/ forecast with the actual value. The accuracy measurement for numerical prediction can be divided into three type:

- scale-dependent measurement,
- percentage base measurement, and
- relative error base measurement.

For the rest of the sub-section, these notations will be used. Suppose there are *n* period of data and a forecast have been done using a particular method. Error for each period *t* can be calculated based on the actual data and the forecast result.

$$e_t = Y_t - F_t \tag{3.2}$$

Where:

e_t	: forecasting error of the period *t*
Y_t	: the actual observation of period *t*
F_t	: the forecast of period *t*

3.7.1. Scaled-dependent measurement

The value of scale-dependent measurement is dependent by the scale of the data, however, it is difficult to compare between one to another forecasting of different data or series using this method. For example, the forecasting of a company stock cannot be compared to another company since the price are different. However, this method may be used for comparing two different forecasting method with the same data or series. This method may also be used for the data that has typical scale such as stock return since the stock return has the same scale for all stocks. Below are some methods in this type of measurement.

a. *Average of Errors (E)*

$$E = \frac{\sum_{t=1}^{n} e_t}{n} \tag{3.3}$$

b. *Mean Absolute Error (MAE) or Mean Absolute Deviation (MAD)*

$$MAE = MAD = \frac{\sum_{t=1}^{n} |e_t|}{n} \tag{3.4}$$

c. *Median Absolute Error (MdAE)*

$$MdAE = Median(|e_t|) \tag{3.5}$$

d. *Sum Squared Error (SSE)*

$$SSE = \sum_{t=1}^{n} e_t^2 \tag{3.6}$$

e. *Mean Squared Error (MSE) or Mean Squared Prediction Error (MSPE)*

$$MSE = MSPE = \frac{\sum_{t=1}^{n} e_t^2}{n} \tag{3.7}$$

f. *Root Mean Squared Error (RMSE)*

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} e_t^2}{n}} \tag{3.8}$$

In the average error (E), the direction of the error, (-) and (+), can eliminate each other, therefore, average error cannot be used for measuring the accuracy performance since a big error will have the same E as a small error if the direction balance in the same. However, this method can be used to measure the error direction tendency. A forecast with positive *E* tend to be over-estimate while negative value of *E* tends to be under-estimate.

MAE or MAD has a simple formula making this measure easy to be implemented. However, it is highly affected by outliers and still scale-dependent. This measure can be applied for comparing prediction methods for return prediction, since the scale of return, in term of proportion or percentage, is tend to same for all stock. Moreover, this method can be interpreted easily for return prediction. MdAE is less affected by outliers but still scale dependent and tend to avoid the unbalance density in the calculation.

SSE, MSE and RMSE are the variance of square error. These are popular methods, mostly because of their theoretical relevance in statistical modelling. The bigger errors are disliked and therefore given more weight, the error itself, However, they are more sensitive to outliers than MAE or MdAE, which has led some authors to recommend against their use in forecast accuracy evaluation. Rankings of methods based on the RMSE were highly unreliable (Armstrong, 2001; Armstrong and Collopy, 1992; Hyndman and Koehler, 2006) .

3.7.2. Percentage base measurement

These measures are not dependent of the data scale, and therefore, suitable for comparing forecast result of different data. However, these measures have the disadvantage of being infinite or undefined if $Y_t = 0$ for any t in the period of interest, and probably having large value when Y_t is close to zero (Hyndman and Koehler, 2006). Below are some methods in this type of measurement.

a. *Mean Absolute Percentage Error (MAPE)*

$$MAPE = \frac{\sum_{t=1}^{n} \left|\frac{e_t}{Y_t}\right|}{n} * 100\% \tag{3.9}$$

b. *Median Absolute Percentage Error (MdAPE)*

$$MdAPE = Median(\left|\frac{e_t}{Y_t}\right| * 100\%) \tag{3.10}$$

c. *Root Mean Square Percentage Error (RMSPE)*

$$RMSPE = \sqrt{\frac{\sum_{t=1}^{n} \left(\frac{e_t}{Y_t}\right)^2}{n}} \tag{3.11}$$

d. *Root Median Square Percentage Error (RMdSPE)*

$$RMdSPE = \sqrt{Median\left(\left(\frac{e_t}{Y_t}\right)^2\right)} \qquad (3.12)$$

e. *Symmetric Mean Absolute Percentage Error (sMAPE)*

$$sMAPE = \frac{\sum_{t=1}^{n} \left|\frac{2 * e_t}{Y_t + F_t}\right|}{n} * 100\% \qquad (3.13)$$

f. *Symmetric Median Absolute Percentage Error (sMdAPE)*

$$sMdAPE = Median\left(\left|\frac{2 * e_t}{Y_t + F_t}\right| * 100\%\right) \qquad (3.14)$$

g. *Median Absolute Prediction Error as a percentage of the Standard deviation (MdAPES)*

$$MdAPES = Median\left(\left|\frac{e_t}{SD}\right| * 100\%\right) \qquad (3.15)$$

Where: *SD = the standard deviation of* Y_t

MAPE is the most popular accuracy measure for forecasting (Fildes and Goodwin, 2007). The reasons are it is intuitive, easy to be interpreted and can be applied to measure or compare accuracy for both individual item and across item groups (Kolassa and Martin, 2011). However, the MAPE has shortcomings. Some researchers has revealed the disadvantages of using MAPE (Davydenko and Fildes, 2014; Foss et al., 2003; Hyndman and Koehler, 2006; Kolassa and Martin, 2011; Makridakis, 1993). *First*, the value of MAPE, and also all of its variance, will be undefined if one or more of the actual demands are zero, and it explodes if there is actual data which are very small compared to the forecast, and if the value of the actual data is close to zero. *Second*, outlier data can make a significant change of MAPE. *Third*, equal errors above the actual value result in a greater APE (Absolute Percentage Error) than those below the actual value (Makridakis, 1993). For example, when the actual value is 150 and the forecast is 100 the APE is 33.33%, however, when the actual is 100 and the forecast 150 the APE is 50%. *Forth*, using the MAPE for forecast methods comparison will lead to a systematically under-forecast result, especially for series that fluctuate widely (Kolassa and Martin, 2011). The later problem is illustrated nicely by (Kolassa and Martin, 2011) using the rolling dice problem. The dice roll could represent natural random fluctuation for actual data without trend, seasonality, or causal factors to influence to the data. Suppose a dice is rolled and forecaster should predict the result for each rolling. The expected value in this case will be 3.5 and that is indeed the best forecast in this situation. However, using MAPE, forecast of 2 will lead to better performance than forecast with 3.5. This problem arise as the low denominator will produce bigger APE than high denominator for the same error, thus, the undervalue forecast will tend to have the smaller APE than the higher forecast.

Some researchers suggest the use of median instead of mean for calculating percentage error while others suggest the use of relative error. (Armstrong and Collopy, 1992) suggest the use of MdAPE to select among the forecasting methods when many series of data are available while (Billah et al., 2006) proposes the MdAPES. The use of MdAPE can reduce the effect of outlier data, however, it cannot reduce the other drawbacks. Moreover, medians are not relative measures and they are not recommended for general use (Makridakis, 1993). The use of square percentage error does not eliminate any disadvantage of the MAPE, instead it increases the severe from the outlier data.

Even with its shortcomings, Makridakis (Makridakis, 1993) argues that MAPE is the only choice for the forecasting accuracy method due to its applicability, interpretability and reliability. Thus, he suggests that it is better to look for ways of correcting the drawbacks of MAPE rather than searching for alternative measures which are less desirable than MAPE. He suggests the use of sMAPE. However, (Goodwin and Lawton, 1999) show that sMAPE is also produce an asymmetric result.

3.7.3. Relative base measurement

Another way of scaling is to divide each forecasting error by the error obtained using another standard method of forecasting. Let denote $r_t = e_t / e_t^*$ as the relative error, where e_t^* is the forecast error obtained from the benchmark method. Usually, the benchmark method is the random walk, or naïve, where F_t is equal to actual data the last period before the forecasting period, Y_{t-1} (Armstrong and Collopy, 1992).

a. *Mean Relative Absolute Error (MRAE)*

$$MRAE = \frac{\sum_{t=1}^{n} |r_t|}{n} \tag{3.16}$$

b. *Median Relative Absolute Error (MdRAE)*

$$MdRAE = Median(|r_t|) \tag{3.17}$$

c. *Geometric Mean Relative Absolute Error (GMRAE)*

$$GMRAE = \sqrt[n]{\prod_{t=1}^{n} |r_t|} \tag{3.18}$$

d. *Percent Better*

$$Percent\ Better = \frac{count\_if(|e_t| \le |e_t^*|)}{n} * 100\% \tag{3.19}$$

e. *Relative Mean Absolute Error (RelMAE) or Cumulative Relative Absolute Error (CumRAE)*

$$RelMAE = CumRAE = MAE/MAE^* \tag{3.20}$$

Where: MAE* = MAE *of the benchmark*

f. *Theil's U2 (U2)*

$$U2 = \frac{RMSE}{RMSE^*} \tag{3.21}$$

Where: RMSE* = RMSE *of the benchmark*

g. *Mean Absolute Scaled Error (MASE)*

$$MASE = \frac{\sum_{t=1}^{n} \left| \frac{e_t}{MAE^*} \right|}{n} \tag{3.22}$$

h. *Normalized Mean Square Error (NMSE)*

$$NMSE = \frac{\sum_{t=1}^{n}(e_t^2/\sigma_y)}{n} \tag{3.23}$$

Where: σ_y *is the standard deviation of target value*

The use of relative error can reduce the effect of low value denominator of MAPE. (Armstrong and Collopy, 1992) suggest the use of GMRAE to calibrate the parameters of a given model, and recommend MdRAE for suggest selecting among forecasting methods when using a small number of time series. However, they warned that median error measures, and also percent better measure, are not sensitive because further improvements in forecasting that series produce no change when summarizing across series. Moreover, percent better does not consider the magnitude of error. Big error and small error have the same value if they have the same case where they are better than the benchmark forecasting method.

MASE is recommended by (Hyndman and Koehler, 2006) and it is proved work well for the dice rolling problem (Kolassa and Martin, 2011). MASE is also supported by (Franses, 2016) since it has the property of Diebold and Mariano properties (Diebold and Mariano, 2002). The use of MAE* as the denominator in RelMAE and MASE make the error is equally weighted. In term of stock price prediction, it is unfavorable. For example, the forecasting error of \$ 0.2 will be much different between when the price of a stock is \$ 1 and when the price of the stock is at \$ 10. The same condition may occur in demand forecasting.

4. RESEARCH METHOD

This chapter will describe the methodology of this research including the steps of the research, the dataset, the proposed algorithm and the timetable.

4.1. Research Steps

Research steps is depicted in Figure 4.1. The research begins with problem identification. As mentioned before, there is an issue in GEP algorithm about the parameter setting especially in gene length setting. There is a need for some developments to cover that problem.

After the problem is identified, the literatures related to the problem are collected and reviewed. The literatures cover the field of GEP development and application, self-adaptive parameter setting, stock market prediction and other related literatures. After that, problem statement can be defined. This research tries to solve one parameter setting problem in GEP by designing an adaptive gene length.

The next step is creating codes for the basic GEP and Robust GEP. These codes are necessary for comparing with the proposed algorithm. Creating code can also help to understand the principle of the algorithm. The code will be built in Python language. Python has many useful libraries in artificial intelligence field and provides a function to convert string into function which is necessary for the decoding process.

The proposed ALGEP and the proposed genetic operators is built in the same platform. ALGEP be compared with the basic GEP and Robust GEP using three SRPs. The proposed ALGEP and the testing functions will be described in the next sub-section. The validation of the algorithm is done for every code function by checking the return of the function with various inputs including extreme possible input.

Combination of the proposed adaptive length GEP and the proposed genetic operators will be used to solve a real complex problem which is stock market prediction. The performance of the algorithm will be compared with the basic GEP and, a well-known prediction tool, ANN. At first, the algorithms will be applied to predict IDX Composite Index. After that, the algorithms will be applied to predict a LQ45 stocks price. In this step, ALGEP and ANN is coded using MATLAB. The reason of using MATLAB is to make the computation time faster since MATLAB has vectorization and function handle feature that can make the computation time faster especially for large data.

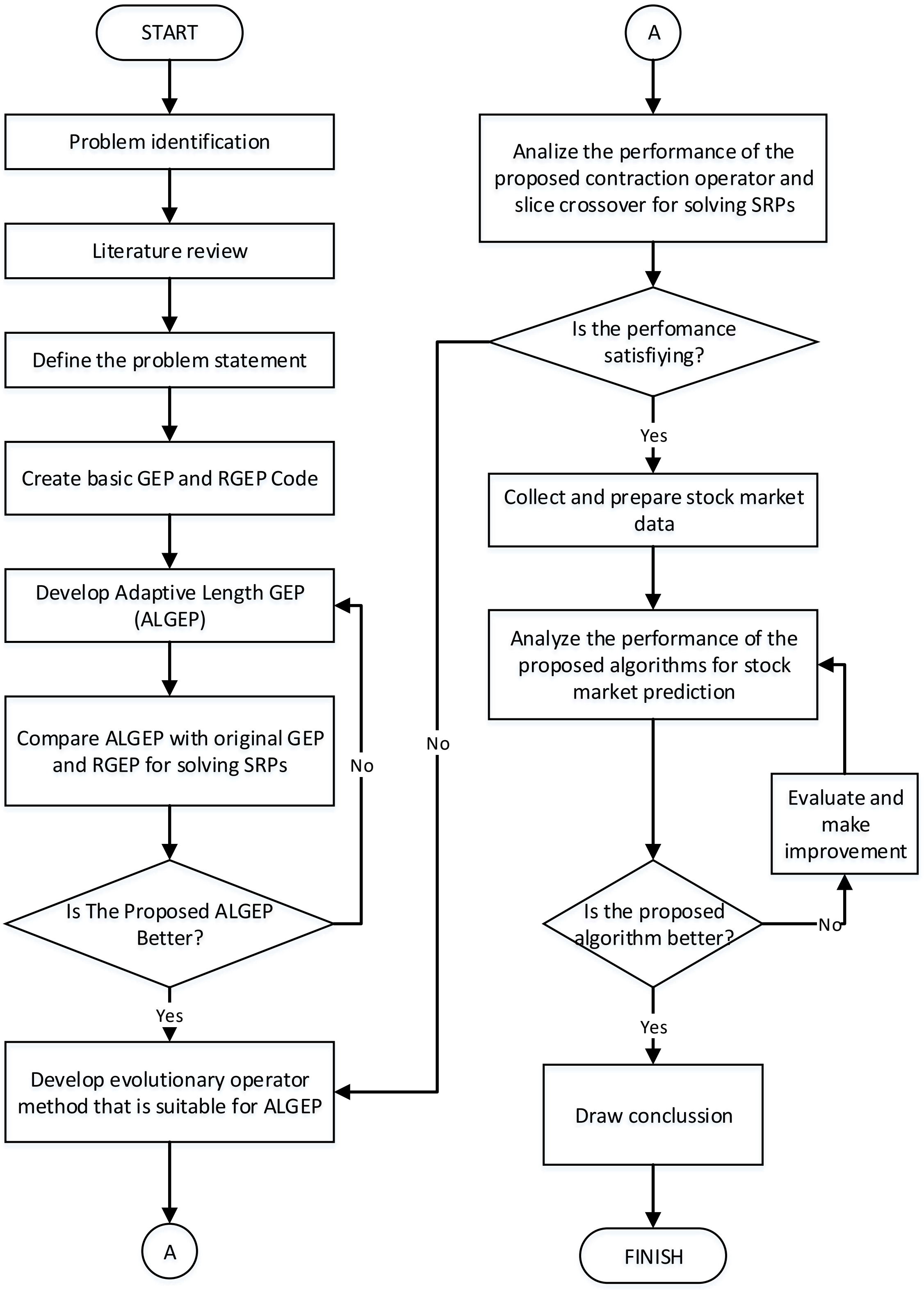

START
Problem identification
Literature review
Define the problem statement
Create basic GEP and RGEP Code
Develop Adaptive Length GEP (ALGEP)
Compare ALGEP with original GEP and RGEP for solving SRPs
Is The Proposed ALGEP Better?
No
Yes
Develop evolutionary operator method that is suitable for ALGEP
A
A
Analize the performance of the proposed contraction operator and slice crossover for solving SRPs
Is the perfomance satisfiying?
No
Yes
Collect and prepare stock market data
Analyze the performance of the proposed algorithms for stock market prediction
Evaluate and make improvement
Is the proposed algorithm better?
No
Yes
Draw conclussion
FINISH

Figure 4.1. Flowchart of the Research Method

4.2. The Proposed Adaptive Length GEP (ALGEP)

ALGEP is a development of RGEP with an adaptive gene length and some revised evolutionary operators. The process of ALGEP is the same as original GEP except for the encoding, adaptive gene length, and the proposed operators those will be described in the next subsections. In ALGEP, the gene length is not fixed and could be varied for each individual in the population. When generating the initial population, the length of a gene is randomly chosen from a specified range. The evolutionary process adjusts the gene length to ensure that the individual with the proper length survives.

Figure 4.2 shows an example of a population in ALGEP. Each individual may have a different gene length in the specified range. Users could also choose whether the first allele is always a function or any random from function and terminal. When the first allele is chosen as a function, it is replaced by a random function in the initial population or along the evolutionary process. For instance, individual 2 in Figure 4.2 may be changed to "+ a * b +". The function "+" in the first allele is chosen randomly from the function set.

Individual 1:	*sqrt*	*	+	*	*a*	*	*c*	*d*	*a*	*	-	*b*
Individual 2:	*a*	*b*	*	*b*	+							
Individual 3:	-	*	*b*	*a*	/	*c*	*					
Individual 4:	*b*	+	+	/	/	*a*	*a*	*b*				
Individual 5:	/	/	+	*a*	*a*	*b*						

Figure 4.2. Example of a population in GEP

The following is pseudocode for generating initial population in ALGEP.

```
Procedure: Generating_initial_population
Input:
        n         : population size
        f         : set of functions
        t         : set of terminals
        l_min     : minimum gene length
        l_max     : maximum gene length
        f_first   : True, if the first allele should be a function
                    False, otherwise
Output:
        pop       : a population containing n individuals

Description:
        1. For i = 1 to n:
              a. k = a random integer between l_min to l_max
              b. generate pop[i] as a random k-combination from set(f + t)
              c. If (f_first == True) and (pop[i][1] is not a member of set(f)):
                  Choose randomly pop[i][1] from set(f)
        2. Return pop
```

4.3. The Proposed Length Adjustment Operator

The length adjustment operator could delete a bit of the chromosome to shorten it or insert a bit to extend it with a specified probability, pcon. A random number is generated in the length adjustment process. The length adjustment is conducted when the random number is less than or equal to pcon. Moreover, a simple rule is used to decide whether the individual is to be contracted or expanded. When the random number generated is less than pcon /2, a bit in the random position is deleted, and the gene length is shortened. Otherwise, a bit of a random string is inserted in a random position. This operation is expected to result in a proper gene length in the evolutionary process. Figure 4.3 shows an example of the length adjustment operation of an individual. In the example, a bit in position #5 is deleted, shortening the gene length from 10 to 9 bits. A random bit could also be inserted in position 5 to extend the gene length to 11 bits.

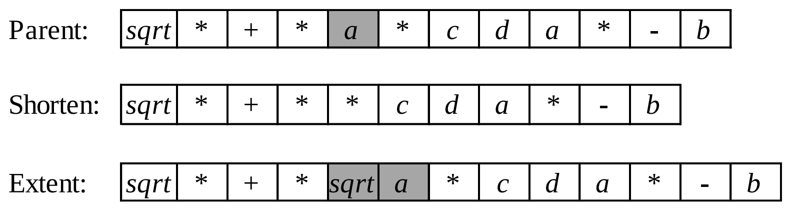

Parent: sqrt * + * a * c d a * - b
Shorten: sqrt * + * * c d a * - b
Extent: sqrt * + * sqrt a * c d a * - b

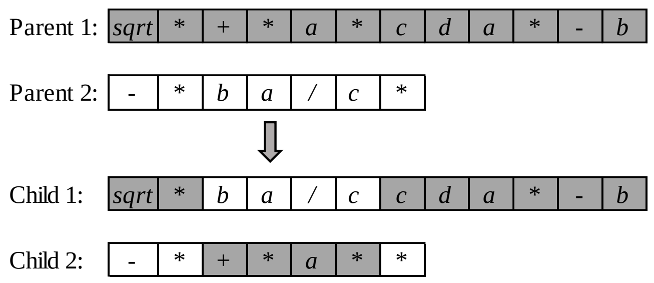
Figure 4.3. Example of a length adjustment

The following is pseudocode for length adjustment operation.

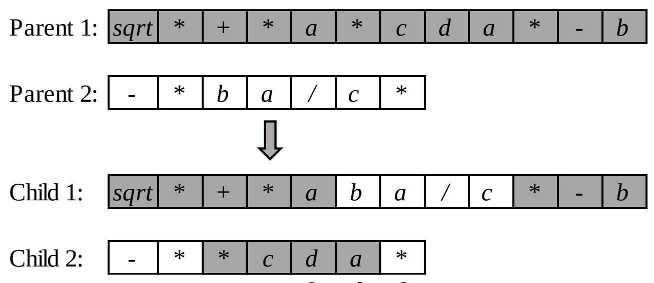
```
Procedure: length adjustment
Input:
    indiv   : individual which its length to be shorten or lengthen
    f       : set of functions
    t       : set of terminals
    l_min   : minimum gene length
    l_max   : maximum gene length

Output:
    indiv   : individual after shorten or lengthen

Description:
    1. length = length of indiv
    2. rand = random from 0 to 1
    3. c_point = round((0.5 – rand)*2*length)
    4. if c_point >= 0 and length < l_max:
            insert a random choice from $set(f + t)$ into $indiv$ at $c\_point$ position
    5. else if $c\_point < 0$ and $length > l\_min$:
            delete a bit of $indiv$ at $absolute(c\_point)$ position
    6. return $indiv$
```

4.4. The Proposed Slice Crossover

The gene length in ALGEP may be varied for each individual, though this makes the crossover process ineffective. Figure 4.4 shows a two-point crossover for two individuals with gene lengths of 12 and 7 at positions 3 and 6, respectively. Using two-point crossover, the position from bit 8 to 12 in parent 1 cannot become the subject of crossover. This condition could reduce the probability of producing a better child since it might be a good gene in that position.


Parent 1: sqrt * + * a * c d a * - b
Parent 2: - * b a / c *
Child 1: sqrt * b a / c c d a * - b
Child 2: - * + * a * *
Figure 4.4. Example of a two-point crossover in ALGEP

Slice crossover was proposed to overcome this problem, where the crossover position in parents 1 and 2 could be different and chosen randomly. For instance, Figure 4.5 shows that crossover points in parents 1 and 2 are at positions 6 to 9 and 3 to 6, respectively. Therefore, all positions of the parents could become the subjects of crossover and increase the probability of producing good children.


Parent 1: sqrt * + * a * c d a * - b
Parent 2: - * b a / c *
Child 1: sqrt * + * a b a / c * - b
Child 2: - * * c d a *
Figure 4.5. Example of a slice crossover

The following is pseudocode for slice crossover.

```
Procedure: slice crossover
Input:
        parent_1        : individual 1 to be mated
        parent_2        : individual 2 to be mated

Output:
        offspring_1     : new individual as a result of crossover
        offspring_2     : another new individual as a result of crossover

Description:
        1. offspring_1 = parent_1
        2. offspring_2 = parent_2
        3. l = random int from 1 to min(length of parent_1, length of parent_2) - 1
        4. pos_1 = random int from 1 to ((length of parent_1) – 1)
        5. pos_2 = random int from 1 to ((length of parent_2) – l)
        4. offspring_1[pos_1 : pos_1 + l] =
                                        parent_2[pos_2 : pos_2 + l]
        5. offspring_2[pos_2 : pos_2 + l] =
                                        parent_1[pos_1 : pos_1 + l]
        6. return offspring_1, offspring_2
```

Another proposed crossover is flexi-slice crossover. In this method, the length of slice from each parent might be different. As in Figure 4.6, parent 1 gives a 4-length slice from gene 6-9 to child 2 while parent 2 only gives 2-length slice to child 1. Using this method, the length of the children can be different from their parent and hopefully the optimal gen length can be achieved during evolutionary process. This crossover method will be applied for stock market prediction.

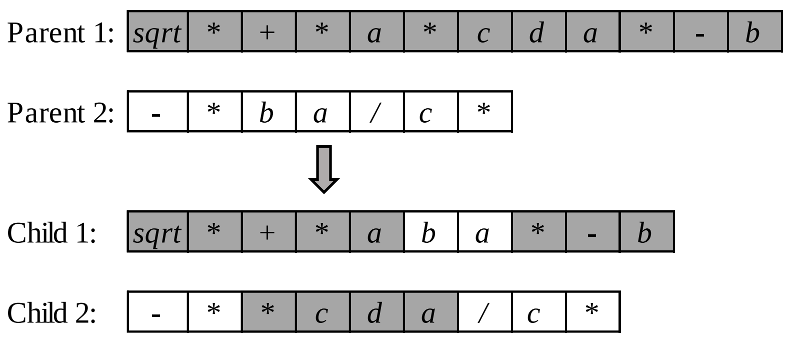

Parent 1: sqrt * + * a * c d a * - b
Parent 2: - * b a / c *
Child 1: sqrt * + * a b a * - b
Child 2: - * * c d a / c *

Figure 4.6. Example of a flexi-slice crossover

The following is pseudocode for slice crossover.

```
Procedure: flexi-slice crossover
Input:
        parent_1        : individual 1 to be mated
        parent_2        : individual 2 to be mated
        l_min           : minimum gene length
        l_max           : maximum gene length

Output:
        offspring_1     : new individual as a result of crossover
        offspring_2     : another new individual as a result of crossover

Description:
        1. offspring_1 = parent_1
        2. offspring_2 = parent_2
        3. cond = True
        4. while cond == True:
           # to make sure that final length of the offsprings are in range l_min and l_max
              a. l_1 = random int from 1 to length of parent_1 - 1
              b. l_2 = random int from 1 to length of parent_2 - 1
              c. l_offspring_1 = length of parent_1 – l_1 + l_2
              d. l_offspring_2 = length of parent_2 – l_2 + l_1
              e. if l_offspring_1 and l_ofspring_2 between l_min and l_max:
                    cond = False
        5. pos_1 = random int from 1 to ((length of parent_1) – l_slice_1)
        6. pos_2 = random int from 1 to ((length of parent_2) – l_slice_2)
        7. slice_1 = parent_1[pos_1 : pos_1 + l_1]
        8. slice_2 = parent_2[pos_2 : pos_2 + l_2]
        9. delete offspring_1[pos_1 : pos_1 + l_1] and insert with slice_2
        10. delete offspring_2[pos_2 : pos_2 + l_2] and insert with slice_1
        11. return offspring_1, offspring_2
```

4.5. The Proposed Constant Creation

GEP, and also RGEP or ALGEP, can create simple constant just using basic functions and terminals. For example, to create *"2x + 3"* can be produced using *"x + x + (x + x + x)/x"*. However, to produce more complex equation, some development is needed. Ferreira (2006) uses random numerical constants (RNC) to build complex constant. **B**esides head and tail, another domain is introduced, called Dc (domain of constant). In addition, a set of random is also used, called RNC. Figure 4.7 is an example of GEP-RNC chromosome with its related RNC in Figure 4.8.

Head							Tail								Dc							
+	?	*	+	?	*	*	a	a	a	?	?	a	a	a	6	8	0	8	3	2	9	5

Figure 4.7 Example of a chromosome in GEP-RNC

	0	1	2	3	4	5	6	7	8	9
RNC:	0.611	1.184	2.449	2.98	0.496	2.286	0.93	2.305	2.737	0.7555

Figure 4.8 Example of a NRC

The chromosome in Figure 4.7 is then translated into equation tree. The translation process is shown in Figure 4.9. First, the chromosome is translated into equation tree just like usual, then the sign "?" is replaced by its related Dc, and then the number is replaced by the related index of RNC. Finally, the equation tree is translated to an equation, "0.93 * (2.737 + ((a+a) + (a+0.611)))" in this case.

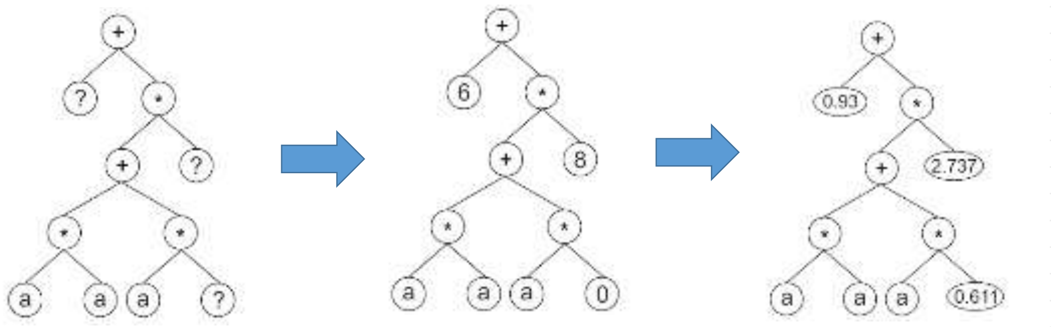


Figure 4.9 Equation Tree of the GEP-RNC example

This research proposes a simpler method to deal with constant creation. Instead of using Dc and RNC, the proposed constant creation uses numerical constant directly into the chromosome. The constant numbers are randomly generated in the gene at the initial population generation. The constant gene can be adjusted during evolutionary process using mutation and constant mutation those will be described next. Figure 4.10 shows how the same equation above is decoded into a chromosome using the proposed constant creation. This chromosome can be easily translated into the equation directly. To create an initial population with constant in computer programs, just add "C" letter in the terminal and after the population is generated, replace each "C" letter with random generated from specified range.

+	0.93	*	+	*	a	a	*	a	0.611	0.2737

Figure 4.10 Example of a chromosome in the proposed constant creation

The following is pseudocode for constant creation.

```
Procedure: Constant creation
Input:
        f       : set of functions
        t       : set of terminals
        c_min   : minimum constant
        c_max   : maximum constant
        f_first : True, if the first allele should be a function
                  False, otherwise
Output:
        ind     : individual

Description:
        1. set(t) = set(t + “C”)
        2. Generate chromosome with input f and t
        3. For i = 1 to number of “C” in chromosome:
               a. k = a random number between c_min to c_max
               b. Replace “C” with k
        4. Return chromosome
```

4.6. Mutation

In the proposed algorithm, mutation can be done in the entire gene. User can define how many points of mutation, one-point or two point for example. During this operation, a chosen allele is mutated to another value that is randomly chosen. It can be function, terminal or random value. For example, bit 10 in Figure 4.10, that is “0.611” can be mutated to any function, terminal or another random number.

4.7. Constant Mutation

This proposed operator is to change the chosen random number a little bit. The change can be up or down in a specified range. For example, if user specifies maximum constant change to 10% then a chosen random number can be increased up to 10% or decreased up to -10%. This operation is only targeted to alleles with random number. For example, only gene 2, 10 and 11 in Figure 4.10 will be subject to constant mutation. The mutation point will be chosen randomly between these three points. For instance, if gene 2 is chosen, then gene 2, that is “0.93” can be increased or decreased up to 10%.

The following is pseudocode for constant mutation.

```
Procedure: constant_mutation
Input:
        indiv           : individual to be mutated
        max_rate        : max mutation rate

Output:
        indiv   : individual after shorten or lengthen

Description:
        1. find index of all constant and save to ind_contant
        2. mut_point = random chosen from ind_constant
        3. rand = random from 0 to 1
        3. indiv[mut_point] = indiv[mut_point]*(1 + max_rate*(-1 + 2*rand))
        6. return indiv
```

4.8. SRPs for Testing functions

The effectiveness of ALGEP will be compared with original GEP and RGEP for solving 3 SRPs using standard evolutionary process of GEP. After that, the effectiveness of length adjustment operator and slice crossover will be evaluated using the same functions. The first SRP (4.1) is relatively simple polynomial function with one variable, taken from Koza (1992) while the second SRP (4.2) is a sequence induction problem from Korns (2011) which is relatively difficult to solve. The last SRP is the most difficult among previous functions with three variables and complex structure, modified from Keijzer (2003). The modification of the third function is done to avoid the use of constant in the GEP since this experiment is to evaluate the basic chromosome structure and therefore avoid any other factors including constant.

$$f(x) = x^4 + x^3 + x^2 + x \tag{4.1}$$

$$f(x) = 4x^4 + 3x^3 + 2x^2 + x \tag{4.2}$$

$$f(x) = \frac{x_1 x_2}{(x_1 - 1)x_3^2} \tag{4.3}$$

For the all functions, 20 random values are generated as the training data set without noise. First function (4.1) and second function (4.2) use uniform random value in the range [-10,10]. For the third function (4.3), uniform random values in the range [-1,1] is used for variable x1 and x2, while x3 is given uniform random values in the range [1,2]. Table 4.1 shows the generated data for each SRP.

Table 4.1 Generated SRP data for training

SRP 1		**SRP 2**		**SRP 3**			
X	Y	X	Y	X1	X2	X3	Y
-7.57	2904.90	-8.12	15946.02	-0.38	0.78	0.05	74.76
1.90	25.48	6.86	9918.62	-0.99	-0.94	1.19	-0.33
7.77	4173.84	2.47	207.89	-0.09	0.89	1.28	0.05
-2.57	30.44	7.71	15631.75	0.61	0.62	0.27	-13.38
-7.57	2905.72	-2.88	216.42	-0.35	-0.37	0.53	-0.35
-1.04	0.09	1.61	45.88	-0.21	0.89	0.21	3.37
-9.47	7272.65	-4.22	1078.79	-0.65	0.53	0.41	1.27
4.27	432.44	4.85	2601.87	0.42	0.12	0.80	-0.14
-1.83	6.55	6.74	9278.87	-0.99	0.63	0.09	36.00
5.83	1392.30	-4.72	1716.66	0.44	-0.42	1.07	0.29
2.22	42.66	7.79	16304.90	0.19	0.46	1.38	-0.06
6.87	2608.06	-9.05	24810.65	-0.52	0.88	1.22	0.20
3.43	194.11	-8.30	17367.24	-0.99	0.55	1.52	0.12
6.04	1594.84	8.16	19507.45	0.40	-0.62	1.38	0.22
7.74	4119.09	8.25	20322.18	0.44	-1.00	1.47	0.37
5.68	1260.32	2.79	325.14	0.36	-0.89	1.73	0.17
-2.52	28.07	6.01	5966.64	0.37	0.55	0.46	-1.50
-6.72	1770.91	5.51	4261.97	0.85	0.95	1.20	-3.83
0.39	0.62	6.38	7484.77	0.17	0.75	1.01	-0.15
8.96	7251.14	-0.10	-0.08	0.47	0.37	0.13	-19.62

4.9. Experiment for SRP

The data will be trained using population size of 100 individuals over 100 generations. For each treatment, the evolutionary process will be repeated 100 runs. For each run, if the accuracy value is below the precision number, the run is categorized as success. In these experiments, the precision level is set to be 0.0001. Normalized mean square error (NMSE) is applied for the accuracy performance measurement. NMSE is used because it can reduce the effect of different scaling measurement by dividing the square error with target standard deviation. The equation to calculate NMSE is as Equation (3.3). The fitness function (4.4) representing the quality of the solution-*i* (f_i) is measured base on the NMSE.

$$f_i = \frac{1}{1 + NMSE_i} \tag{4.4}$$

Two experiments will be conducted. The first experiment will evaluate the performance of ALGEP compared with the original GEP and RGEP using standard operators and parameters setting, while the second experiment will evaluate the effectiveness of length adjustment operator and slice crossover. Table 4.2 gives parameter setting of Experiment 1. Note that in RGEP and ALGEP, there are no head and tail sections. The parameter first function indicates whether the first bit in the initial population is set to be function or not. If it is true, the first bit in all individuals in the initial population will be set to be a function. Roulette wheel (RW) without replacement is applied for selection method to choose the next generation. With this method, an individual cannot be chosen twice for the next generation.

Experiment 2 is conducted using ALGEP scheme (see Table 4.3). Treatment 4 uses length adjustment with the probability of 0,2, Treatment 5 uses slice crossover, and Treat 6 combines length adjustment operator and slice crossover. Note that gene length in ALGEP is varied and can be shortened or expanded using length adjustment operator and the parameter gene length in this Experiment indicate maximum gene length of the initial population.

In all experiment, only basic mathematical operators, i.e. addition (+), subtraction (-), multiplication (*) and division (/), are used to develop function. Constant will also be produced using these operators during evolutionary process. For example, to produce constant 2, an equation like (X1+X1)/X1 can be developed during the evolutionary process.

In the experiments, the number of genes in each chromosome is only one. In GEP or its variants including ALGEP, a chromosome can contain one or more genes. If the chromosome contains two or more genes, it is usually called multi-gene chromosome. Experiment 3 for stock market prediction is an example of multi-gene application as shown in Figure 4.12.

The probability of inversion and transposition are taken from Ferreira (2006), the probability of crossover and mutation and the population size are determined from common practice in genetic algorithm, while other parameters are determined by the rule of thumb. For generation replacement, roulette wheel without replacement is used to relax the selection pressure for the next generation. Using this methods, the individuals with bad fitness, which may have some good sections of gene, will still have better chance to be selected than if using roulette wheel.

The success rate for each treatment is recorded. The success rate as in (4.4) is the result of dividing the number of success run and total run times 100%. Success run means that the exact proper equation is found in the run. It is indicated by accuracy performance below the precision number.

$$Success\ rate = \frac{Number\ of\ success\ run}{Total\ number\ of\ run}\ x\ 100\% \tag{4.5}$$

Table 4.2 Design of Experiment 1

TREATMENT	**1**	**2**	**3**
ENCODING METHOD	**BASIC GEP**	**RGEP**	**ALGEP**
Population size	100	100	100
Max generation	100	100	100
Run	100	100	100
Length of head	5-51	-	-
Length of tail	l_head * (max_arg -1) +1	-	-
Length of chromosome	l_head + l_tail = 11-103	11-103	11-103
First Function	True	True	True
Probability of mutation	0.1	0.1	0.1
Probability of inversion	0.1	0.1	0.1
Probability of transposition	0.1	0.1	0.1
Probability of recombination or crossover	0.9	0.9	0.9
Parent selection method for crossover	RW	RW	RW
Selection method for next generation	RW without replacement	RW without replacement	RW without replacement
Mutation method	'two-point'	'two-point'	'two-point'
Crossover method	'two-point'	'two-point'	'two-point'
Best possible fitness (perfect solution)	1	1	1
Accuracy performance measurement	NMSE	NMSE	NMSE
Selection Range	1	1	1
Precision	0.0001	0.0001	0.0001
With constant	No	No	No
Random method for gene generation	Uniform	Uniform	Uniform
Mathematical operator	+ - * /	+ - * /	+ - * /

Table 4.3 Design of Experiment 2

TREATMENT	**4**	**5**	**6**
ENCODING METHOD	**AL-GEP**	**AL-GEP**	**AL-GEP**
Population size	100	100	100
Max generation	100	100	100
Run	100	100	100
Max length of chromosome for initial population	11-103	11-103	11-103
First Function	True	True	True
Probability of mutation	0.1	0.1	0.1
Probability of inversion	0.1	0.1	0.1
Probability of transposition	0.1	0.1	0.1
Probability of length adjustment (up and down)	0.2	0	0.2
Probability of crossover	0.9	0.9	0.9
Parent selection method for crossover	RW	RW	RW
Selection method for next generation	RW without replacement	RW without replacement	RW without replacement
Mutation method	'two-point'	'two-point'	'two-point'
Crossover method	'two-point'	'slice'	'slice'
Best possible fitness (perfect solution)	1	1	1
Accuracy performance measurement	NMSE	NMSE	NMSE
Selection Range	1	1	1
Precision	0.0001	0.0001	0.0001
With constant	No	No	No
Random method for gene generation	Uniform	Uniform	Uniform
Mathematical operator	+ - * /	+ - * /	+ - * /

4.10. Experiment for Stock Market Prediction

The proposed algorithm and the comparisons will be applied to predict stock market. At first, the algorithms will be applied to predict monthly IDX Composite Index or *Indeks Harga Saham Gabungan (IHSG)* in Bahasa Indonesia. The data is obtained from Yahoo Financial [https://finance.yahoo.com]. The maximum available data, which is from July 1997 to July 2019, will be collected. The input variable for each set is open, high, low, close, volume, change, and volume change. The output variable is the next close.

The next application is for predicting individual daily stock price of an LQ45 member. LQ45 is the collection of 45 most liquid stocks in IDX. The chosen stock is Bank BRI (BBRI) stock price since it has the biggest average daily volume of all LG45 members. The data is also obtained from Yahoo Financial. Same with the IDX Composite

Index prediction, the input variable for each data set is open, high, low, close, volume, change, and volume change while the output variable is the next close. Table 4.4 shows the design of experiment.

Table 4.4 Design of Experiment 3

PARAMETER	**TREATMENT**			
	1	**2**	**3**	**4**
Method	ALGEP	ANN	ALGEP	ANN
Data	Monthly IHSG	Monthly IHSG	Daily BBRI	Daily BBRI
Data input	Open, high, low, close, volume, change, and volume change	Open, high, low, close, volume, change, and volume change	Open, high, low, close, volume, change, and volume change	Open, high, low, close, volume, change, and volume change
Data output	Next close	Next close	Next close	Next close
Set of data	296	296	4253	4253
Number of gen	20000	20000	20000	20000
Structure	4 genes per chrom 5-20 gene length	3 hidden layers (12-6-3 nodes)	4 genes per chrom 5-20 gene length	3 hidden layers (12-6-3 nodes)
Method parameters	Crossover method: Flexi-slice Probability of constant mutation: 10%	Learning method: Back propagation Activation function: binary sigmoid	Crossover method: Flexi-slice Probability of constant mutation: 10%	Learning method: Back propagation Activation function: binary sigmoid
Stag step for stopping	1000	1000	1000	1000
Validation data proportion	20%	20%	20%	20%
Data division method	Random	Random	Random	Random
Performance measure	MAPE	MAPE	MAPE	MAPE
Repetition	3	3	3	3

The structure of ANN contain 7 inputs, 3 hidden layers and one output as depicted in Figure 4.11. The number of nodes in the hidden layer 1, 2, 3 are 12, 6 and 3 respectively. For all node, the activation function is binary sigmoid. Backpropagation is applied as the training method. Initial weights are generated randomly. Learning rate is 0.9, momentum is 0.2 and sigmoid multiplier is 1.

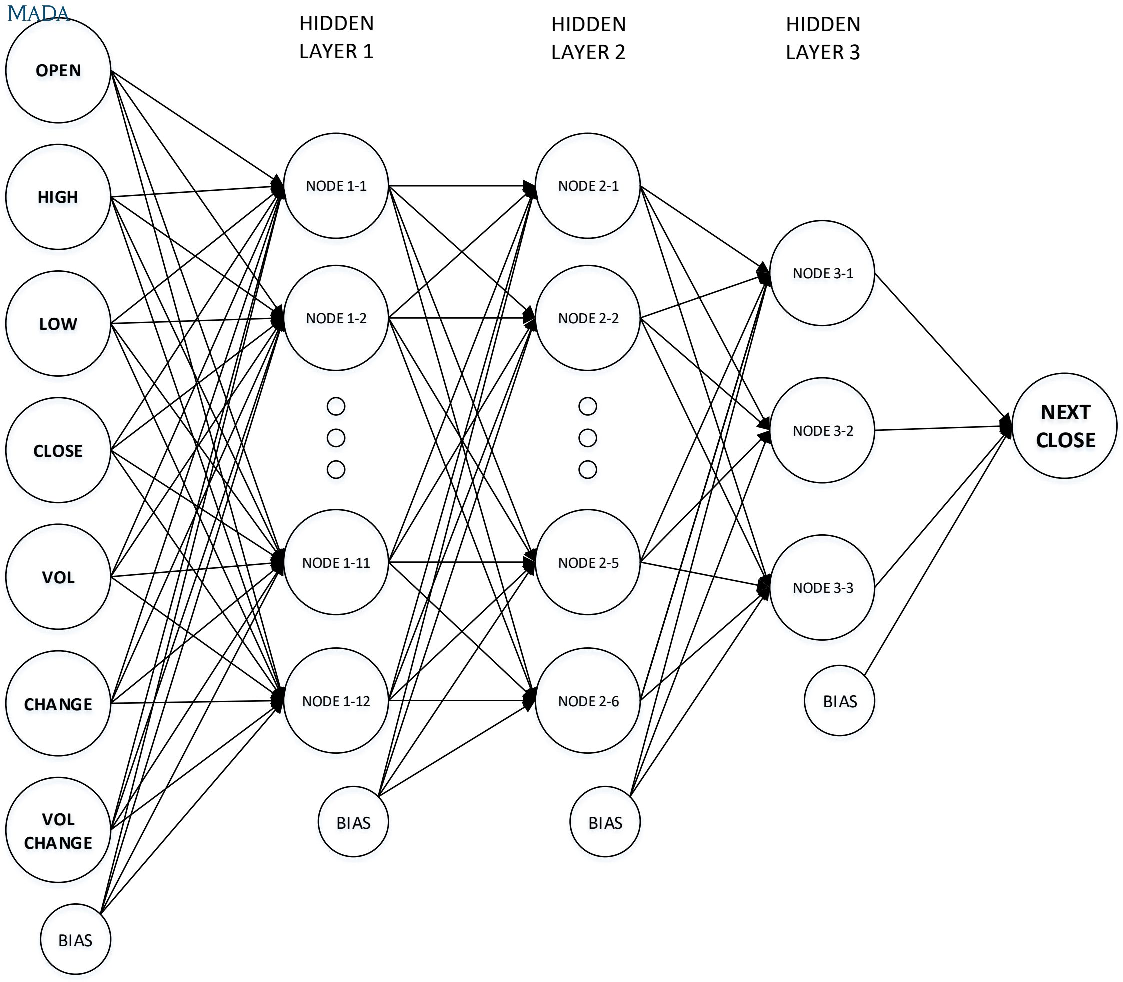


Figure 4.11 The network structure of ANN

After the data obtained, the next step is cleaning process. Some periods with a blank data is erased. Other periods with error indication, such as too small volume, is also eliminated. The periods those are effected by the erased periods will also be eliminated. After cleaning process, 296 sets of monthly IHSG data and 4253 sets of daily BBRI data are obtained. After cleaning, the data is normalized to between 0.1 - 0.9. Equation (4.6 shows how the data is normalized.

$$Normalized\ data = 0.1 + 0.8 * \left(\frac{Raw\ data - Lowest\ data}{Largest\ data - Lowest\ data}\right) \qquad (4.6)$$

The data is divided into training data and validation data. The training data is 80% of the data which is chosen randomly, and the rest will be used as validation data. In each repetition, the training data set and validation data set is the same for ALGEP and ANN methods. The data training and validation data set will be chosen randomly again in the next repetition.

To measure the accuracy performance of the algorithms, MAPE is used. MAPE is a good measure of forecasting accuracy if the historical value does not contain zero number or close to zero. Moreover, MAPE is easy to be interpreted.

In this experiment, all features of the proposed algorithm are used except length adjustment operator since this operator does not give any significant performance increase in the previous experiment. Instead of using one gene, ALGEP method in this experiment will use 4 genes in one chromosome or individual as in Figure 4.12. Length of each gene may be varied and the equation result of each genes are then combined using "+" operator in the encoding process. For crossover, flexi-slice method is applied. The proposed constant creation and constant mutation operator are also applied.

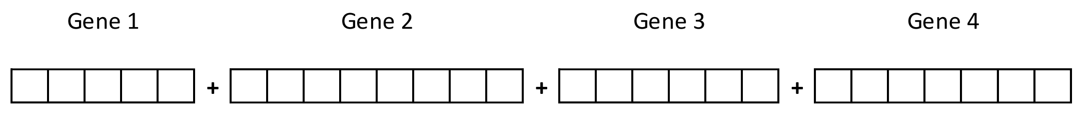


Figure 4.12 The structure of a chromosome in Experiment 3

5. RESULTS AND DISCUSSION

5.1. Experiment Setting

To analyze the performance of ALGEP, the first experiment is conducted. In the first experiment, the algorithm will be applied to solve 3 SRPs and compared with basic GEP and RGEP as like described in the previous section. The second experiment is held to analyze the performance of length adjustment operator and slice crossover. The algorithm will also be applied to solve the same SRPs. The algorithm will be coded in Python using Spyder 4.0, a Python IDE for scientific use. The code can be seen in Attachment 1.

Experiment 1 will use 3 algorithms, Basic GEP, RGEP and ALGEP. Each algorithm will be applied to solve 3 SRPs. Each SRP will be solved using gene length 11-103 with 4-step interval, meaning that gene length for first running is 11, then 15 and so on until 103. Each gene length for each algorithm and SRP will be run 100 times and the success rate percentage will be recorded. There will be 3x3x25x100 = 22,500 running for first experiment.

The second experiment will run as the same as experiment 1, therefore, the number of running for experiment 1 and 2 is 45,000 times. The experiments are conducted in Simulation and Computation Laboratory, Department of Mechanical and Industrial Engineering, Faculty of Engineering, Universitas Gadjah Mada on 25-31 March 2021. For running the experiments, computers with Core i5 Gen 6 processor, Ram 16GB, Intel Motherboard with HDD Drive 500GB are used.

Experiment 3 is conducted to compare ALGEP with a well-known deep learning method which is deep learning ANN. The experiment for comparing ALGEP and ANN is coded using MATLAB. The reason of using MATLAB is to make the computation time faster since MATLAB has vectorization and function handle feature that can make the computation time faster especially for large data. The experiment is run on 31 January – 2 February 2022.

5.2. Result

5.2.1. Experiment 1

Success rate for each treatment in Experiment 1 is shown in Table 5.1. The result indicate that ALGEP perform better to solve the SRPs in term of success rate compare to basic GEP and RGEP. As in the previous study (Ryan and Hibler, 2011), RGEP is perform better than basic GEP. Figure 5.1, Figure 5.2 and Figure 5.3 also support the superiority of ALGEP among GEP and RGEP.

Table 5.1 Success rate (%) of Experiment 1

Chrom Length	**SRP 1**			**SRP 2**			**SRP 3**		
	Treatment			**Treatment**			**Treatment**		
	GEP	**RGEP**	**ALGEP**	**GEP**	**RGEP**	**ALGEP**	**GEP**	**RGEP**	**ALGEP**
11	0	0	42	0	0	0	0	4	2
15	0	13	58	0	0	0	0	2	7
19	8	31	61	0	0	0	0	1	2
23	13	28	54	0	0	0	1	3	2
27	29	41	64	0	0	1	1	1	4
31	24	32	68	1	1	2	1	4	5
35	27	44	68	1	1	5	0	1	3
39	31	48	66	2	3	4	1	2	4
43	37	52	75	1	4	7	0	1	4
47	43	58	68	1	3	6	0	2	4
51	44	53	63	5	6	8	1	0	2
55	49	59	67	1	8	8	1	0	1
59	37	53	74	6	7	10	0	2	5
63	38	57	74	5	13	11	1	0	2
67	39	55	78	6	5	14	1	0	3
71	40	68	62	8	14	19	2	0	3
75	42	59	78	8	12	15	0	1	3
79	45	67	81	8	12	23	0	0	3
83	37	69	74	7	14	19	0	2	4
87	36	69	67	6	19	21	0	1	0
91	42	64	83	9	16	16	0	0	4
95	35	62	87	8	13	21	0	0	4
99	28	59	83	9	16	21	0	0	3
103	28	58	83	6	21	23	1	0	6
AVG	**31.33**	**49.96**	**69.92**	**4.08**	**7.83**	**10.58**	**0.46**	**1.13**	**3.33**

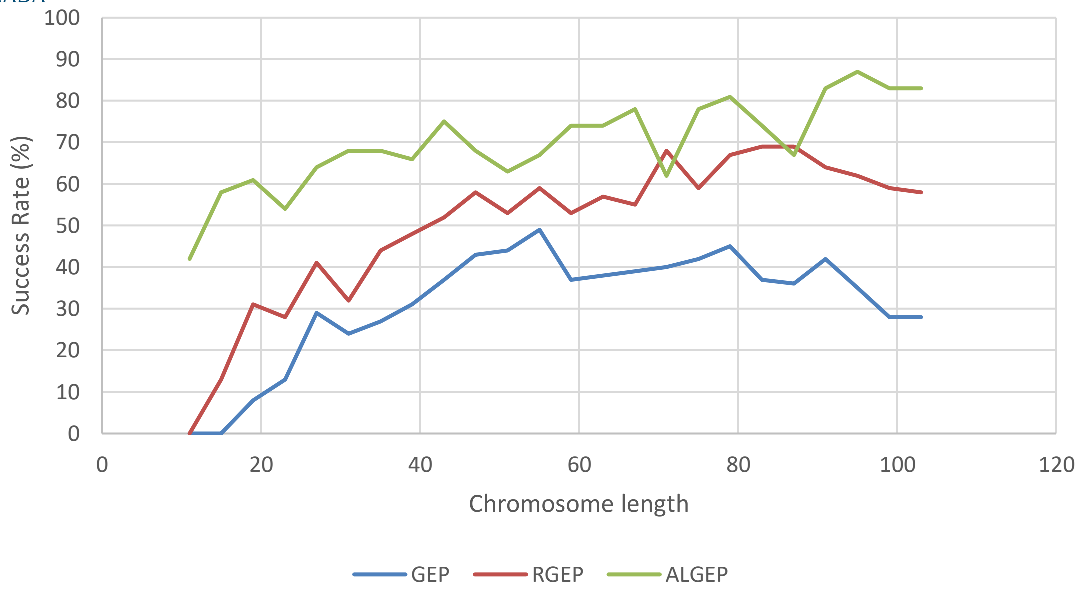


Figure 5.1 Success Rate of GEP, REG and ALGEP for solving SRP1

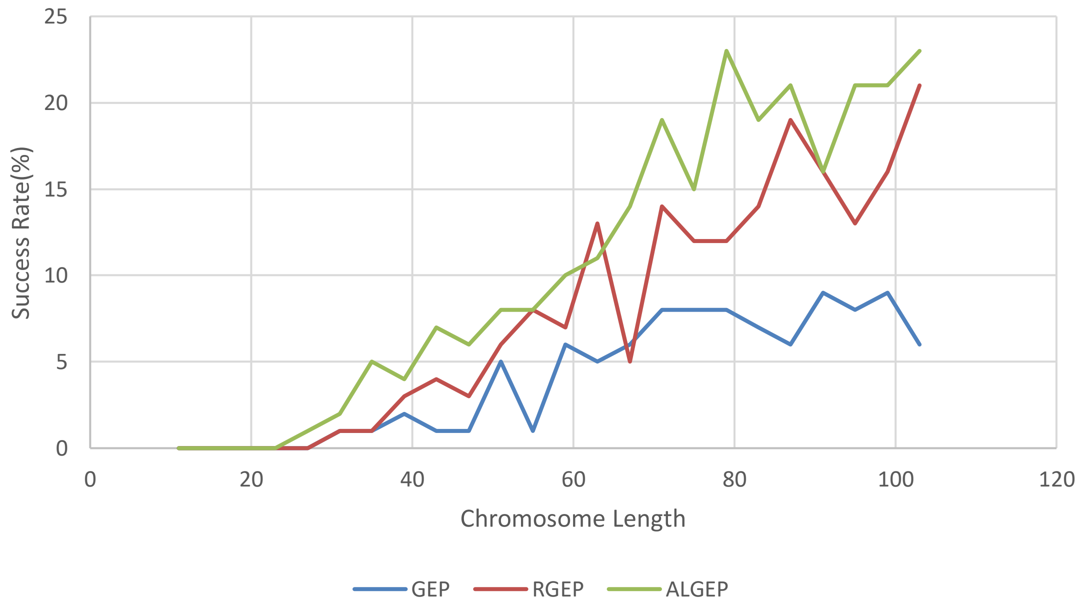


Figure 5.2 Success Rate of GEP, REG and ALGEP for solving SRP2

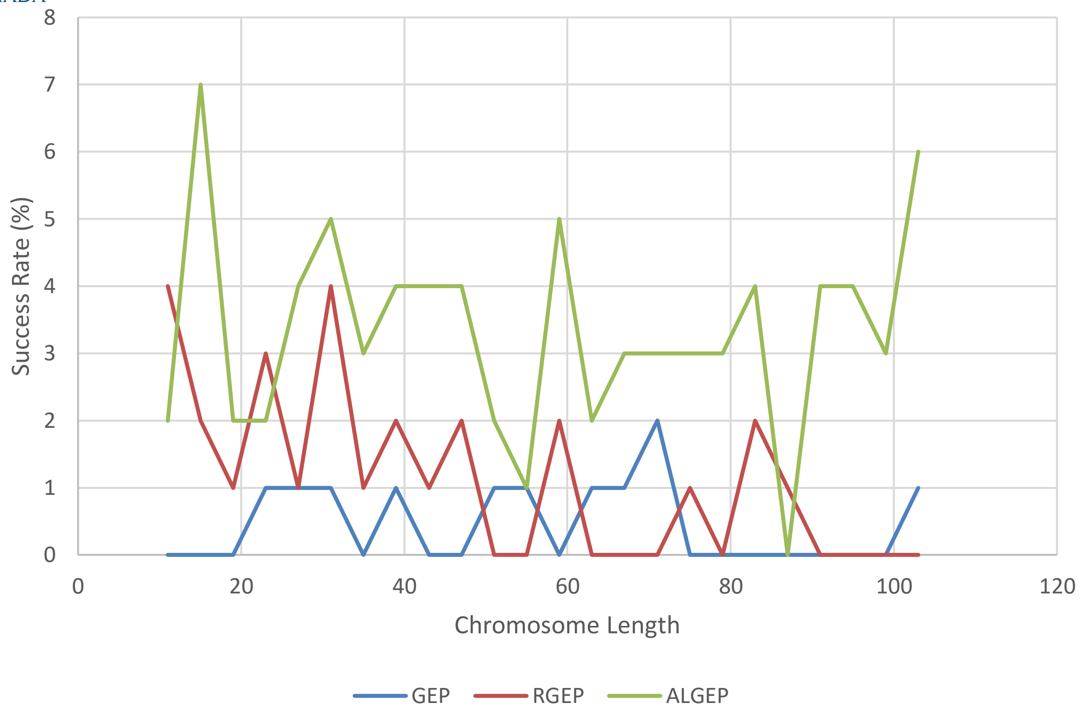


Figure 5.3 Success Rate of GEP, REG and ALGEP for solving SRP3

Repeated Measures ANOVA (RMA) is conducted to test the different between GEP, RGEP and ALGEP performance. RMA, also known as within-subjects ANOVA, is suitable for paired samples with 3 or more treatment. Before calculating the significance level in RMA, the homogeneity of the variance (also called sphericity) within all possible pairs need to be assess using Mauchly's test. If the P value of Mauchly's test is insignificant ($P \geq 0.05$), equal variances are assumed and P value for RMA would be taken from sphericity assumed test. If the P value is less than 0.05, meaning that the variance is not homogeneous, epsilon (ε) value will determine the statistical method to calculate P value for RMA. When $\varepsilon < 0.75$, Greenhouse-Geisser method is suitable to draw the conclusion, otherwise, Huynh-Feldt method will be used. If the RMA is significant, the next step is post-hoc test which is pair-wise comparison contains multiple paired t tests with a Bonferroni correction. The Bonferroni correction is used to adjust the significance level to control the overall probability of a Type I error for multiple hypothesis tests. To adjust the significance level, the significance level (α) for a single test is divided by the number of tests (n). Table 5.2 shows that there is a significant different of success rate among GEP, RGEP and ALGEP.

Table 5.2 RMA test for Experiment 1

SRP	P value of Mauchly's test	Statistical method used	P value for RMA	Differences
SRP 1	0.152	Sphericity Assumed	0.000	Significant
SRP 2	0.015	Huynh-Feldt	0.000	Significant
SRP 3	0.51	Sphericity Assumed	0.000	Significant

Since the RMA test indicates the significant different, the next step is conducting multiple paired t-test with Bonferroni correction. As shown from Table 5.3, there are significant different of success rate between GEP, RGEP and ALGEP where the success rate of ALGEP algorithm is significantly bigger than the other algorithms.

Table 5.3 Paired t-test with Bonferroni correction for Experiment 1

SRP	Success Rate Mean Comparison	p-value	Significance (α=0.05/3)
SRP 1	RGEP > GEP	1.65E-10	Significant
	ALGEP > GEP	1.08E-14	Significant
	ALGEP > RGEP	3.83E-08	Significant
SRP 2	RGEP > GEP	1.65E-10	Significant
	ALGEP > GEP	1.08E-14	Significant
	ALGEP > RGEP	3.83E-08	Significant
SRP 3	RGEP > GEP	1.65E-10	Significant
	ALGEP > GEP	1.08E-14	Significant
	ALGEP > RGEP	3.83E-08	Significant

5.2.2. Experiment 2

Success rate for each treatment in Experiment 2 is shown in Table 5.1. The average of success rate indicates that application of slice crossover on ALGEP makes the performance of ALGEP better. However, the proposed length adjustment operator does not give good performance as expected. Compared to basic ALGEP, the addition of length adjustment operator does not give a significant effect as seen in Figure 5.4 - Figure 5.6. The combination of length adjustment operator and slice crossover also does not increase the performance of ALGEP compare to slice crossover alone. Table 5.6 clearly shows that slice crossover can increase the performance of ALGEP significantly in term of success rate, while length adjustment operator does not give a significant effect.

Table 5.4 Success rate (%) of Experiment 2

Chrom Length	**SRP 1**			**SRP 2**			**SRP 3**		
	Treatment			**Treatment**			**Treatment**		
	Contr. Opr.	**Slice Crossover**	**Comb.**	**Contr. Opr.**	**Slice Crossover**	**Comb.**	**Contr. Opr.**	**Slice Crossover**	**Comb.**
11	40	88	85	0	0	0	5	3	8
15	56	94	97	0	0	0	4	10	5
19	61	99	100	0	7	6	3	6	13
23	62	100	98	0	23	21	5	8	12
27	58	97	100	0	38	37	3	13	14
31	65	100	98	2	41	44	4	14	13
35	69	99	99	6	52	42	7	7	10
39	72	97	99	10	49	57	7	8	8
43	74	94	96	7	57	59	3	12	8
47	66	94	95	9	49	62	5	11	13
51	79	97	94	10	62	61	5	8	9
55	80	96	97	11	70	58	5	7	6
59	73	96	98	12	59	51	0	7	10
63	75	98	95	17	60	68	3	8	7
67	74	99	96	21	66	63	4	8	4
71	78	98	93	20	59	68	0	3	4
75	75	96	96	16	61	72	3	3	9
79	78	100	99	16	61	58	1	8	8
83	69	96	98	18	74	71	1	5	11
87	77	94	98	24	70	56	1	7	9
91	72	97	92	20	67	68	4	5	4
95	76	97	93	20	52	61	0	7	8
99	82	91	95	24	68	63	4	4	2
103	77	94	98	25	56	61	1	7	6
AVG	**70.33**	**96.29**	**96.21**	**12.00**	**50.04**	**50.29**	**3.25**	**7.46**	**8.38**

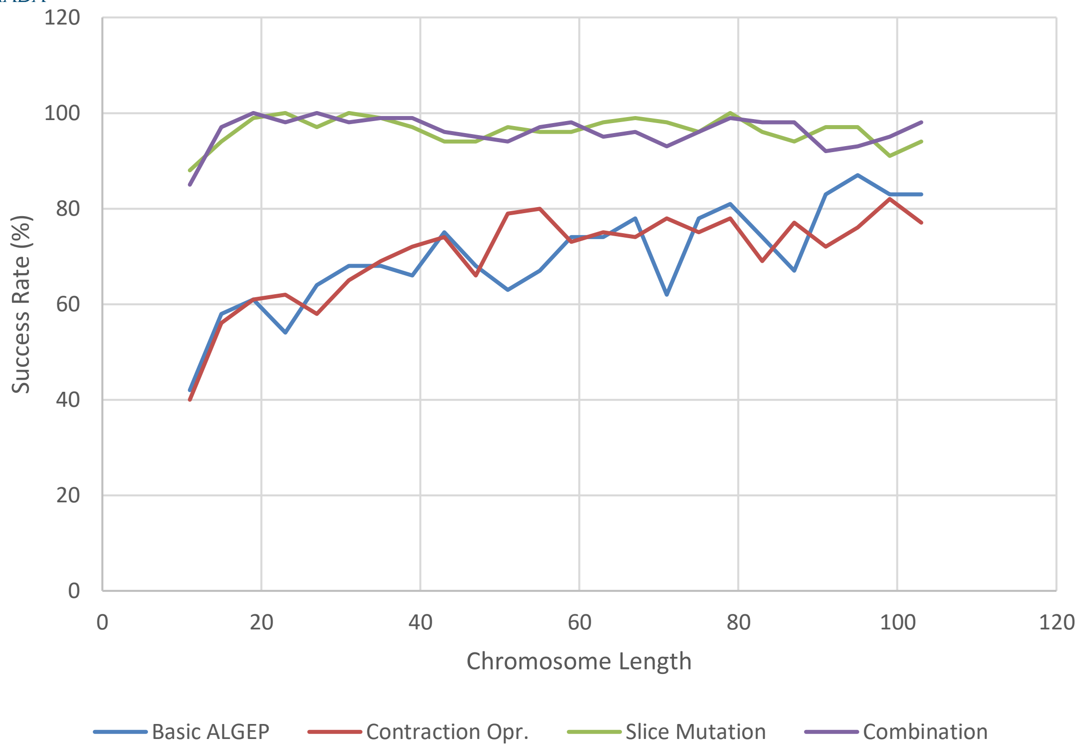


Figure 5.4 Performance of length adjustment operator and slice crossover for SRP 1

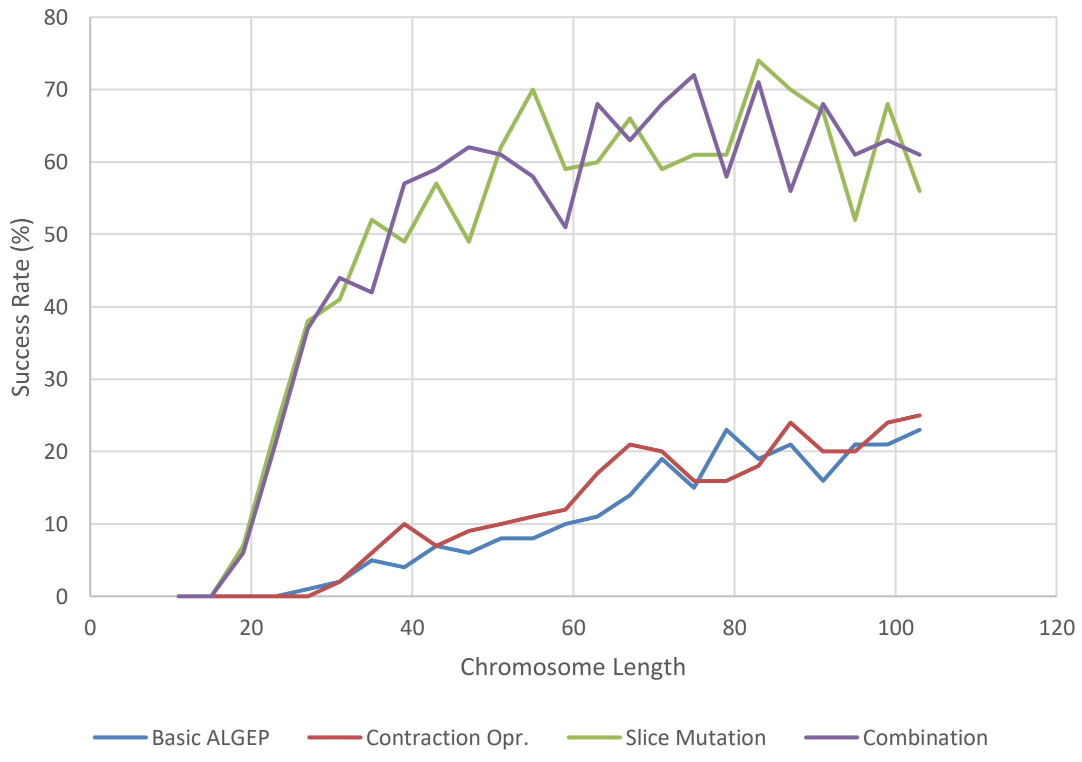


Figure 5.5 Performance of length adjustment operator and slice crossover for SRP 2

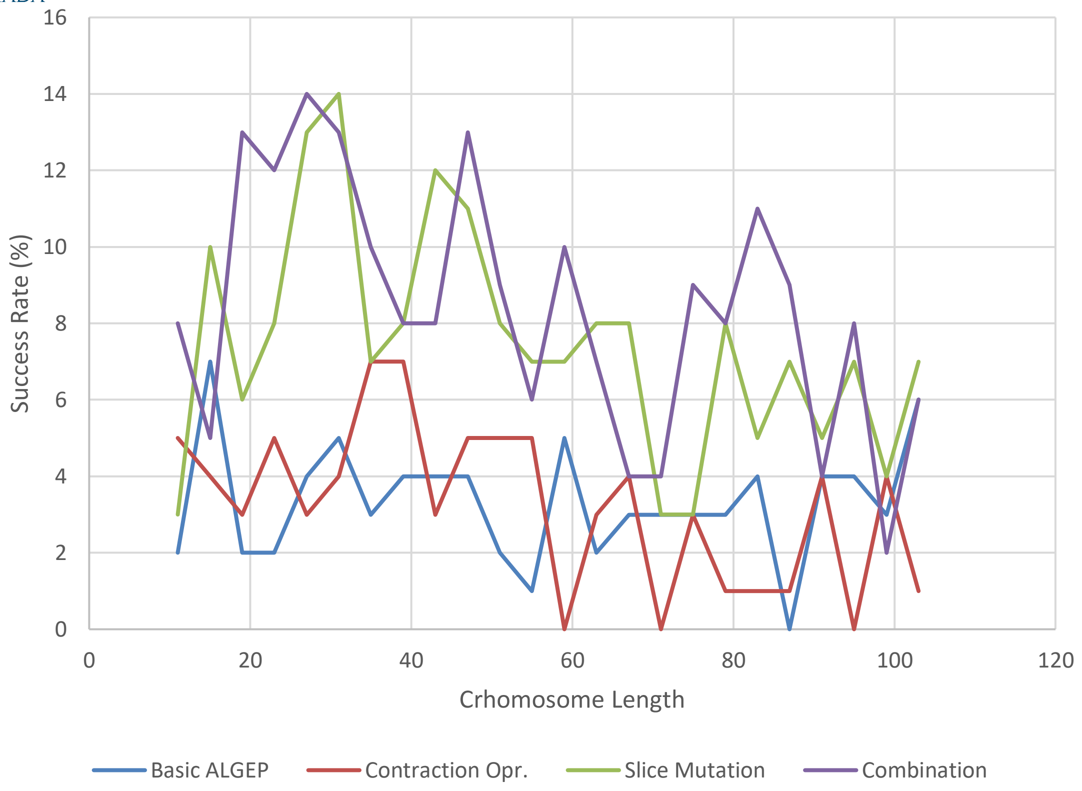


Figure 5.6 Performance of length adjustment operator and slice crossover for SRP 3

The RMA test indicates that there are significant different of success rate among treatment as in Table 5.5. Multiple paired comparison using paired t-test, as in Table 5.6, shows that the use of Slice Crossover make significant increasing in performance of ALGEP for all SRP’s, while the use of length adjustment operator does not make any significant impact.

Table 5.5 RMA test for Experiment 2

SRP	P value of Mauchly's test	Statistical method used	P value for RMA	Differences
SRP 1	0.000	Greenhouse-Geisser	0.000	Significant
SRP 2	0.000	Greenhouse-Geisser	0.000	Significant
SRP 3	0.484	Sphericity Assumed	0.000	Significant

Table 5.6 Paired t-test with a Bonferroni correction for Experiment 2

SRP	Success Rate Mean Comparison	p-value	Significance (α=0.05/5)
SRP 1	Length adjustment Opr. > Basic ALGEP	0.79	Not Significant
	Slice Crossover > Basic ALGEP	1.26E-11	Significant
	Combination > Basic ALGEP	1.17E-11	Significant
	Slice Crossover > Length adjustment Opr.	1.35E-12	Significant
	Combination > Length adjustment Opr.	1.53E-12	Significant
	Combination < Slice Crossover	0.89	Not Significant
SRP 2	Length adjustment Opr. > Basic ALGEP	0.02	Not Significant
	Slice Crossover > Basic ALGEP	4.39E-11	Significant
	Combination > Basic ALGEP	5.86E-11	Significant
	Slice Crossover > Length adjustment Opr.	3.26E-11	Significant
	Combination > Length adjustment Opr.	4.04E-11	Significant
	Combination > Slice Crossover	0.87	Not Significant
SRP 3	Length adjustment Opr. > Basic ALGEP	0.88	Not Significant
	Slice Crossover > Basic ALGEP	2.90E-07	Significant
	Combination > Basic ALGEP	7.49E-07	Significant
	Slice Crossover > Length adjustment Opr.	2.22E-06	Significant
	Combination > Length adjustment Opr.	7.36E-07	Significant
	Combination > Slice Crossover	0.18	Not Significant

5.2.3. Experiment 3

Experiment 3 is conducted to compare the proposed algorithm (ALGEP) with deep learning ANN. The result of Experiment 3 for monthly IHSG data is resumed in Table 5.7, while result for daily BBRI data is resumed in Table 5.8. All experiment show that the performance of ALGEP is better than ANN for validation data and for almost all training data. Beside producing better performance, ALGEP also gives an advantage of showing the equation, therefore, the user can understand how the prediction is built.

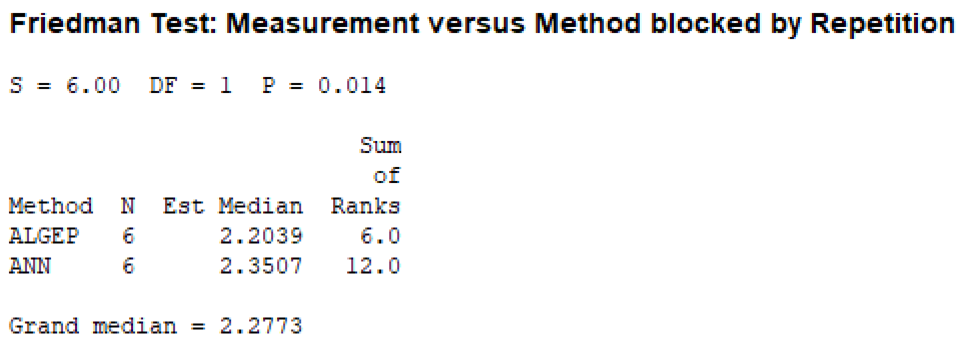
Table 5.7 Result of Experiment 3 for monthly IHSG data

Result	**Monthly IHSG**					
	Repetition 1		**Repetition 2**		**Repetition 3**	
	ALGEP	**ANN**	**ALGEP**	**ANN**	**ALGEP**	**ANN**
Best MAPE for training	**3.3289**	3.5607	3.1786	**3.1708**	**3.1403**	3.4230
Best MAPE for validation	**2.9943**	3.9146	**3.7413**	3.8437	**3.6365**	3.8720
Best equation for validation data	NextClose = 1.002 * Close - 0.284 * Close^2 * CloseChange * (0.621 * Close^2 * CloseChange - 0.191) + 0.621 * CloseChange^2 * Low * VolChange * (0.318 * CloseChange + (0.058 * Low) / High)	-	Nextclose = Close + 0.038 * Volume - (0.033 * Volume^2) / High	-	Next Close = Close + Close * Closechange * Volchange^2 – Open * Volume * Volchange * (Close * High * Open + 0.232) + 0.0003	

Table 5.8 Result of Experiment 3 for daily BBRI data

Result	**Daily BBRI**					
	Repetition 1		**Repetition 2**		**Repetition 3**	
	ALGEP	**ANN**	**ALGEP**	**ANN**	**ALGEP**	**ANN**
Best MAPE for training	**1.0219**	1.0636	**1.0112**	1.0536	**1.0169**	1.1815
Best MAPE for validation	**0.9845**	1.0084	**1.0200**	1.0987	**1.0045**	1.1957
Best equation for validation data	NextClose = Close + 0.000000067	-	NextClose = 1.0 * Close + Close * Open^2 * Volume^5 * VolChange - 0.047 * Low * Open^2 * Volume^3 * VolChange + (0.079 * Close * Open^2 * Volume^4) / (Volume + 0.293)	-	NextClose = Close	-

ALGEP produce smaller MAPE for validation than ANN in all repetitions and it can be concluded that ALGEP perform better than ANN in predicting IHSG and BBRI stock price using defined data input. A non-parametric test, Friedman Test, is applied to the experiment result. In the test, method (ALGEP and ANN) is set as treatment, while repetition is set as block. The test result, as shown in Figure 5.7, shows that validation MAPE of ALGEP significantly smaller than validation MAPE of ANN.



Friedman Test: Measurement versus Method blocked by Repetition

```
S = 6.00  DF = 1  P = 0.014

                           Sum
                            of
Method  N  Est Median  Ranks
ALGEP   6      2.2039    6.0
ANN     6      2.3507   12.0

Grand median = 2.2773
```

Figure 5.7 Friedman Test of IHSG and BBRI price forecasting

ALGEP also tends to give faster convergent. Figure 5.8 - Figure 5.10 shows clearly how ALGEP performance achieves convergent earlier than ANN for monthly IHSG forecasting. The same condition is also shown for daily BBRI forecasting as indicated in Figure 5.11 - Figure 5.13.

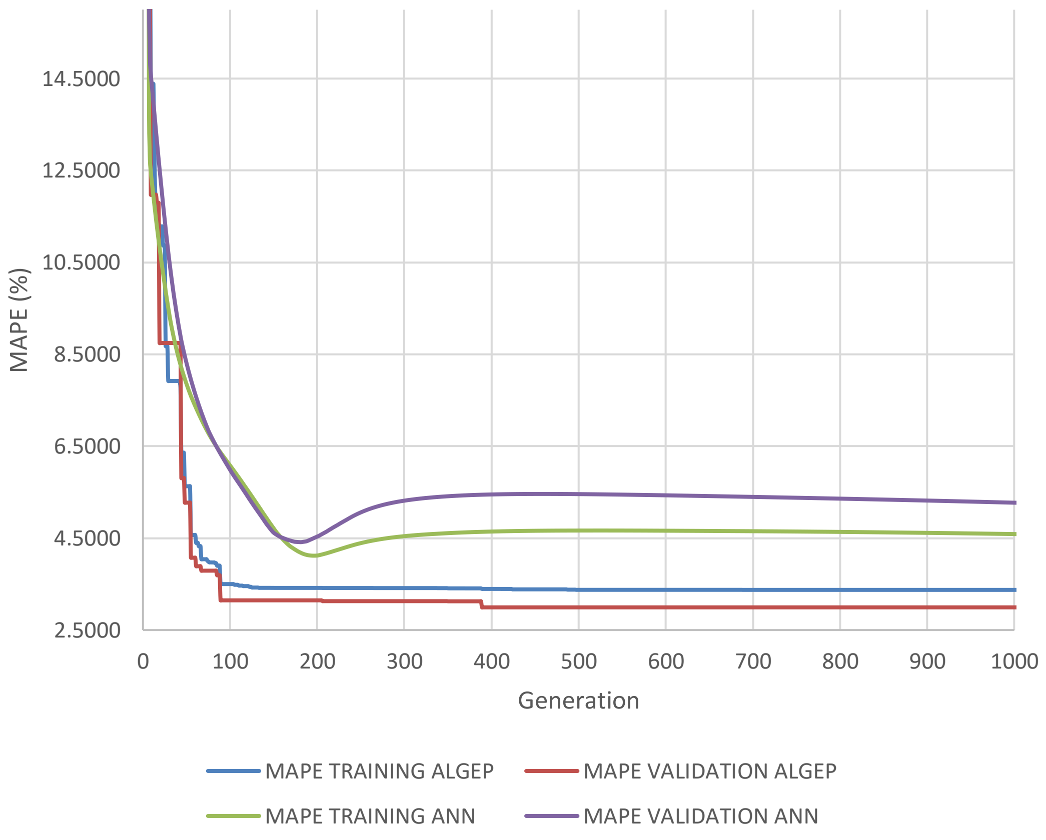


Figure 5.8 MAPE of monthly IHSG forecasting rep. 1

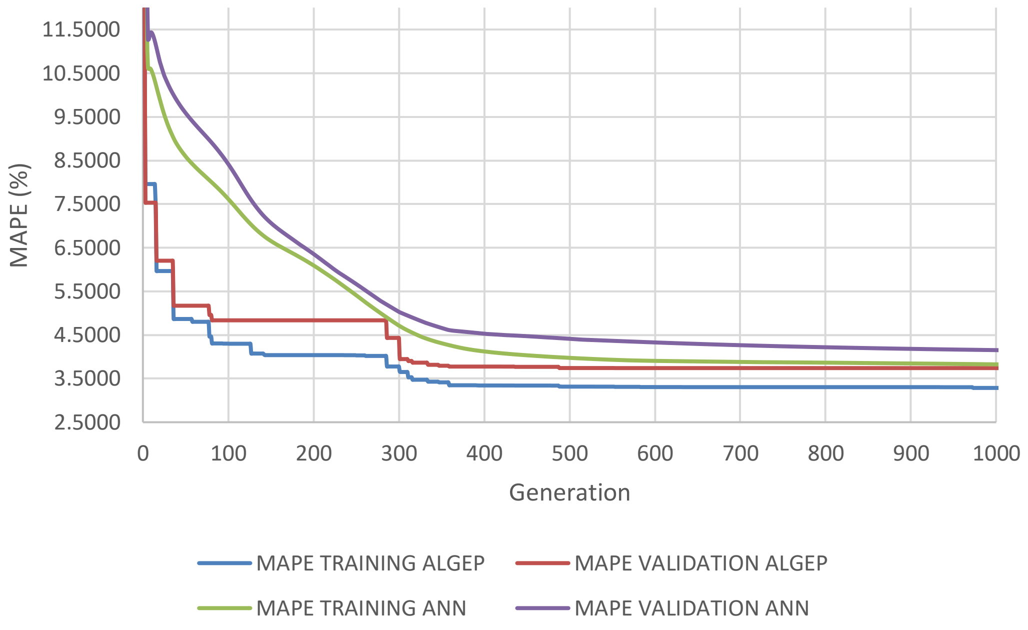


Figure 5.9 MAPE of monthly IHSG forecasting rep. 2

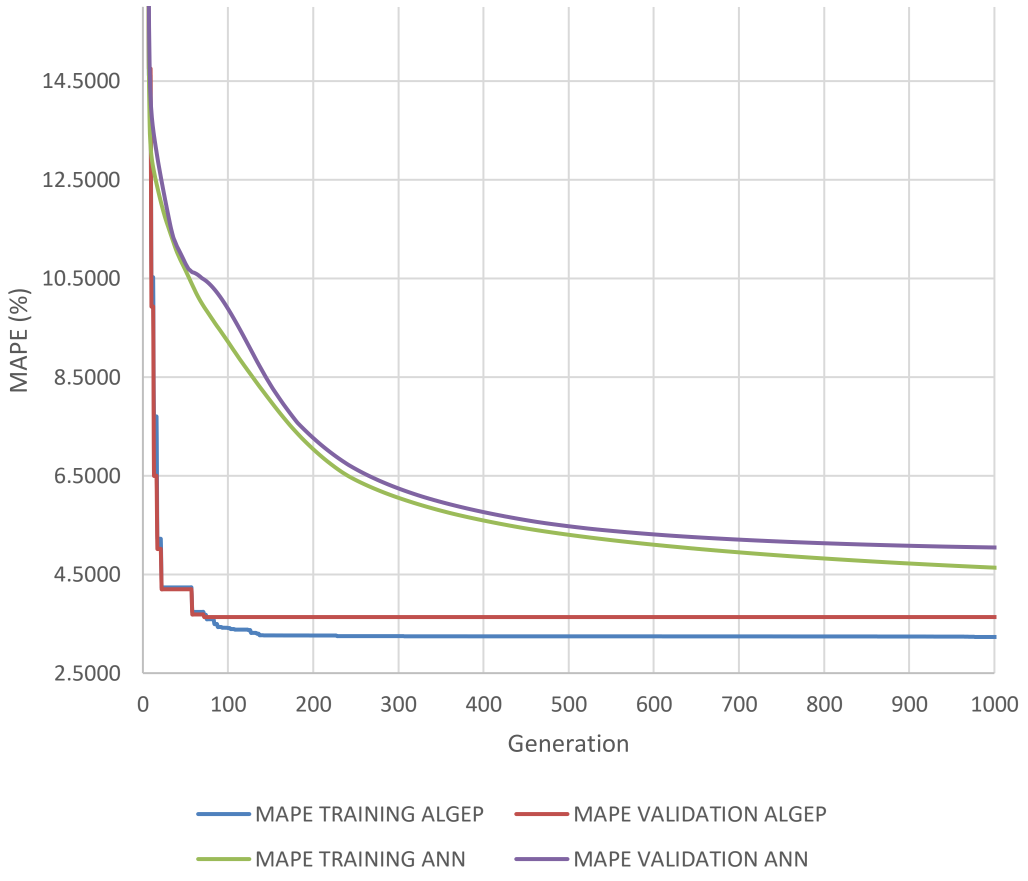


Figure 5.10 MAPE of monthly IHSG forecasting rep. 3

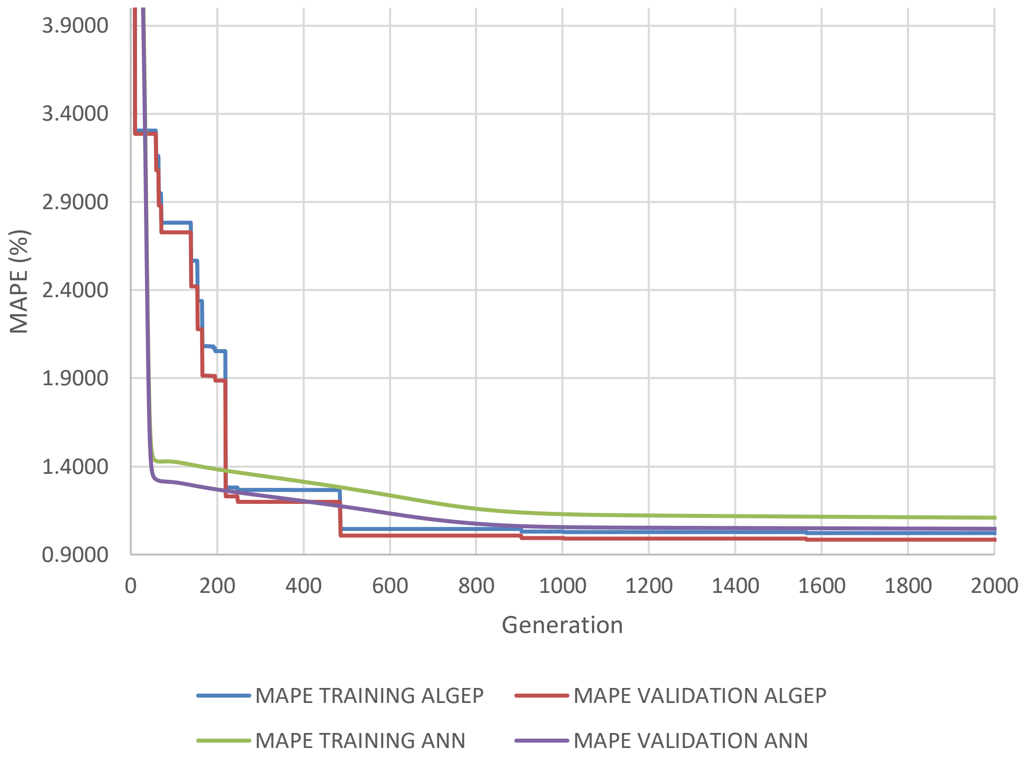


Figure 5.11 MAPE of daily BBRI forecasting rep. 1

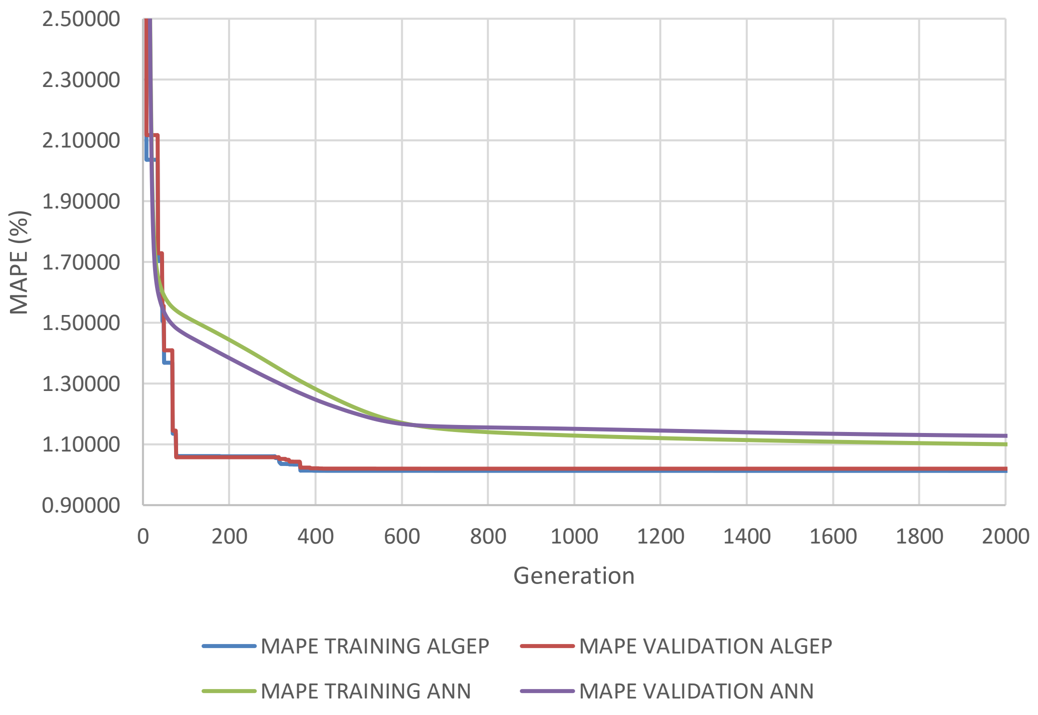


Figure 5.12 MAPE of daily BBRI forecasting rep. 2

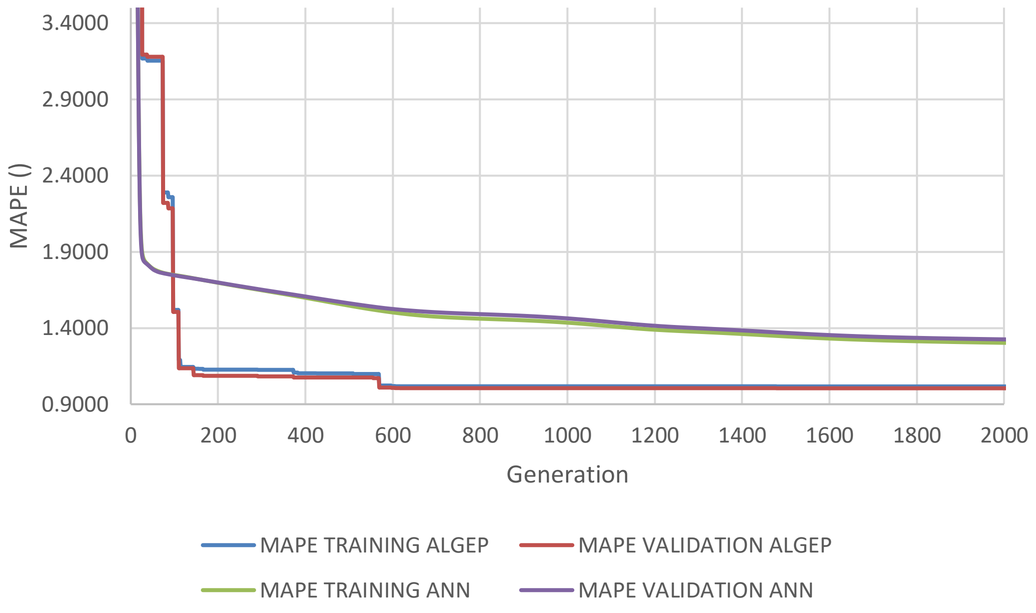


Figure 5.13 MAPE of daily BBRI forecasting rep. 3

Additionally, a k-folds cross-validation is done for predicting monthly IHSG to cestimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. In this case, k=5 is used. To perform 5-folds cross validation, the data is suflled, then divided into 5 group. After that 5 experiments is done where for the first experiment, the first group is used as validation data and the rest is for training data. For the second experiment, the second group is set as validation data and the rest as training dat, and so on. In this experiment, the maximum number of generation or epoch is 2000. Table 5.9 and Table 5.10 shows the performance of ALGEP and ANN for predicting monthly IHSG using 5-fold experiment. The table indicate that in term of MAPE, ALGEP shows better performance. However, the computation time of ALGEP for monthly IHSG is much longer then ANN but, the opposite happens in daily BBRI where the computation time of ALGEP is slightly faster than ANN. It seems that the computation time of ANN is more influenced by the number of data.

Table 5.9 k-fold cross validation result for monthly IHSG

Fold	ALGEP			ANN		
	Best MAPE Training	**Best MAPE validation**	**Computation time (s)**	**Best MAPE Training**	**Best MAPE validation**	**Computation time (s)**
1	3.36	2.98	1173.81	3.49	3.41	94.86
2	3.41	3.29	1130.80	3.45	3.18	94.34
3	3.50	2.87	697.59	3.47	3.28	89.34
4	3.14	3.74	918.69	3.39	4.36	93.42
5	3.18	3.91	1226.88	3.22	3.90	94.05
Average	**3.32**	**3.36**	**1029.55**	**3.40**	**3.62**	**93.20**

Table 5.10 k-fold cross validation result for daily BBRI

Fold	ALGEP			ANN		
	Best MAPE Training	**Best MAPE validation**	**Computation time (s)**	**Best MAPE Training**	**Best MAPE validation**	**Computation time (s)**
1	1.01	1.03	930.59	1.21	1.23	1502.81
2	1.00	1.05	1251.78	1.15	1.21	1284.58
3	1.02	1.00	1022.84	1.22	1.22	1236.22
4	1.02	1.01	1007.95	1.16	1.17	1263.52
5	1.02	0.98	842.77	1.11	1.04	1289.97
Average	**1.01**	**1.01**	**1011.19**	**1.17**	**1.17**	**1315.42**

5.3. Discussion

5.3.1. Algorithm Performance

The result of Experiment 1 shows that ALGEP performs better than basic GEP and RGEP to solve the SRPs. RGEP, as expected, also demonstrates a better performance than basic GEP. The superior performance of ALGEP indicates that the adaptive length in population work well to find the proper gene length, where the proper chromosome length will tend to survive during the evolutionary process. This result provides the basis to use ALGEP for the next researches and applications. With some developments and combination with other methods, ALGEP may become a promising algorithm in prediction field. However, more experiments in other problems need to be done before drawing general conclusion about the superior performance of ALGEP.

The use of proposed slice crossover operator in ALGEP increase the performance of the algorithm significantly as indicated from Experiment 2 result. The next researches may explore more about slice crossover to make it better. The application of n-point crossover in ALGEP will make some position of the genes in the longest chromosome cannot become the subject of crossover, while in slice crossover, all positions of the parents could become the subjects of crossover and increase the probability of producing good children. On the other hand, the application of length adjustment operator in ALGEP does not gives a significant effect. Length adjustment operator may produce more destructive effect on the gene than constructive effect and therefore does not make improvement of the population fitness.

In the Experiment 3, the performance of ALGEP is compared to the performance of deep learning ANN to predict monthly IHSG and daily BBRI stock price. MAPE is used as performance measurement. The result shows that ALGEP perform better than ANN. ALGEP also tends to achieve faster convergence compared to ANN. It is interested to see the comparison between the validation MAPE and the training MAPE (Table 5.7 and Table 5.8). Using ANN method, the training MAPE is lower than the validation MAPE for most of time. This condition is common in machine learning. However, using ALGEP method, the training MAPE does not indicate lower value than the validation MAPE. More experiments need to be done to get conclusion from this phenomena.

Using generation or epoch limit, the performance of ALGEP is better than ANN. How if the running time is restricted? Table 5.11 and Table 5.12 shows that ALGEP also produces better performance compare to ANN using time limit 300 seconds.

Table 5.11 ALGEP and ANN performance for predicting BBRI (300s time limit)

Fold	**ALGEP**		**ANN**	
	Best MAPE Training	**Best MAPE validation**	**Best MAPE Training**	**Best MAPE validation**
1	1.180	1.175	1.197	1.222
2	1.006	1.045	1.341	1.385
3	1.102	1.101	1.276	1.262
4	1.009	1.035	1.306	1.423
5	1.030	0.953	1.617	1.515
Average	**1.066**	**1.062**	**1.347**	**1.361**

Table 5.12 ALGEP and ANN performance for predicting IHSG (300s time limit)

Fold	**ALGEP**		**ANN**	
	Best MAPE Training	**Best MAPE validation**	**Best MAPE Training**	**Best MAPE validation**
1	3.612	3.401	3.547	3.506
2	3.350	3.320	3.565	3.574
3	3.269	3.518	3.672	4.097
4	3.413	3.314	3.685	3.449
5	3.338	3.540	3.393	3.704
Average	**3.396**	**3.419**	**3.572**	**3.666**

Another experiment is carried out to see the effect of function set to the performance of ALGEP for predicting daily BBRI price. The first treatment is using basic arithmetic operator which are *"+", "-", "*"* and *"/"* and the other is by adding *sin* and *cos* function. The experiment is run in one thousand generations. Table 5.13 shows that the adding *sin* and *cos* does not make the performance better. *Sin* and *cos* function might be work well in capturing cyclic data but in daily BBRI price case, the data seems to do not have cyclical pattern.

Table 5.13 The effect of function sets in ALGEP for predicting daily BBRI price

Fold	**Function set: { + - * / }**			**Function set: { + - * / *sin cos* }**		
	Best MAPE Training	**Best MAPE Validation**	**Computation time (s)**	**Best MAPE Training**	**Best MAPE validation**	**Computation time (s)**
1	1.0112	1.0237	509.2031	1.0112	1.0270	612.2031
2	1.0355	0.9292	537.1719	1.0355	0.9292	594.6563
3	1.0049	1.0523	543.2656	1.0049	1.0523	598.2031
4	1.0144	1.0145	551.6563	1.0144	1.0145	686.3125
5	1.0035	1.0488	1026.8281	1.0058	1.0489	498.6406
Average	**1.0139**	**1.0137**	**633.6250**	**1.0144**	**1.0144**	**598.0031**

5.3.2. Algorithm Efficiency

For each run, the algorithm stop when reaching maximum generation or getting best performance. Figure 5.14 shows the average number of generation for each algorithm. ALGEP has the smallest average number of generation among the other for almost all of chromosome length. It means that ALGEP can reach best performance faster than others.

In the experiment, each run has maximum 100 generations. The average time per run is depicted in Figure 5.15 while average computation time per generation is depicted in Figure 5.16. ALGEP has smallest average computation time per run because the average number of generation per run is also smallest. It is interested to see the average computation time per generation. In small chromosome length, it is higher than the others, and then becoming lower than other when chromosome length is higher. In short chromosome, there is no advantage of adaptive length since the optimal chromosome length is probably the same as max chromosome length. In this case, computation time of ALGEP is higher because ALGEP need to calculate the length of each individual chromosome involved in crossover or mutation. However, if the maximum chromosome length is higher, ALGEP will adjust the optimal chromosome length to its optimal length and therefore does not redundantly decode and calculate longest chromosome as in GEP or RGEP.

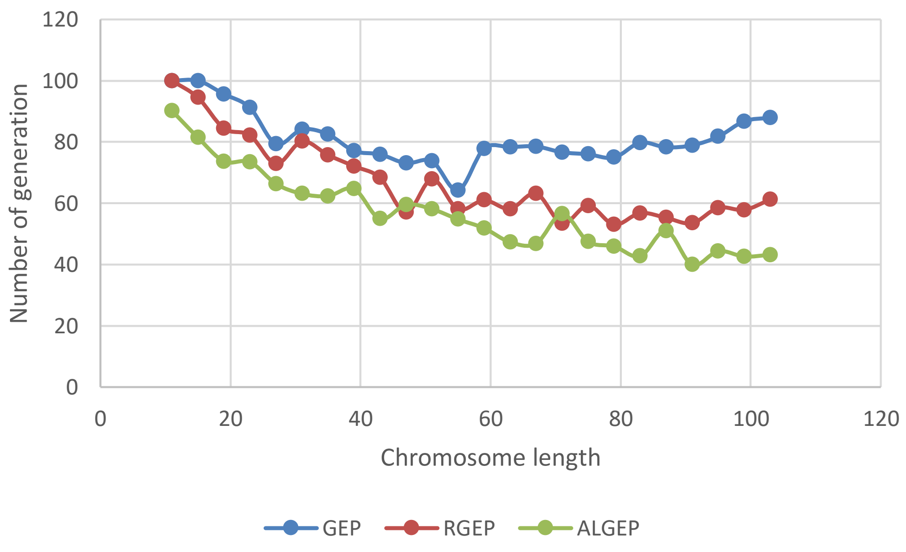


Figure 5.14 Average number of generation for SRP 1

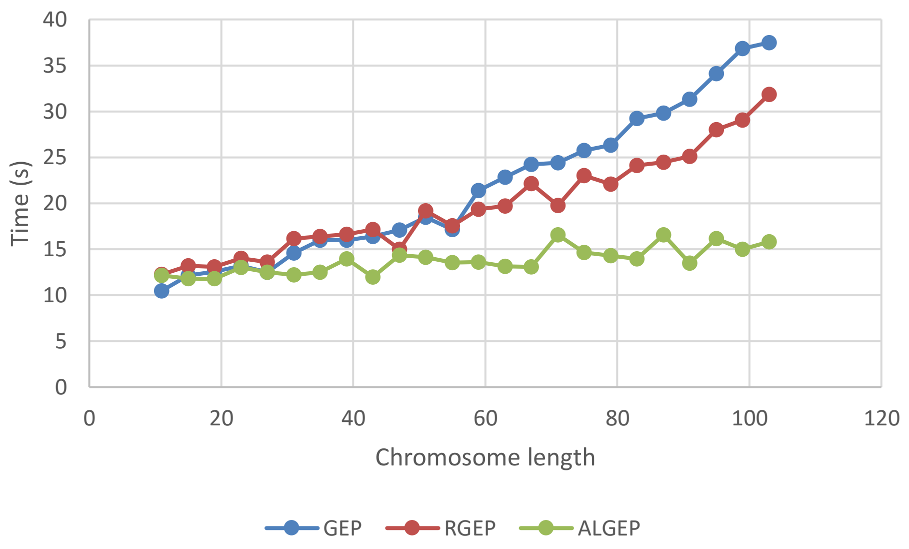


Figure 5.15 Average computation time per run

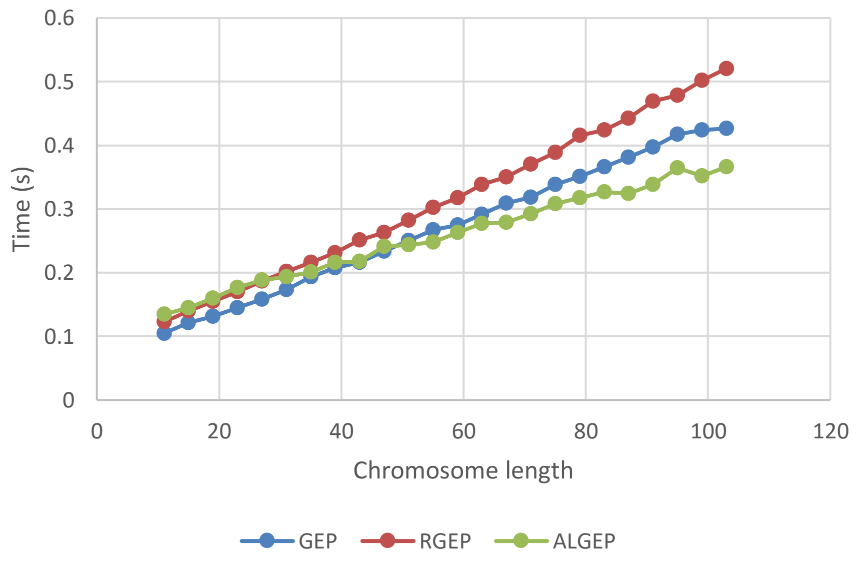


Figure 5.16 Average computation time per generation

Computation time of ALGEP is much longer than ANN especially for monthly IHSG as shown in k-fold experiment (Table 5.9) and experiment 3 (Table 5.14) but not much different for daily BBRI data as in Table 5.10 and Table 5.15. It seems that the computation time of ANN is more influenced by the number of data and ALGEP is more influenced by the number of generations. It is necessary to consider computation time in the application of ALGEP even though ALGEP is generally faster compared to GEP and RGEP in solving symbolic regression problem.

Table 5.14 Computation time for predicting monthly IHSG

Repetition	**Computation time (s)**	
	ANN	**ALGEP**
1	789.73	14486.72
2	789.27	7839.86
3	787.31	15057.31
Average	788.77	12461.3

Table 5.15 Computation time for predicting daily BBRI

Repetition	**Computation time (s)**	
	ANN	**ALGEP**
1	10327.31	17938.20
2	11103.47	9672.67
3	10495.48	9373.58
Average	10642.09	12328.15

It is depicted from Figure 5.8 to Figure 5.13 that ALGEP produces faster generation convergence than ANN. Table 5.16 shows that ALGEP has reached convergence before 600th generation in IHSG prediction where ANN has not reached convergence until maximum generation. Table 5.17 shows the similar result for BBRI prediction. Faster convergence has advantages and disadvantages. One of the advantages for quick convergence is faster in reach good solution, in the other hand, converging too quickly can prevent the discovery of better solutions. To adjust the convergence speed, user can set the selection pressure via parent selection and generation selection methods.

Table 5.16 Convergent generation for predicting monthly IHSG

Convergent (Generation)		
Repetition	**ALGEP**	**ANN**
1	389	-
2	553	-
3	72	-

Table 5.17 Convergent generation for predicting daily BBRI

Convergent (Generation)		
Repetition	**ALGEP**	**ANN**
1	17904	-
2	645	-
3	1479	-

6. CONCLUSION AND SUGGESTION

6.1. Conclusion

One of parameter setting issues in GEP is gene length. Different problem may need different gene length for best performance. This research tries to develop ALGEP, an GEP algorithm that can adaptively find the proper gene length. Instead of using fix gene length, the individuals in ALGEP have varying gene length. The individuals having proper gene length will tend to survive during evolutionary process.

This research proposes a length adjustment operator that can adjust the length of chromosome during evolutionary process. This operator can lengthen or shorten the gene for one allele. This research also develops a crossover operator that is suitable for the proposed encoding where instead of using the same crossover point for both parent, it may use different point. Using this operator will make all section of gene subject to crossover. A simple constant creation method and a constant mutation operator are also proposed.

The performance of ALGEP is compared to GEP and RGEP for solving 3 SRPs and the result shows that ALGEP outperforms GEP and RGEP significantly. The effectivity of length adjustment operator and slice crossover are also examined to solve 3 SRPs. As a conclusion, the use of the proposed slice crossover on ALGEP can increase the performance of the algorithm significantly. This is caused by the ability of slice crossover's ability to reach all parent chromosome areas. Unfortunately, application of the proposed length adjustment operator on ALGEP does not contribute in increasing performance. Length adjustment operator is suspected producing more destructive effect on the gene than constructive effect and, as a result, does not make improvement of the population fitness.

The performance of ALGEP is compared with deep learning ANN to predict monthly IHSG and daily BBRI stock prices. MAPE is applied as performance indicator. The result shows that ALGEP outperform deep learning ANN in all validation data forecasting. It can be concluded that ALGEP can perform better than ANN for stock price forecasting. However, the computation time of ALGEP that is longer than ANN especially for small number of data and it needs to be considered in the application of ALGEP.

6.2. Suggestion

Applying slice crossover on ALGEP produce superior performance among GEP, RGEP and basic ALGEP, therefore, this algorithm is promising for the next researches and application. However, the performance of the algorithm is still not good enough when applied to solve difficult SRPs. Some developments and tunings is required to make the algorithm better. Furthermore, this research only explores ALGEP using one gene for solving SRP. It would be interested to develop adaptive number of genes in the chromosome and compare the result with AdaGEP, ADF, SL-GEP of other multi-gene algorithms.

The capability of ALGEP in prediction is comparable to other machine learning algorithm and therefore, ALGEP can be good choice to be developed in the machine learning research field, especially for the problem where user needs to know the model or equation which underlines the decision. SRP is one filed where ALGEP is suitable for use. Other areas where GEP is frequently used such as combinatorial optimization and automated model design problem are also suitable areas for the use of ALGEP.

In the experiment to predict IHSG and BBRI price, the use of sin and cos function does not provide a significant effect. However, for different data types, different sets of functions may be required. For example, the sin and cos functions is probably suitable for capturing seasonal or cyclical data patterns. Logical functions can also be used to capture non-continuous functions.

REFERENCES

Agrawal, J.G., Chourasia, D.V.S., Mittra, D.A.K., 2013. State-of-the-Art in Stock Prediction Techniques. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering 2, 7.

Alghieth, M., 2016. Gene expression programming for Efficient Time-series Financial Forecasting. De Montfort University, Leichester.

Alghieth, M., Yang, Y., Chiclana, F., 2015. Development of 2D curve-fitting genetic/gene-expression programming technique for efficient time-series financial forecasting, in: 2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA). Presented at the 2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA), IEEE, Madrid, Spain, pp. 1–8. https://doi.org/10.1109/INISTA.2015.7276734

Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Hasan, M., Van Essen, B.C., Awwal, A.A.S., Asari, V.K., 2019. A State-of-the-Art Survey on Deep Learning Theory and Architectures. Electronics 8, 292. https://doi.org/10.3390/electronics8030292

Armstrong, J.S. (Ed.), 2001. Principles of forecasting: a handbook for researchers and practitioners, International series in operations research & management science. Kluwer Academic, Boston, MA.

Armstrong, J.S., Collopy, F., 1992. Error measures for generalizing about forecasting methods: Empirical comparisons. International Journal of Forecasting 08, 69–80.

Azamathulla, H.Md., Ahmad, Z., 2013. Estimation of Critical Velocity for Slurry Transport through Pipeline Using Adaptive Neuro-Fuzzy Interference System and Gene-Expression Programming. J. Pipeline Syst. Eng. Pract. 4, 131–137. https://doi.org/10.1061/(ASCE)PS.1949-1204.0000123

Barbulescu, A., Bautu, E., 2012. A Hybrid Approach for Modeling Financial Time Series 9, 9.

Bărbulescu, A., Dumitriu, C. Ștefan, 2021. On the Connection between the GEP Performances and the Time Series Properties. Mathematics 9, 1853. https://doi.org/10.3390/math9161853

Bathaee, Y., 2018. The Artificial Intelligence Black Box and the Failure of Intent and Causation. Harvard Journal of Law & Technology 31, 50.

Bautu, E., Bautu, A., Luchian, H., 2010. Evolving Gene Expression Programming Classifiers for Ensemble Prediction of Movements on the Stock Market, in: 2010 International Conference on Complex, Intelligent and Software Intensive Systems. Presented at the 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), IEEE, Krakow, TBD, Poland, pp. 108–115. https://doi.org/10.1109/CISIS.2010.101

Bautu, E., Bautu, A., Luchian, H., 2007. AdaGEP - An Adaptive Gene Expression Programming Algorithm, in: Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2007). Presented at the 2007 Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE, Timisoara, Romania, pp. 403–406. https://doi.org/10.1109/SYNASC.2007.51

Baykasoğlu, A., 2008. Gene expression programming based meta-modelling approach to production line design. International Journal of Computer Integrated Manufacturing 21, 657–665. https://doi.org/10.1080/09511920701370753

Baykasoğlu, A., Göçken, M., 2009. Gene expression programming based due date assignment in a simulated job shop. Expert Systems with Applications 36, 12143–12150. https://doi.org/10.1016/j.eswa.2009.03.061

Billah, B., King, M.L., Snyder, R.D., Koehler, A.B., 2006. Exponential smoothing model selection for forecasting. International Journal of Forecasting 22, 239–247. https://doi.org/10.1016/j.ijforecast.2005.08.002

Chen, H.-H., Yang, C.-B., Peng, Y.-H., 2014. The trading on the mutual funds by gene expression programming with Sortino ratio. Applied Soft Computing 15, 219–230. https://doi.org/10.1016/j.asoc.2013.09.011

Chi Zhou, Weimin Xiao, Tirpak, T.M., Nelson, P.C., 2003. Evolving accurate and compact classification rules with gene expression programming. IEEE Trans. Evol. Computat. 7, 519–531. https://doi.org/10.1109/TEVC.2003.819261

Chollet, F., 2018. Deep learning with Python. Manning Publications Co, Shelter Island, New York.

Davydenko, A., Fildes, R., 2014. Measuring Forecasting Accuracy: Problems and Recommendations (by the Example of SKU-Level Judgmental Adjustments), in: Choi, T.-M., Hui, C.-L., Yu, Y. (Eds.), Intelligent Fashion Forecasting Systems: Models and Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 43–70. https://doi.org/10.1007/978-3-642-39869-8_4

Deng, S., Yuan, C., Yang, L., Zhang, L., 2018. Distributed electricity load forecasting model mining based on hybrid gene expression programming and cloud computing. Pattern Recognition Letters 109, 72–80. https://doi.org/10.1016/j.patrec.2017.10.004

Diebold, F.X., Mariano, R.S., 2002. Comparing predictive accuracy. Journal of Business & Economic Statistics 20, 134–144.

Dikmen, E., 2015. Gene expression programming strategy for estimation performance of LiBr–H2O absorption cooling system. Neural Comput & Applic 26, 409–415. https://doi.org/10.1007/s00521-014-1723-9

Fajfar, I., Tuma, T., 2018. Creation of Numerical Constants in Robust Gene Expression Programming. Entropy 20, 756. https://doi.org/10.3390/e20100756

Ferreira, Candida, 2006. Gene expression programming: mathematical modeling by an artificial intelligence, 2nd rev. and extended ed. ed, Studies in computational intelligence. Springer-Verlag, Berlin ; New York.

Ferreira, Cândida, 2006. Designing Neural Networks Using Gene Expression Programming, in: Abraham, A., de Baets, B., Köppen, M., Nickolay, B. (Eds.), Applied Soft Computing Technologies: The Challenge of Complexity. Springer-Verlag, Berlin/Heidelberg, pp. 517–535. https://doi.org/10.1007/3-540-31662-0_40

Fildes, R., Goodwin, P., 2007. Against Your Better Judgment? How Organizations Can Improve Their Use of Management Judgment in Forecasting. Interfaces 37, 570–576. https://doi.org/10.1287/inte.1070.0309

Foss, T., Stensrud, E., Kitchenham, B., Myrtveit, I., 2003. A simulation study of the model evaluation criterion mmre. IIEEE Trans. Software Eng. 29, 985–995. https://doi.org/10.1109/TSE.2003.1245300

Franses, P.H., 2016. A note on the Mean Absolute Scaled Error. International Journal of Forecasting 32, 20–22. https://doi.org/10.1016/j.ijforecast.2015.03.008

Gao, X.W., Guan, B.B., Guan, X.J., 2013. Study on the Optimize Strategies of Gene Expression Programming. AMM 432, 565–570. https://doi.org/10.4028/www.scientific.net/AMM.432.565

Ghahtarani, A., Sheikhmohammady, M., Najafi, A.A., 2019. Mathematical modeling for a new portfolio selection problem in bubble condition, using a new risk measure. Scientia Iranica 0, 0–0. https://doi.org/10.24200/sci.2019.51577.2258

Gharagheizi, F., Ilani-Kashkouli, P., Farahani, N., Mohammadi, A.H., 2012. Gene expression programming strategy for estimation of flash point temperature of non-electrolyte organic compounds. Fluid Phase Equilibria 329, 71–77. https://doi.org/10.1016/j.fluid.2012.05.015

Gholami, A., Bonakdari, H., Zaji, A.H., Akhtari, A.A., Khodashenas, S.R., 2015. Predicting the velocity field in a 90° Open channel bend using a gene expression programming model. Flow Measurement and Instrumentation 46, 189–192. https://doi.org/10.1016/j.flowmeasinst.2015.10.006

Goodwin, P., Lawton, R., 1999. On the asymmetry of the symmetric MAPE. International Journal of Forecasting 15, 405–408. https://doi.org/10.1016/S0169-2070(99)00007-2

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D., 2018. A Survey of Methods for Explaining Black Box Models. ACM Comput. Surv. 51, 1–42. https://doi.org/10.1145/3236009

Hashmi, M.Z., Shamseldin, A.Y., 2014. Use of Gene Expression Programming in regionalization of flow duration curve. Advances in Water Resources 68, 1–12. https://doi.org/10.1016/j.advwatres.2014.02.009

Huang, C.-H., Yang, C.-B., Chen, H.-H., 2013. Trading Strategy Mining with Gene Expression Programming, in: Proceedings of the 2013 International Conference on Applied Mathematics and Computational Methods in Engineering 80. p. 6.

Hyndman, R.J., Koehler, A.B., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001

Imani, M., You, R.-J., Kuo, C.-Y., 2014. Forecasting Caspian Sea level changes using satellite altimetry data (June 1992–December 2013) based on evolutionary support vector regression algorithms and gene expression programming. Global and Planetary Change 121, 53–63. https://doi.org/10.1016/j.gloplacha.2014.07.002

J. L. Ávila, E. L. Gibaja, A. Zafra, S. Ventura, 2011. A Gene Expression Programming Algorithm for Multi-Label Classification.pdf. J. of Mult.-Valued Logic & Soft Computing 17, 183–206.

Janeiro, F., Ramos, P., 2012. Sensor characterization using gene expression programming evolutionary algorithms, in: 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings.

Presented at the 2012 IEEE International Instrumentation and Measurement Technology Conference (I2MTC 2012), IEEE, Graz, pp. 1–5. https://doi.org/10.1109/I2MTC.2012.6851794

Janeiro, F.M., Santos, J., Ramos, P.M., 2013. Gene Expression Programming in Sensor Characterization: Numerical Results and Experimental Validation. IEEE Trans. Instrum. Meas. 62, 1373–1381. https://doi.org/10.1109/TIM.2012.2224275

Karatahansopoulos, A., Sermpinis, G., Laws, J., Dunis, C., 2014. Modelling and Trading the Greek Stock Market with Gene Expression and Genetic Programing Algorithms: Gene Expression and Genetic Programing Algorithms. J. Forecast. 33, 596–610. https://doi.org/10.1002/for.2290

Karathanasopoulos, A., 2017. Modelling and trading the London, New York and Frankfurt stock exchanges with a new gene expression programming trader tool: Gene expression Programming Trader Tool. Intell. Sys. Acc. Fin. Mgmt. 24, 3–11. https://doi.org/10.1002/isaf.1401

Keijzer, M., 2003. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling, in: Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (Eds.), Genetic Programming, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 70–82. https://doi.org/10.1007/3-540-36599-0_7

Keshavarz, A., Mehramiri, M., 2015. New Gene Expression Programming models for normalized shear modulus and damping ratio of sands. Engineering Applications of Artificial Intelligence 45, 464–472. https://doi.org/10.1016/j.engappai.2015.07.022

Kolassa, S., Martin, R., 2011. Percentage Errors Can Ruin Your Day (and Rolling the Dice Shows How). Foresight Fall, 8.

Korns, M.F., 2011. Accuracy in Symbolic Regression, in: Riolo, R., Vladislavleva, E., Moore, J.H. (Eds.), Genetic Programming Theory and Practice IX, Genetic and Evolutionary Computation. Springer New York, New York, NY, pp. 129–151. https://doi.org/10.1007/978-1-4614-1770-5_8

Koza, J.R., 1992. Genetic programming: on the programming of computers by means of natural selection, Complex adaptive systems. MIT Press, Cambridge, Mass.

Lee, C.-H., Yang, C.-B., Chen, H.-H., 2014. Taiwan Stock Investment with Gene Expression Programming. Procedia Computer Science 35, 137–146. https://doi.org/10.1016/j.procs.2014.08.093

Li, X., Zhou, C., Xiao, W., Nelson, P.C., 2005. Prefix Gene Expression Programming. Presented at the Genetic and Evolutionary Computation Conference (GECCO)’05, p. 7.

Makridakis, S., 1993. Accuracy measures: theoretical and practical concerns. International Journal of Forecasting 9, 527–529. https://doi.org/10.1016/0169-2070(93)90079-3

Md. Azamathulla, H., 2013. Gene-expression programming to predict friction factor for Southern Italian rivers. Neural Comput & Applic 23, 1421–1426. https://doi.org/10.1007/s00521-012-1091-2

Mehr, A.D., 2018. An improved gene expression programming model for streamflow forecasting in intermittent streams. Journal of Hydrology 563, 669–678. https://doi.org/10.1016/j.jhydrol.2018.06.049

Mohammed, M., Khan, M.B., Bashier, E.B.M., 2016. Machine Learning: Algorithms and Applications, 0 ed. CRC Press, Boca Raton : CRC Press, 2017. https://doi.org/10.1201/9781315371658

Mousavi, S.M., Mostafavi, E.S., Hosseinpour, F., 2014. Gene expression programming as a basis for new generation of electricity demand prediction models. Computers & Industrial Engineering 74, 120–128. https://doi.org/10.1016/j.cie.2014.05.010

Mwaura, J., Keedwell, E., 2015. Evolving robot sub-behaviour modules using Gene Expression Programming. Genet Program Evolvable Mach 16, 95–131. https://doi.org/10.1007/s10710-014-9229-x

Mwaura, J., Keedwell, E., 2009. Adaptive Gene Expression Programming Using a Simple Feedback Heuristic. Presented at the AISB Conference, p. 6.

Nazari, A., 2012. Prediction performance of PEM fuel cells by gene expression programming. International Journal of Hydrogen Energy 37, 18972–18980. https://doi.org/10.1016/j.ijhydene.2012.08.101

Nedjah, N., Abraham, A., Macedo Mourelle, L. de (Eds.), 2006. Genetic systems programming: theory and experiences, Studies in computational intelligence. Springer, Berlin.

Nie, L., Gao, L., Li, P., Li, X., 2013. A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. J Intell Manuf 24, 763–774. https://doi.org/10.1007/s10845-012-0626-9

Nie, L., Shao, X., Gao, L., Li, W., 2010. Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. Int J Adv Manuf Technol 50, 729–747. https://doi.org/10.1007/s00170-010-2518-5

Peng, Y., Yuan, C., Qin, X., Huang, J., Shi, Y., 2014. An improved Gene Expression Programming approach for symbolic regression problems. Neurocomputing 137, 293–301. https://doi.org/10.1016/j.neucom.2013.05.062

Rashidi, S., Ranjitkar, P., 2015. Bus Dwell Time Modeling Using Gene Expression Programming: Bus Dwell Time Modelling Using GEP. Computer-Aided Civil and Infrastructure Engineering 30, 478–489. https://doi.org/10.1111/mice.12125

Ryan, N., Hibler, D., 2011. Robust Gene Expression Programming. Procedia Computer Science 6, 165–170. https://doi.org/10.1016/j.procs.2011.08.032

Sabar, N.R., Ayob, M., Kendall, G., Qu, R., 2015. Automatic Design of a Hyper-Heuristic Framework With Gene Expression Programming for Combinatorial Optimization Problems. IEEE Trans. Evol. Computat. 19, 309–325. https://doi.org/10.1109/TEVC.2014.2319051

Sadat Hosseini, S.S., Gandomi, A.H., 2012. Short-term load forecasting of power systems by gene expression programming. Neural Comput & Applic 21, 377–389. https://doi.org/10.1007/s00521-010-0444-y

Sermpinis, G., Fountouli, A., Theofilatos, K., Karathanasopoulos, A., 2013. Gene Expression Programming and Trading Strategies, in: Papadopoulos, H., Andreou, A.S., Iliadis, L., Maglogiannis, I. (Eds.), Artificial Intelligence Applications and Innovations. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 497–505. https://doi.org/10.1007/978-3-642-41142-7_50

Sivanandam, S.N., Deepa, S.N., 2007. Introduction to genetic algorithms. Springer, Berlin ; New York.

Teodorescu, L., 2006. Gene Expression Programming Approach to Event Selection in High Energy Physics. IEEE Trans. Nucl. Sci. 53, 2221–2227. https://doi.org/10.1109/TNS.2006.878571

Traore, S., Guven, A., 2013. New algebraic formulations of evapotranspiration extracted from gene-expression programming in the tropical seasonally dry regions of West Africa. Irrig Sci 31, 1–10. https://doi.org/10.1007/s00271-011-0288-y

Visoiu, A., 2011. Deriving Trading Rules Using Gene Expression Programming. Informatica Economica 15, 9.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Computat. 1, 67–82. https://doi.org/10.1109/4235.585893

Wu, Z., Fan, H., Liu, G., 2015. Forecasting Construction and Demolition Waste Using Gene Expression Programming. J. Comput. Civ. Eng. 29, 04014059. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000362

Xu, K., Liu, Y., Tang, R., Zuo, J., Zhu, J., Tang, C., 2009. A novel method for real parameter optimization based on Gene Expression Programming. Applied Soft Computing 9, 725–737. https://doi.org/10.1016/j.asoc.2008.09.007

Xue-song Yan, Wei Wei, Rui Liu, San-you Zeng, Li-shan Kang, 2006. Designing Electronic Circuits by Means of Gene Expression Programming, in: First NASA/ESA Conference on Adaptive Hardware and Systems (AHS’06). Presented at the First NASA/ESA Conference on Adaptive Hardware and Systems (AHS’06), IEEE, Istanbul, Turkey, pp. 194–199. https://doi.org/10.1109/AHS.2006.31

Yang, B., Zhang, W., Wang, H., 2019. Stock Market Forecasting Using Restricted Gene Expression Programming. Computational Intelligence and Neuroscience 2019, 1–14. https://doi.org/10.1155/2019/7198962

Yang, J., Ma, J., 2016. A hybrid gene expression programming algorithm based on orthogonal design. International Journal of Computational Intelligence Systems 9, 778–787. https://doi.org/10.1080/18756891.2016.1204124

Yang, Z., Wen, Y., Chen, Y., 2018. sEMG-Based Drawing Trace Reconstruction: A Novel Hybrid Algorithm Fusing Gene Expression Programming into Kalman Filter. Sensors 18, 3296. https://doi.org/10.3390/s18103296

Yassin, M.A., Alazba, A.A., Mattar, M.A., 2016. Artificial neural networks versus gene expression programming for estimating reference evapotranspiration in arid climate. Agricultural Water Management 163, 110–124. https://doi.org/10.1016/j.agwat.2015.09.009

Yavas, B.F., Dedi, L., 2016. An investigation of return and volatility linkages among equity markets: A study of selected European and emerging countries. Research in International Business and Finance 37, 583–596. https://doi.org/10.1016/j.ribaf.2016.01.025

Zhang, K., Sun, S., 2013. Web music emotion recognition based on higher effective gene expression programming. Neurocomputing 105, 100–106. https://doi.org/10.1016/j.neucom.2012.06.041

Zhang, Q., Zhou, C., Xiao, W., Nelson, P.C., 2007. Improving gene expression programming performance by using differential evolution, in: Sixth

International Conference on Machine Learning and Applications (ICMLA 2007). Presented at the Sixth International Conference on Machine Learning and Applications (ICMLA 2007), IEEE, Cincinnati, OH, USA, pp. 31–37. https://doi.org/10.1109/ICMLA.2007.62

Zhang, Y., Xiao, J., 2010. A New Strategy for Gene Expression Programming and Its Applications in Function Mining. Universal Journal of Computer Science and Engineering Technology 1, 122.

Zhong, J., Feng, L., Ong, Y.-S., 2017. Gene Expression Programming: A Survey [Review Article]. IEEE Comput. Intell. Mag. 12, 54–72. https://doi.org/10.1109/MCI.2017.2708618

Zhong, J., Ong, Y.-S., Cai, W., 2016. Self-Learning Gene Expression Programming. IEEE Trans. Evol. Computat. 20, 65–80. https://doi.org/10.1109/TEVC.2015.2424410

Zuo, J., Tang, C., Li, C., Yuan, C., Chen, A., 2004. Time Series Prediction Based on Gene Expression Programming, in: Li, Q., Wang, G., Feng, L. (Eds.), Advances in Web-Age Information Management. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 55–64. https://doi.org/10.1007/978-3-540-27772-9_7

ATTACHMENT

Attachment 1 Python Code

Experiment.py:

```
"""
Created on Thu Oct 17 10:53:57 2019

@author: AMAR
"""
from SAR_GEP5 import main
from time import perf_counter, localtime
import openpyxl
from winsound import Beep

# Data
data = {}

#For stock time series
####################################################
# data['file'] = 'D:\Disertasi\AI for Stock Price Prediction\IHSG MONTHLY.xlsx'
# data['sheet'] = 'IHSG MONTHLY'
# data['cycle'] = 6
# data['n_period'] = 60
# data['normalize'] = False
####################################################

#For SRP
####################################################
data['file'] = 'BBRI DATA1.xlsx'
data['sheet'] = 'Sheet1'
###################################################

data['n_input'] = 32
data['n_output'] = 1

# Functions and terminals
function = ['Sub','Add','Mul','Div']

# GEP Parameters
param = {}
param['max_gen'] = 10
param['pop_size'] = 100
param['encoding'] = 'sargep'
param['p'] = 0.0001
param['p con'] = 0.1
param['R'] = 1
param['recom_method'] = 'slice'
param['best_fitness'] = 1
param['with_const'] = False
param['perf'] = 'NSE'
param['rand_weight'] = 'uniform'

# Experiment variable
run = 1
l_head_min = 50
l_head_max = 50
l_head_step = 1


wb = openpyxl.Workbook()
sheet=wb.get_sheet_by_name('Sheet')
wb.create_sheet(index=1, title='RESUME')
resume=wb.get_sheet_by_name('RESUME')
```

```
wb.create_sheet(index=1, title='PARAMETERS')
parameters=wb.get_sheet_by_name('PARAMETERS')

# Recording parameters in Excel
j=1
parameters.cell(row=j,column = 1).value = 'NON DEFAULT PARAMETERS:'
for i in param:
    j += 1
    parameters.cell(row=j,column = 1).value = i
    parameters.cell(row=j,column = 2).value = param[i]


resume['A1'] = 'GENE LENGTH'
resume['A2'] = 'SUCCESS RATE (%)'
resume['A3'] = 'AVERAGE BEST FITNESS'
resume['A4'] = 'BEST FITNESS'
resume['A5'] = 'AVERAGE BEST ' + param['perf']
resume['A6'] = 'BEST ' + param['perf']
resume['A7'] = 'AVERAGE GENERATION'
resume['A8'] = 'AVERAGE COMP TIME/ RUN'
resume['A9'] = 'AVERAGE COMP TIME/ GEN'
resume['A10'] = 'FITNESS EVALUATION UNTIL MEET TARGET OR MAX GEN'


l_head = l_head_min

col = 1 # for writing column in excel result

while l_head <= l_head_max:

    param['l_head'] = l_head

    #initial Value
    success = 0
    sum_time = 0
    sum_b_fit = 0
    sum_b_perf = 0
    b_fit_max = 0
    sum_fit_count = 0
    sum_gen = 0


    for i in range(run):
        start = perf_counter()

        # Calling main SAR-GEP algorithm
        (b_fit, b_perf, b_eq, fit_count, gen, l_chrom) = main(data, function, param, i)

        gen += 1
        sum_fit_count += fit_count
        sum_b_fit += b_fit
        sum_b_perf += b_perf
        sum_gen += gen

        if b_fit == param['best_fitness']:
            success += 1

        if b_fit > b_fit_max:
            b_fit_max = b_fit
            b_perf_max = b_perf
            b_eq_max = b_eq

        # Calculating computing time for each run
        end = perf_counter()
        comp_time = end - start
        sum_time = sum_time + comp_time

        # Recording the result
        sheet.cell(row=i+3,column=(col-1)*5+1).value = i+1
        sheet.cell(row=i+3,column=(col-1)*5+2).value = b_fit
        sheet.cell(row=i+3,column=(col-1)*5+3).value = b_perf
```

```
            sheet.cell(row=i+3,column=(col-1)*5+4).value = gen
            sheet.cell(row=i+3,column=(col-1)*5+5).value = comp_time

        sheet.cell(row=1, column=(col-1)*5+1).value = 'GENE LENGTH = ' + str(l_chrom)
        sheet.cell(row=2, column=(col-1)*5+1).value = 'RUN'
        sheet.cell(row=2, column=(col-1)*5+2).value = 'BEST FITNESS'
        sheet.cell(row=2, column=(col-1)*5+3).value = 'BEST ' + param['perf']
        sheet.cell(row=2, column=(col-1)*5+4).value = 'GENERATION'
        sheet.cell(row=2, column=(col-1)*5+5).value = 'COMPUTING TIME'

        col +=1

        resume.cell(row=1, column=col).value = l_chrom
        resume.cell(row=2, column=col).value = success * 100 / run
        resume.cell(row=3, column=col).value = sum_b_fit / run
        resume.cell(row=4, column=col).value = b_fit_max
        resume.cell(row=5, column=col).value = sum_b_perf / run
        resume.cell(row=6, column=col).value = b_perf_max
        resume.cell(row=7, column=col).value = sum_gen / run
        resume.cell(row=8, column=col).value = sum_time / run
        resume.cell(row=9, column=col).value = sum_time / sum_gen
        resume.cell(row=10, column=col).value = sum_fit_count / run

        l_head += l_head_step



    # Creating file name to save
    time_now = [str(i) for i in list(localtime())]
    time_now = '-'.join(time_now[:3])+', '+'.'.join(time_now[3:6])
    const_cond = ' with constant' if param['with_const'] == True else ' without constant'
    file_name = 'Result ' + data['sheet'] + ' ' + param['encoding'] + \
        const_cond + '  at ' + time_now

    # Saving the Result
    wb.save(file_name + '.xlsx')
    print(file_name)

    # print(file_name)
    Beep(1000, 500)
```

SAR_GEP5.py:

```
# -*- coding: utf-8 -*-

"""

Created on Mon Oct  7 12:10:38 2019


@author: AMAR


This is libraries for Self-Adaptive Robust Gene Expression Programming (SAR-GEP)

"""

# ==============================================================================



# Import necesary modules

from random import random, choices, choice, uniform, randrange

from numpy.random import choice as numpyChoice

from statistics import stdev

from openpyxl import load_workbook

from inspect import getargspec

#from math import sin

from sympy import simplify




# ==============================================================================

# List of function and convertion to list type



# Math functions---------------------------------------------------------------

def Add(a,b):

    return ['(', str(''.join(a)), ' + ', str(''.join(b)), ')']



def Sub(a,b):

    return ['(', str(''.join(a)), ' - ', str(''.join(b)), ')']



def Mul(a,b):
    return ['(', str(''.join(a)), ' * ', str(''.join(b)), ')']


def Div(a,b):

    return ['(', str(''.join(a)), ' / ', str(''.join(b)), ')']


def Sqr(a,b):

    return ['(', str(''.join(a)), ' ** ', str(''.join(b)), ')']


def Srt(a):

    return ['(', str(''.join(a)), ' ** ', str(0.5), ')']


def Min(a,b):

    return ['min(', str(''.join(a)), ' , ', str(''.join(b)), ')']


def Max(a,b):

    return ['max(', str(''.join(a)), ' , ', str(''.join(b)), ')']


def Avg2(a,b):

    return ['((', str(''.join(a)), ' + ', str(''.join(b)), ')/2)']


def Sin(a):

    return ['sin(', str(''.join(a)), ')']


def Cos(a):

    return ['cos(', str(''.join(a)), ')']


# Boolean functions------------------------------------------------------------

def And(a,b):

    return ['(', str(''.join(a)), ' and ', str(''.join(b)), ')']


def Or(a,b):

    return ['(', str(''.join(a)), ' or ', str(''.join(b)), ')']
```

```
def Not(a):
    return ['not(', str(''.join(a)), ')']

# ==============================================================================

def avg(x):
    """Get the average value of a list."""

    return sum(x)/len(x)

# ==============================================================================

def isnumber(x):
    ''' Return true if x is number and false otherwise'''

    try:
        float(x)
        return True
    except:
        return False

# ==============================================================================

def num_index(x):
    ''' Return indexes of number elements in a list'''

    y = [i for i in range(len(x)) if isnumber(x[i]) ]

    return y

# ==============================================================================
def get_n_arg(func):
    """ Get the number of argument of a function
        <func> should be in string type
    """

    return len(getargspec(eval(func))[0])

# ==============================================================================

def get_data(file_name, sheet_name, n_input, n_output):
    ''' Get data from a specified excel file '''

    # Loading whorksheet and related sheet_name
    wb = load_workbook(file_name)
    sheet = wb.get_sheet_by_name(sheet_name)

    max_row = sheet.max_row
    in_data = {}
    out_data = {}

    # Getting input and output title
    input_title = [sheet.cell(row = 1, column = i).value \
                   for i in range(1,n_input+1)]
    output_title = [sheet.cell(row = 1, column = i).value \
                    for i in range(n_input + 1,n_input+1+n_output)]

    # Getting input data
    for i in range(n_input):
        in_data[input_title[i]]=[sheet.cell(row = j, column = i+1).value\
                                 for j in range(2,max_row+1)]
    # Getting output data
    for i in range(n_output):
        out_data[output_title[i]]=[sheet.cell(row = j, column = i+n_input+1).value \
                                   for j in range(2,max_row+1)]

    return in_data, out_data

# =============================================================================

def t_series_data(file_name, sheet_name, cycle, n_period, normal):
    ''' Get time series data from a specified excel file '''

    # Loading whorksheet and related sheet_name
    wb = load_workbook(file_name)
    sheet = wb.get_sheet_by_name(sheet_name)

    max_row = sheet.max_row

    # Getting data
    data=[sheet.cell(row = j, column = 6).value for j in range(2,max_row+1)]
    data = data[len(data)-n_period:]

    #Normalize data
    if normal:
        max_data = max(data)
        min_data = min(data)
        data = [0.1 + 0.8*(i-min_data)/(max_data - min_data) for i in data]

    # Getting output data
    in_data, out_data = {}, {}
    for period in range(cycle):
        in_data['Lag'+str(cycle-period)]=data[period:len(data)-cycle+period]
```

```
            print('Lag'+str(cycle-period),in_data['Lag'+str(cycle-period)][1:10])

        out_data = {'Y':[i for i in data[cycle:]]}

        return in_data, out_data

# ============================================================================

def c_range(input, output):
    """Determine constant range base on input and output value."""

    min_diff = [min(output)/min(i) for i in input]
    max_diff = [max(output)/max(i) for i in input]
    range_diff = [(max(output)-min(output))/(max(i)-min(i)) for i in input]

    LB = min(0,min(min(min_diff),min(max_diff),min(range_diff)))
    UB = min(10,abs(max(min(min_diff),min(max_diff),min(range_diff))))

    cons_range = [LB,UB]
    return cons_range

# ============================================================================

def intersection(list1, list2):
    ''' Get the intersection of two lists.'''

    list3 = [value for value in list1 if value in list2]

    return list3

#============================================================================
def find_in(x,y):

    ''' find indexes of element x in list y'''


    return [i for i in range(len(y)) if y[i] == x]


#==============================================================================


def init_chrom(function, terminal, w_rand, l_head, l_tail, l_chrom,
cons_range,first_func,encoding):

    ''' Create initial chromosome'''


    if cons_range != []:

        terminal = terminal + len(terminal)*['C']


    if encoding == 'sargep': #No head and tail for sargep

        l_chrom_min = 4 # minimum gene length

        l_chrom = randrange(l_chrom_min,l_chrom+1)

        chrom = choices(function + terminal, weights = w_rand, k = l_chrom)


    elif encoding =='robust': #No head and tail for robust

        chrom = choices(function + terminal, weights = w_rand, k = l_chrom)


    else: # for original GEP, using karva encoding

        # Creating head and tail

        head = choices(function + terminal, weights = w_rand, k = l_head)

        tail = choices(terminal, k = l_tail)

        chrom = head + tail


    if cons_range != []: # Replace 'C' with random constant value

        for C in find_in('C',chrom):

            chrom[C] = str(uniform(cons_range[0],cons_range[1]))


    if first_func: # if first function is true, the first bit has to be a function
```

```
        chrom[0] = choice(function)

    return chrom

# ============================================================================

def init_pop(pop_size, function, terminal, w_rand, l_head, l_tail, l_chrom, cons_range, 
first_func, encoding):
    ''' Create initial population'''

    pop = [init_chrom(function, terminal, w_rand, l_head, l_tail, l_chrom, cons_range, 
first_func, encoding)\
           for i in range(pop_size)]

    return pop

# ============================================================================

def chrom_trim(chrom, func, l_head):
    '''
    Make the first symbol to be function

    '''

    # Delete symbols before the first function and return the first symbol if no function 
in the t_chromosome

    first = 0
    for n in range(l_head):
        try:
            func.index(chrom[n])
            first = n
            break
        except:
            pass

    t_chrom = chrom[0] if first == 0 else chrom[first:]

    return t_chrom

# ==============================================================================

def karva(chrom, func, l_head):
    ''' Convert crhomosome to equation using Karva expression'''

    chrom = chrom[:]

    point = 0
    l_expr = 1
    expression = []
    expression.append(chrom[0])
    chrom.pop(0)

    while point < l_expr:
        if expression[point] in func:

            # Determining input argument
            arg = ''
            n_arg = get_n_arg(expression[point])
            for i in range(n_arg):
                if i == 0:
                    arg = arg + '(\''
                else:
                    arg = arg + '\',\''
                arg = arg + chrom[0]
                chrom.pop(0)
            arg = arg + '\')'

            # Creating function replacement
            f_replace = eval(expression[point] + arg)

            # Replace function symbol with readable function
            expression.pop(point)
            for i in range(len(f_replace)):
                expression.insert(point+i, f_replace[i])

        point += 1
        l_expr = len(expression)

    equation = ''.join(expression)
    return equation

# ============================================================================

def prefix(chrom, func, l_head):
    ''' Convert crhomosome to equation using Prefix encoding'''

    expression = []
    arguments = []

    for i in range(len(chrom)):
        if i>0 and len(arguments) == 0: break

        if chrom[i] in func:
            expression.append(chrom[i])
            if len(arguments) > 0:
                arguments[-1] -= 1
            arguments.append(get_n_arg(chrom[i]))
```

```
                expression.append('(')


            else:

                expression.append('\'' + chrom[i] + '\'' )

                if len(arguments) > 0:

                    arguments[-1] -= 1


                    while arguments[-1] == 0:

                        arguments.pop(-1)

                        expression.append(')')

                        if len(arguments) == 0: break


                if len(arguments) > 0:

                    expression.append(',')


    expression = ''.join(expression)

    expression = ''.join(eval(expression))


    return expression


# ============================================================================


def robust(chrom, func, l_head):

# Convert crhomosome to equation using Robust GEP


    # Avoid the original input value from being changed

    chrom = chrom[:]


    # Return the first symbol if no function in the chromosome

    if intersection(chrom,func) == []:

        chrom = [chrom[0]]
    else:

        # Removing all bit before the first function

        while True:

                try:

                    func.index(chrom[0])

                    break

                except:

                    chrom.pop(0)


    # Removing all functions those do not have enough arguments

    arg_left = 0

    for i in range(len(chrom)-1,-1,-1):

        if chrom[i] in func:

                if arg_left < get_n_arg(chrom[i]):

                    chrom.pop(i)

                else:

                    arg_left = arg_left - get_n_arg(chrom[i]) + 1

        else:

                arg_left += 1


    # Assign zero if empty

    if chrom == []:

        chrom = ['0']


    # Encode the chromosome after trimmed

    expression = prefix(chrom, func, l_head)


    return expression


# ==============================================================================

def sargep(chrom, func, l_head):

# Convert crhomosome to equation using Robust GEP
```

```
    return robust(chrom, func, l_head)


# ==============================================================================

def show_chrom(chrom):

# Show chromosome in a nice format

    return '( ' + ' '.join(chrom) + ' )'


# ==============================================================================


def eval_eq(equation, input_data):

# Return result of an equation with giving inputs


    result = []

    var_name = list(input_data.keys())


    for i in range(len(input_data[var_name[0]])):

        eq = equation

        try:

            for j in var_name:

                eq=eq.replace(j,str(input_data[j][i]))

            result.append(eval(eq))

        except:

            result.append(0) # if evaluation is fail, return 0


    return result


# ==============================================================================


def selection(fitness, method, n_parent):

# Select a number of parents from a given population using a certain method
```

```
    if method == 'RW': # Roulete Wheel

        index = [i for i in range(len(fitness))]

        parent_index = choices(index, weights = fitness, k = n_parent)



    elif method == 'adjusted-RW': # Weighted Roulete Wheel

        adj_fitness = [i + avg(fitness) for i in fitness]

        index = [i for i in range(len(fitness))]

        parent_index = choices(index, weights = adj_fitness, k = n_parent)



    elif method == 'DRW': # Distributed Roulete Wheel, without replacement

        index = [i for i in range(len(fitness))]

        sum_fit = sum(fitness)

        adj_fitness = [i/sum_fit for i in fitness]

        parent_index = numpyChoice(index,size=n_parent,replace=False, p=adj_fitness)



    return parent_index



# ============================================================================



def mutation(method, l_head, l_tail, function, terminal, w_rand, cons_range, parent,
encoding):

# perform mutation of an individual



    mutan = parent[:]



    if method == 'one-point':

        point = randrange(len(mutan))



        # If using constant

        if cons_range != []:

            terminal = terminal + len(terminal)*['C']
        if encoding in ['sargep' , 'robust2']: # No head and tail section for sargep
Encoding

            mutan[point] = choices(function + terminal, weights = w_rand)[0]


        elif encoding == 'robust': # No head and tail section for Robust Encoding

            mutan[point] = choice(function + terminal)


        else:

            if point == 0:

                mutan[point] = choice(function)

            elif point < l_head:

                mutan[point] = choice(function + terminal)

            else:

                mutan[point] = choice(terminal)


        # Replace char 'C' with random constant

        if mutan[point] == 'C':

            mutan[point] = str(uniform(cons_range[0],cons_range[1]))




    elif method == 'two-point':

        n_mut = 2

        for i in range(n_mut):

            mutation('one-point', l_head, l_tail, function, terminal, w_rand, cons_range,
parent, encoding)


    return mutan


# ============================================================================


def constant_mutation(parent, max_change):

    mutan = parent[:]

    c_index = num_index(parent)
    if c_index != []:

        point = randrange(len(c_index))

        mutan[c_index[point]] = str(float(parent[c_index[point]])*(1+uniform(-
max_change,max_change)))


    return mutan


# ============================================================================


def inversion(l_head, parent,encoding):

# perform inversion of an individual


    max_inv = 3

    mutan = parent[:]

    l_chrom = len(mutan)


    if encoding in ['robust', 'robust2','sargep']:

        start = randrange(l_chrom)

        end = randrange(start+1,min(l_chrom+1,start+max_inv+1))


    else:

        start = randrange(1,l_head)

        end = randrange(start+1,min(l_head+1,start+max_inv+1))


    mutan[start : end] =  mutan[end-1 : start : -1]+mutan[start:start+1]


    return mutan


# ============================================================================


def IS(l_head,parent,function,encoding):

# perform inversion of an individual
```

```
mutan = parent[:]

max_IS = 3 # Maximum length of IS

l_IS = randrange(1,max_IS+1)

start = randrange(len(parent)-l_IS)

end = start + l_IS


section = parent[start : end] # Copy the section for inserting


if encoding in ['robust', 'robust2','sargep']:

    del(mutan[start:end])

    try:

        point = randrange(len(mutan))

    except:

       point = 0

    mutan[point:point] = section


else:

    mutan_head = parent[:l_head]

    del(mutan_head[start:end])


    # if the first section is function, section can be inserted anywhere

    # if the first section is terminal, section cannot be inserted at first location

    a = 0 if section[0] in function else 1

    try:

        point = randrange(a,len(mutan_head))

    except:

        point = 0


    # Insert section

    mutan_head[point:point] = section

    mutan[:l_head] = mutan_head[:l_head]
    return mutan


# =============================================================================


def length adjustment(l_chrom, parent, function, terminal, w_rand, cons_range, randir):
# perform length adjustment of an individual


    mutan = parent[:]
    l_parent = len(parent)
    point = randrange(l_parent)
    min_length = 4 # min chrom length


    # If using constant
    if cons_range != []:
        terminal = terminal + len(terminal)*['C']



    if randir > 0:
        mutan[point:point] = [choices(function + terminal, weights=w_rand)][0]


        # Replace char 'C' with random constant
        if mutan[point] == 'C':
            mutan[point] = str(uniform(cons_range[0],cons_range[1]))



    elif l_parent > min_length:
        mutan.pop(point)


    return mutan



# =============================================================================
def recombination(parent1,parent2,method):

# perform recombination or crossover of an individual


    mutan1 = parent1[:]

    mutan2 = parent2[:]


    if mutan1 == mutan2 and method is not 'slice':

        pass


    else:


        l1, l2 = len(mutan1), len(mutan2)

        min_len = min(l1,l2)


        if method == 'slice':

            l_section = randrange(1,min_len)

            point1 = randrange(l1-l_section)

            point2 = randrange(l2-l_section)

            mutan1[point1:point1+l_section] = parent2[point2:point2+l_section]

            mutan2[point2:point2+l_section] = parent1[point1:point1+l_section]


        elif method == 'flexi-slice':

            l_section1 = randrange(1,l1)

            l_section2 = randrange(1,l2)

            if (l1 - l_section1 + l_section2)>=4 and \

                (l2 - l_section2 + l_section1)>=4 and \

                    (l1 - l_section1 + l_section2)<=200 and \

                        (l2 - l_section2 + l_section1)<=200:

                point1 = randrange(l1-l_section1)

                point2 = randrange(l2-l_section2)

                copy1 = mutan1[point1:point1+l_section1]

                copy2 = mutan2[point2:point2+l_section2]
                del mutan1[point1:point1+l_section1]

                del mutan2[point2:point2+l_section2]

                mutan1[point1:point1] = copy2

                mutan2[point2:point2] = copy1


        elif method == 'one-point':

            point = randrange(min_len)

            mutan1[point:] = parent2[point:]

            mutan2[point:] = parent1[point:]


        elif method == 'two-point':

            point1 = randrange(min_len-2)

            point2 = randrange(point1+1, min_len)

            mutan1[point1:point2] = parent2[point1:point2]

            mutan2[point1:point2] = parent1[point1:point2]




    return mutan1, mutan2


# ==============================================================================


def fit_eval(result, target, method, R, p): # R is the selection range

# Calculate fitness using a specified method


    perf = 0

    fit = 0

    percentage = False


    if method in ['n_hits_p', 'MAPE', 'APE']: percentage = True


    for i in range(len(target)):
```

```
E = result[i]-target[i]

AE = abs(E)

if AE < (p*target[i]):      AE = 0


# Calculating Percentage Error

if percentage:

    if target[i] != 0:          PE = 100 * E / target[i]

    elif result[i] == target[i]:PE = 0

    else:                       PE = 100 * E / (sum(target)/len(target))

    APE = abs(PE)

    if APE > 100: APE = 100

    if APE < p*100: APE = 0


if method == 'n_hits': # number of hits method

    perf = perf + (1 if AE == 0 else 0)

    fit = perf


elif method == 'n_hits_p': # number of hits (percentage) method

    perf = perf + (1 if APE == 0 else 0)


elif method == 'MAPE': # Mean Absolute Percentage Error method

    if i == 0: sum_APE = 0

    sum_APE = sum_APE + APE


elif method == 'MSE': # Mean Square Error method

    SE = AE**2

    if i == 0: sum_SE = 0

    sum_SE = sum_SE + SE


elif method == 'NSE': # Normalized Square Error method

    SE = AE**2

    st_dev = stdev(target)
```

```
                NSE = SE/st_dev

                if i == 0: sum_NSE = 0

                sum_NSE = sum_NSE + NSE


            elif method == 'AE': # Absolute Error method

                fiti = (R - AE) if (R - AE)>0 else 0

                fit = fit + fiti


            elif method == 'APE': # Absolute Percentage Error method

                fiti = (R - APE) if (R - APE)>0 else 0

                fit = fit + fiti


        if method == 'MAPE':

            perf = sum_APE / len(target)

            fit = 100 - perf


        if method == 'MSE':

            perf = sum_SE / len(target)

            fit = R / (1 + perf)


        if method == 'n_hits_p':

            perf = 100 * perf / len(target)

            fit = perf


        if method == 'NSE':

            perf = sum_NSE / len(target)

            fit = 1 / (1 + perf)



        return fit, perf


    # ============================================================================
    def main(data, function, par, run):


    # Parameters

        # Determining max number of arguments in the functions

        n_arg = [get_n_arg(i) for i in function]

        max_arg = max(n_arg)



        # Default parameters:

        l_head      = par.get('l_head'      , 10 )                      # length of head

        l_tail      = l_head * (max_arg -1) +1                          # Length of tail

        l_chrom     = par.get('l_chrom'     , l_head + l_tail )         # Length of chomosome

        first_func  = par.get('first_func'  , True )                    # If True, the first
    gen is always functions in initial population

        pop_size    = par.get('pop_size'    , 100 )                     # Population size

        p_mut       = par.get('p_mut'       , 2 / l_chrom )             # Probability of
    mutation

        p_c_mut     = par.get('p_c_mut'     , 0.2 )                     # Probability of
    constant mutation

        p_inv       = par.get('p_inv'       , 0.1 )                     # Probability of
    inversion

        p_IS        = par.get('p_IS'        , 0.1 )                     # Probability of
    transposition or Insertion Sequence (IS)

        p_con       = par.get('p_contract'  , 0.2 )                     # Probability of
    length adjustment (up and down)

        p_recom     = par.get('p_recom'     , 0.9 )                     # Probability of
    recombination or crossover

        parent_sel_method   = par.get('parent_sel_method'   , 'RW' )    # Parent selection
    method for crossover

        gen_sel_method      = par.get('gen_sel_method'   , 'DRW' )      # Selection method for
    next generation

        mut_method  = par.get('mut_method'  , 'two-point' )             # Mutation method

        recom_method= par.get('recom_method', 'two-point' )             # Recombination or
    crossover method

        max_gen     = par.get('max_gen'     , 1000 )                    # Max generation

        best_fitness= par.get('best_fitness', 100 )                     # Best possible
    fitness (perfect solution)

        encoding    = par.get('encoding'    , 'karva' )                 # Gen encoding method
        perf        = par.get('perf'        , 'MAPE' )                  # Accuration
    performace measurement

        R           = par.get('R'           , 100 )                     # Selection Range

        p           = par.get('p'           , 0 )                       # Precision

        cons_range  = par.get('cons_range'  , [])                       # Range of constants

        with_const  = par.get('with_const'  , False )                   # If True, using
    cnstants

        max_c_mut   = par.get('max_c_mut'   , 0.1 )                     # Maximum range of
    constant mutation (up and down)

        rand_weight = par.get('rand_weight' ,'uniform')                 # Random weight for
    generate gene: uniform and adjusted


        # Data

        file_name = data['file']

        sheet_name = data['sheet']


        # For stock time series prediction

        ############################################

        # cycle = data['cycle']

        # n_period = data['n_period']

        # normalize = data['normalize']

        # input_data, output_data = t_series_data(file_name, sheet_name, cycle, n_period,
    normalize)

        # ############################################


        n_input = data['n_input']

        n_output = data['n_output']


        input_data, output_data = get_data(file_name, sheet_name, n_input, n_output)

        terminal = list(input_data.keys())

        Y  = output_data['Y']


        # Determining constant range

        if with_const and cons_range == []:

            c_input = [input_data[i] for i in terminal]
            cons_range = c_range(c_input, Y)


        # Determining weight of function and terminal for random

        l_func = len(function)

        l_term = len(terminal) if cons_range == [] else 2*len(terminal) # If with constant,
    the terminal is doubled for constants


        if rand_weight == 'adjusted':

            w_rand = l_func*[1/l_func] + l_term*[avg([x-1 for x in n_arg])/l_term]


        else:

            w_rand = (l_func+l_term)*[1]


        #Initialization

        fit_pop = []

        perf_pop = []

        best_fit = -1

        gen = 0

        fit_count = 0


        # Generate initial population

        pop = init_pop(pop_size, function, terminal, w_rand, l_head, l_tail, l_chrom,
    cons_range, first_func, encoding)


        # Calculating fitness of the initial population

        for j in range(pop_size):

            eq = eval(encoding)(pop[j],function,l_head)

            result = eval_eq(eq, input_data)

            fitness, performance = fit_eval(result,Y,perf,R,p)

            fit_pop.append(fitness)

            perf_pop.append(performance)

            fit_count += 1
        # Get maximum fitness

        best_fit = max(fit_pop)

        best_index = fit_pop.index(best_fit)

        best_perf = perf_pop[best_index]

        best_pop = pop[best_index]



        for gen in range(max_gen): #Main iteration



            print('L_CHROM:', l_chrom, ' RUN:', run+1, ' Best Fitness gen-', gen ,': ',
    "%0.2f" %best_fit, \

                  ' Fitness mean: ', "%0.2f" %(sum(fit_pop)/pop_size), ' Best ', perf ,': ',
    "%0.2f" %best_perf)

            print()



            # Break if find optimal solution

            if best_fit >= best_fitness: break




            # candidate selection

            candidate = []



            for i in range(pop_size):



                #Mutation

                a = random()

                if a < p_mut:

                    new_candidate = mutation(mut_method, l_head, l_tail, function, terminal,
    w_rand, cons_range, pop[i], encoding)

                    candidate.append(new_candidate)



                # Inversion

                a = random()

                if a < p_inv:

                    new_candidate = inversion(l_head, pop[i],encoding)
```

```
                    candidate.append(new_candidate)

                # Transposition
                a = random()
                if a < p_IS:
                    new_candidate = IS(l_head,pop[i],function,encoding)
                    candidate.append(new_candidate)

                # Length adjustment
                if encoding is 'sargep' and p_con > 0 :
                    a = random()
                    if a < p_con:
                        randir = 1 if a > (p_con/2) else -1
                        new_candidate = length adjustment(l_chrom, pop[i], function, terminal,
w_rand, cons_range, randir)
                        candidate.append(new_candidate)

                # Constant Mutation
                if with_const:
                    a = random()
                    if a < p_c_mut:
                        new_candidate = constant_mutation(pop[i], max_c_mut)
                        candidate.append(new_candidate)

            # Recombination or crossover
            parent_index = selection(fit_pop, parent_sel_method, pop_size)
            parent = [pop[i] for i in parent_index]
            child = parent[:]
            for i in range(0,pop_size,2):
                a = random()
                if a < p_recom:
                    child[i], child [i+1] = recombination(parent[i],parent[i+1],recom_method)
                    candidate.append(child[i])
```

```
        candidate.append(child[i+1])

# evaluating fitness of the candidates
fit_can = []
perf_can = []
for j in range(len(candidate)):
    eq = eval(encoding)(candidate[j],function,l_head)
    result = eval_eq(eq, input_data)
    fitness, performance = fit_eval(result,Y,perf,R,p)
    fit_can.append(fitness)
    perf_can.append(performance)
    if best_fit < best_fitness: fit_count += 1

# Get maximum fitness
max_fit = max(fit_can)
max_index = fit_can.index(max_fit)
max_perf = perf_can[max_index]
max_candidate = candidate[max_index]
if max_fit > best_fit:
    best_fit = max_fit
    best_perf = max_perf
    best_pop = max_candidate[:]


# Select individual for next generation
pool = pop + candidate
fit_pool = fit_pop + fit_can
perf_pool = perf_pop + perf_can
chosen_index = selection(fit_pool, gen_sel_method, pop_size)
pop = [pool[i] for i in chosen_index][:]
fit_pop = [fit_pool[i] for i in chosen_index][:]
perf_pop = [perf_pool[i] for i in chosen_index][:]
```

```
    # elitism
    if max(fit_pop) < best_fit:
        a = randrange(pop_size)
        pop[a] = best_pop[:]
        fit_pop[a] = best_fit
        perf_pop[a] = best_perf
best_eq = eval(encoding)(best_pop,function,l_head)
print('Original best equation: ', best_eq)
best_eq = str(simplify(best_eq))
print('Best chromosome: ', show_chrom(best_pop))
print('Best equation: ', best_eq)

return best_fit, best_perf, best_eq, fit_count, gen, l_chrom
```

Attachment 2: MATLAB code

ExperimentALGEPvsANN.m

```
clear;
clc;

%GENERAL PARAMETERS
%###############################################################
##############
file_name = "DAILY BBRI DATA";
p_val = 0.2; %proportion of validation data
max_gen = 200; %Max generation (for ALGEP) or max epoch (for
ANN)
stag_stop = 5000;

%ALGEP PARAMETERS
%###############################################################
##############
%Determining functions
func = ["Adc","Sub","Mul","Div"];

%Determining parameters
param.max_gen = max_gen;
param.length_range = [5 25];
param.n_genes = 4;
param.stag_stop = stag_stop;
%###############################################################
##############

%ANN PARAMETERS
%###############################################################
##############
hidden_layer    = [12 6 3];       % hidden layer
AF              = 'binsig';     % Activation Function
max_epoch       = max_gen;            % maximum Epoch
rand_train      = 0;
init_weight     = [];
%###############################################################
##############

%Collecting data
%###############################################################
##############
%Read data
data_table = readtable(file_name);

%Read input and output
all_var = string(data_table.Properties.VariableNames);
data.input_name = all_var(1:end-1);
data.output_name = all_var(end);

%Input and output value
data_input = zeros(size(data_table,1),size(data_table,2)-1);
for i=1:length(data.input_name)
    data_input(:,i) = data_table.(data.input_name(i));
```

```
end
data_output=data_table.(data.output_name);

% Deviding training and validation data
n_data = length(data_output);
n_train = round(n_data*(1-p_val));
rand_ind = randperm(n_data);
data.in_train = data_input(rand_ind(1:n_train),:);
data.out_train = data_output(rand_ind(1:n_train),:);
data.in_val = data_input(rand_ind(n_train+1:end),:);
data.out_val = data_output(rand_ind(n_train+1:end),:);
%##############################################################
##############

%File name
save_file  =  "D:\Disertasi\MATLAB\ALGEP\EXPERIMENT  ALGEP  ANN
UNTUK  "  +  file_name  +"  "+  datestr(now,'dd-mm-yy  HH-MM')  +
".xlsx";

%Running ALGEP
%##############################################################
##############
start = cputime;

[perf_table,  best_perf,  best_eq,  best_val_perf,  best_val_eq,
fit_count, gen] = ALGEP4(data, func, param);

time = cputime-start;

%Save ALGEP performance in excel
writetable(perf_table,save_file,'Sheet','ALGEP PERFORMANCE');
resume_algep   =   table(best_perf,   best_eq,   best_val_perf,
best_val_eq, fit_count, gen, time);
writetable(resume_algep,save_file,'Sheet','RESUME ALGEP');

%Running ANN
%##############################################################
##############
start = cputime;

[epochs,best_epoch,best_weight,ANN_perf,best_ANN_perf,ANN_val_
perf,best_weight_val,best_ANN_val_perf,output_ANN,output_val_A
NN] = ...

BP6(data.in_train,data.out_train,data.in_val,data.out_val,init
_weight,AF,rand_train,max_epoch,hidden_layer, stag_stop);

time = cputime-start;

%Save ANN performance in excel
epochs  =  (1:epochs)';  ANN_perf  =ANN_perf';  ANN_val_perf  =
ANN_val_perf';
ANN_perf_table = table(epochs,ANN_perf,ANN_val_perf);
writetable(ANN_perf_table,save_file,'Sheet','ANN PERFORMANCE');
resume_ANN  =  table(best_epoch,best_ANN_perf,best_ANN_val_perf,
time);
```

```
writetable(resume_ANN,save_file,'Sheet','RESUME ANN');

% Finising alarm
load gong.mat;
sound(y);
```

ALGEP4:

```
function   [perf_table,   best_perf,   best_eq,   best_val_perf,
best_val_eq, fit_count, gen] = ...
    ALGEP2(data, func, param)
%  [best_fit, best_perf, best_eq, fit_count, gen, l_chrom]  =
ALGEP2(data, func, par, run, save_file)

%Default parameters
par.length_range     = [10 100];
par.first_func       = true;
par.pop_size         = 100;
par.n_genes          = 1;
par.p_mut            = 0.1;
par.p_cons_mut       = 0.1;
par.p_inv            = 0.1;
par.p_trans          = 0.1;
par.p_crossover      = 0.9;
par.mut_point        = 1;
par.max_cons_mut     = 0.1;
par.max_trans        = 4; %max length of transposition
par.crossover_method= "FLEXI-SLICE";
par.max_gen          = 100; %Max Generation
par.max_fitness      = 100; %Max fitness value to stop iteration
par.perf_method      = "MAPE";
par.R                = 100;
par.p                = 0;
par.stag_stop        = 1000;
par.cons_range       = [-1 1];
par.gene_combine     = "+";

%Replace the default parameters with custom parameters
custom_param = string(fieldnames(param));
for i=1:length(custom_param)

par=setfield(par,custom_param(i),getfield(param,custom_param(i
)));
end

%Eliminating par name in the variable name
data_field = string(fieldnames(par));
for i=1:length(data_field)
    eval(data_field(i)+"=par."+data_field(i)+";");
end

%Eliminating data name in the variable name
data_field = string(fieldnames(data));
for i=1:length(data_field)
    eval(data_field(i)+"=data."+data_field(i)+";");
```

```
end

%Initiation training and validation value to handle function
[in_train_handle,                in_val_handle]               =
deal(strings(1,length(input_name)));
for i = 1:length(input_name)
    eval("in_train"+i+"=in_train(:,"+ i + ");");
    eval("in_val"+i+"=in_val(:,"+ i + ");");
    in_train_handle(i) = "in_train" + i;
    in_val_handle(i) = "in_val" + i;
end
in_train_handle = join(in_train_handle,",");
in_val_handle = join(in_val_handle,",");

%if not using constant
if p_cons_mut == 0
    cons_range = [];
end

%Determining the number of arguments in each function
n_arg = cellfun(@nargin,func);
avg_arg = mean(n_arg);

%Defining terminal
terminal = input_name;

%If using constant
if ~isempty(cons_range)
    terminal                    =                    [terminal
strings(1,round(length(terminal)/2))+"C"];
end

%Random Weight
l_func = length(func);
l_term = length(terminal);
w_rand = [ones(1,l_func) ones(1,l_term)*avg_arg*l_func/l_term];

%Create a handle function
handle_func = "@(" + strjoin(input_name,",") + ")";

%initial population
pop                                                           =
init_pop(func,terminal,pop_size,n_genes,length_range,cons_rang
e,first_func,w_rand);

%Setting initial values
eq_pop = strings(1,pop_size);
[fit_pop, perf_pop] = deal(zeros(1,pop_size));
[perf_data, val_perf_data] = deal(zeros(max_gen+1,1));
[fit_count, n_mut, n_inv, n_trans, n_cons_mut, stag] = deal(0);

% Calculating fitness of the initial population
for i=1:pop_size
    for j = 1:n_genes
        eq = encoding(pop{i,j},func,n_arg);
        eq_pop(i) = eq_pop(i) + gene_combine + eq;
    end

    func_handle=str2func(handle_func + eq_pop(i));
    eval("result = func_handle("+in_train_handle+");");
    [perf_pop(i),                  fit_pop(i)]                  =
fit_eval(result,out_train,perf_method);
    fit_count = fit_count + 1;
end

%Finding best individual in the population
[best_fit, best_index] = max(fit_pop);
best_pop = pop(best_index,:);
best_perf = perf_pop(best_index);
best_eq = eq_pop(best_index);

%Finding validation performance
func_handle=str2func(handle_func + best_eq);
eval("val_result = func_handle("+in_val_handle+");");
[best_val_perf,                   val_fit]                     =
fit_eval(val_result,out_val,perf_method);

try %simplify best equation
    best_eq = string(str2sym(best_eq));
catch
end

%Saving best validation equation
best_val_eq = best_eq;

%ALGEP ITERATION
%###############################################################
##############
for gen = 0:max_gen

    %Displaying the performance of each generation
    disp("Gen-" + gen + " BEST MAPE:" + best_perf + " BEST VAL
MAPE:" + best_val_perf ...
            + "--> " + output_name + " = " + best_eq)% ...
            %+ "--> Best/Avg length: " + length(best_pop) + "/"
+ mean(cellfun(@length,pop)));

    %Saving MAPE
    perf_data(gen+1) = best_perf;
    val_perf_data(gen+1) = best_val_perf;

    %Stop if getting max_fitness
    stag = stag + 1;
    if best_fit > max_fitness || stag > stag_stop
        break;
    end

    %Mutant prealocation
    %Mutation
    if p_mut > 0
        rand_mut = rand(1,pop_size);
        mut_index = find(rand_mut < p_mut);
        n_mut = length(mut_index); % number of mutation
    end

    %Inversion
    if p_inv > 0
        rand_inv = rand(1,pop_size);
        inv_index = find(rand_inv < p_inv);
        n_inv = length(inv_index); %number of inversion
    end

    %Transposition
    if p_trans > 0
        rand_trans = rand(1,pop_size);
        trans_index = find(rand_trans < p_trans);
        n_trans = length(trans_index); %number of transposition
    end

    %Constant mutation
    if p_cons_mut > 0
        rand_cons_mut = rand(1,pop_size);
        cons_mut_index = find(rand_cons_mut < p_cons_mut);
        n_cons_mut   =   length(cons_mut_index);   %   number   of
constant mutation
    end

    %Create mutan (prealocation)
    mutan = cell(n_mut + n_inv + n_trans + n_cons_mut + pop_size,
n_genes);
    mutan_size = length(mutan);
    mutan_pointer = 1;

    %MUTATION
    for i = 1 : n_mut
        gene_point = randi(n_genes);
        mutan(mutan_pointer,:) = pop(mut_index(i),:);
        mutan{mutan_pointer,gene_point}                        =
mutation(pop{mut_index(i),gene_point},mut_point,[func
terminal], cons_range, w_rand);
        mutan_pointer = mutan_pointer + 1;
    end

    %INVERSION
    for i = 1 : n_inv
        gene_point = randi(n_genes);
        mutan(mutan_pointer,:) = pop(inv_index(i),:);
        mutan{mutan_pointer,gene_point}                        =
inversion(pop{inv_index(i),gene_point});
        mutan_pointer = mutan_pointer + 1;
    end

    %TRANSPOSITION
    for i = 1 : n_trans
        gene_point = randi(n_genes);
        mutan(mutan_pointer,:) = pop(trans_index(i),:);
        mutan{mutan_pointer,gene_point}                        =
transposition(pop{trans_index(i),gene_point},max_trans);
```

```
            mutan_pointer = mutan_pointer + 1;
        end

        %CONSTANT MUTATION
        for i = 1 : n_cons_mut
            gene_point = randi(n_genes);
            mutan(mutan_pointer,:) = pop(cons_mut_index(i),:);
            mutan{mutan_pointer,gene_point}                    =
cons_mutation(pop{cons_mut_index(i),gene_point},max_cons_mut);
            mutan_pointer = mutan_pointer + 1;
        end

        %CROSSOVER
        parent_index = parent_selection(fit_pop,pop_size);
        for i = 1:2:pop_size
            gene_point = randi(n_genes,1,2);
            mutan(mutan_pointer,:) = pop(parent_index(i),:);
            mutan(mutan_pointer+1,:) = pop(parent_index(i+1),:);

[mutan{mutan_pointer,gene_point(1)},mutan{mutan_pointer+1,gene
_point(2)}] = ...

crossover(pop{parent_index(i),gene_point(1)},pop{parent_index(
i+1),gene_point(2)},crossover_method,length_range);
            mutan_pointer = mutan_pointer + 2;
        end

        %Setting initial values
        eq_mutan = strings(1,mutan_size);
        fit_mutan = zeros(1,mutan_size);
        perf_mutan = zeros(1,mutan_size);

        % Calculating fitness of the offsprings
        for i=1:mutan_size
            for j = 1:n_genes
                eq = encoding(mutan{i,j},func,n_arg);
                eq_mutan(i) = eq_mutan(i) + gene_combine + eq;
            end

            try
                func_handle=str2func(handle_func + eq_mutan(i));
                eval("result = func_handle("+in_train_handle+");");
            catch
                result=zeros(size(out_train));
            end

            [perf_mutan(i),              fit_mutan(i)]             =
fit_eval(result,out_train,perf_method);
            fit_count = fit_count + 1;
        end

        %Finding best individual in the offsprings (mutan)
        [best_fit_mutan, best_mutan_index] = max(fit_mutan);
        if best_fit_mutan > best_fit
            best_pop = mutan(best_mutan_index,:);
            best_fit = best_fit_mutan;
            best_perf = perf_mutan(best_mutan_index);
            best_eq = eq_mutan(best_mutan_index);

            %Finding validation performance
            func_handle=str2func(handle_func + best_eq);
            eval("val_result = func_handle("+in_val_handle+");");
            [val_perf,                  val_fit]                  =
    fit_eval(val_result,out_val,perf_method);

            try %simplify best equation
                best_eq = string(str2sym(best_eq));
            catch
            end

            %Saving best validation equation
            if val_perf < best_val_perf
                best_val_perf = val_perf;
                best_val_eq = best_eq;
                stag = 0; %Reset the stagnant generation counter
            end

        end

        % Select individual for next generation
        pool = [pop; mutan];
        fit_pool = [fit_pop fit_mutan];
        perf_pool = [perf_pop perf_mutan];
        gen_index = gen_selection(fit_pool,pop_size);
        for i = 1:pop_size
            pop(i,:) = pool(gen_index(i),:);
            fit_pop(i) = fit_pool(gen_index(i));
            perf_pop(i) = perf_pool(gen_index(i));
        end

        %ELITISM
        if max(fit_pop) < best_fit
            insert_point = randi(pop_size);
            pop(insert_point,:) = best_pop;
            fit_pop(insert_point) = best_fit;
            perf_pop(insert_point) = best_perf;
        end

    end %End ALGEP iteration

    disp("BEST VAL EQUATION: " + output_name + " = " + best_val_eq);

    %Eliminate unnecessary data
    perf_data(gen+2:end) = [];
    val_perf_data(gen+2:end) = [];

    %Save file
    num_gen = (0:gen)';
    perf_table = table(num_gen,perf_data,val_perf_data);
```

BP6.m

```
function
[epoch,bestEpochVal,BestWeight,MAPEi,BestMAPE,MAPEVali,BestWei
ghtVal,BestMAPEVal,output,outputVal] = ...

BP6(InTrain,OutTrain,InVal,OutVal,initW,AF,rand_train,maxEpoch
,hLayer, stag_stop)
%
%--------------------
%Author: Samsul Amar
%--------------------
%
%
[BestWeight,BestMAPE,BestWeightVal,BestMAPEVal,output,outputVa
l] =
%    BP1(InTrain,OutTrain,InVal,OutVal,number of nodes in hidden
layer1, hidden layer2, .... )
%
% Neural Network Training fucntion using backpropagation method
%  for  a  specified   input  data  (InTrain)  and  output  data
(OutTrain)
% Number of hidden layers is unlimited
% Number of nodes for each hidden layer can be specified
% AF = Actifation Function

%Training parameters
initialWMethod   = 'random';   % method for generating initial
weight
alpa             = 0.9;                 % learning rate 0.9
momentum          = 0.2;                % mementum parameter for
updating weight 0.2
s                  = 1;                         % multiplier for
sigmoid activation function

% Stopping Criteria
stdMAPE          = 0;            % minimum MAPE data training for
stopping iteration
stdMAPEVal        = 0;         % minimum MAPE data validation for
stopping iteration

%Visualization
graphic           = 1;                  % Show graph (1) or just
text (0)
MA               = 10;                 % Moving period of MAPE

% Number of data
NTrain = size(InTrain,1);     % number of data for training
NVal = size(InVal,1);    % number of validation data
NLayer = size(hLayer,2) +1;         % number of layers

% Determining number of nodes in each layer
node = [size(InTrain,2) hLayer size(OutTrain,2)];

% Generating initial weight for each layer and preallocating
if isa(initW,'struct')
```

```
    weight = initW;
    for j = 1:NLayer
        sigmaW.layer{j} = zeros(size(weight.layer{j}));
    end

else
    for j = 1:NLayer
        weight.layer{j}                                       =
initialW(node(j),node(j+1),initialWMethod);
        sigmaW.layer{j} = zeros(size(weight.layer{j}));
    end
end

%Preallocating
X.layer{1} = [InTrain ones(NTrain,1)];
XVal.layer{1} = [InVal ones(NVal,1)];

% Initial value
[epoch, stop, stag] = deal(0);

if graphic == 1
    figure
    hold on
end

while (stop == 0)
    epoch = epoch + 1;

    %Training

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

    squence = randperm(NTrain);
    for k = 1:NTrain              %Patern iteration
        if rand_train == 1
            i = squence(k);
        else
            i = k;
        end

        % FORWARD
        for j = 1:NLayer
            Ynet.layer{j}(i,:)                                =
X.layer{j}(i,:)*weight.layer{j};
            Y.layer{j}(i,:) = actF(Ynet.layer{j}(i,:),s,AF);
            X.layer{j+1}(i,:) = [Y.layer{j}(i,:) 1];
        end

        %Error calculation
        E(i,:) = OutTrain(i,:) - Y.layer{NLayer}(i,:);

        % BACKWARD
        for j=NLayer:-1:1
            if j==NLayer
                deltanet.layer{j} = E(i,:);
            else

deltanet.layer{j}=(weight.layer{j+1}(1:node(j+1),:)*delta.laye
r{j+1}')';
            end
            delta.layer{j}                                  =
deltanet.layer{j}.*dActF(Ynet.layer{j}(i,:),s,AF);
            sigmaW.layer{j}                                 =
(alpa*delta.layer{j}'*X.layer{j}(i,:))'+
momentum*sigmaW.layer{j};
        end

        % Updating weight
        for j = 1 : NLayer
            weight.layer{j}      =      weight.layer{j}      +
sigmaW.layer{j};
        end
    end

    % Performance Calculation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

    for i = 1: NTrain       %Training
        for j = 1:NLayer
            Ynet.layer{j}(i,:)                              =
X.layer{j}(i,:)*weight.layer{j};
            Y.layer{j}(i,:) = actF(Ynet.layer{j}(i,:),s,AF);
            X.layer{j+1}(i,:) = [Y.layer{j}(i,:) 1];
        end

        error(i,:) = OutTrain(i,:) - Y.layer{NLayer}(i,:);
        percentError(i,:) = 100*error(i,:)./OutTrain(i,:);
    end

    for i = 1: NVal     % Validation
        for j = 1:NLayer
            YValnet.layer{j}(i,:)                           =
XVal.layer{j}(i,:)*weight.layer{j};
            YVal.layer{j}(i,:)                              =
actF(YValnet.layer{j}(i,:),s,AF);
            XVal.layer{j+1}(i,:) = [YVal.layer{j}(i,:) 1];
        end

        val_error(i,:) = OutVal(i,:) - YVal.layer{NLayer}(i,:);
        val_percentError(i,:)                               =
100*val_error(i,:)./OutVal(i,:);
    end

    % MAPE Calculation
    MAPE = mean(mean(abs(percentError)));

    if NVal == 0
        MAPEVal = MAPE;
    else
```

```
            MAPEVal = mean(mean(abs(val_percentError)));
        end

        %Store the best value
        if (epoch ==1 || MAPE < BestMAPE)
            BestMAPE = MAPE;
            BestWeight = weight;
            output = Y.layer{NLayer};
        end
        if (epoch ==1 || MAPEVal < BestMAPEVal)
            BestMAPEVal = MAPEVal;
            BestWeightVal = weight;
            bestEpochVal = epoch;
            stag = 0;
            if NVal == 0
                outputVal = output;
            else
                outputVal = YVal.layer{NLayer};
            end
        end

        MAPEi(epoch) = MAPE;
        MAPEVali(epoch)=MAPEVal;
        MAPEMA(epoch)= mean(MAPEi(max(1,epoch - MA + 1):epoch));
        MAPEValMA(epoch)=   mean(MAPEVali(max(1,epoch   -   MA   +
    1):epoch));


        %Show iteration
        if graphic == 1         % Graph

            if epoch < 10
                range = 1;
            elseif epoch <100
                range = 10;
            else
                range = 100;
            end

            if (mod(epoch,range) == 0 || epoch ==1)
                plot(1:epoch,MAPEMA(1:epoch),'r-
    ',1:epoch,MAPEValMA(1:epoch),'b-');
                xlabel('Epoch');
                ylabel('MAPE');
                legend(['MAPE  =  '  num2str(MAPE)  '   Best  MAPE:  '
    num2str(BestMAPE)], ['MAPE Val= ' num2str(MAPEVal) '  Best MAPE
    Val: ' num2str(BestMAPEVal)]);
                pause(0.000001);
            end
        end

        %Text
        % disp(['EPOCH: ' num2str(epoch) '   MAPE= ' num2str(MAPE)
    '        Best  MAPE=  '  num2str(BestMAPE)  '      MAPE  Val=  '
    num2str(MAPEVal) '    Best MAPE Val= ' num2str(BestMAPEVal)]);
        %Stopping condition
        stag = stag + 1;
        if (epoch >= maxEpoch) || (MAPE <= stdMAPE) || (MAPEVal <=
    stdMAPEVal) || stag > stag_stop
            stop = 1;
        end

    end

    if graphic == 1
        hold off;
    end
```

Attachment 3: SPSS output for Repeated Measure ANOVA (RMA)

RMA Analysis for Experiment 1 SRP 1

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP1	Dependent Variable
1	GEP
2	RGEP
3	ALGEP

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP1	Pillai's Trace	.938	167.072[b]	2.000	22.000	.000
	Wilks' Lambda	.062	167.072[b]	2.000	22.000	.000
	Hotelling's Trace	15.188	167.072[b]	2.000	22.000	.000
	Roy's Largest Root	15.188	167.072[b]	2.000	22.000	.000

a. Design: Intercept
Within Subjects Design: Success_Rate_SRP1
b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP1	.842	3.773	2	.152	.864	.928	.500

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.
a. Design: Intercept
Within Subjects Design: Success_Rate_SRP1
b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP1	Sphericity Assumed	17871.194	2	8935.597	158.898	.000
	Greenhouse-Geisser	17871.194	1.728	10343.808	158.898	.000
	Huynh-Feldt	17871.194	1.855	9632.803	158.898	.000
	Lower-bound	17871.194	1.000	17871.194	158.898	.000
Error(Success_Rate_SRP1)	Sphericity Assumed	2586.806	46	56.235		
	Greenhouse-Geisser	2586.806	39.738	65.097		
	Huynh-Feldt	2586.806	42.671	60.623		
	Lower-bound	2586.806	23.000	112.470		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP1	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP1	Linear	17864.083	1	17864.083	300.145	.000
	Quadratic	7.111	1	7.111	.134	.717
Error(Success_Rate_SRP1)	Linear	1368.917	23	59.518		
	Quadratic	1217.889	23	52.952		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	182911.681	1	182911.681	363.631	.000
Error	11569.319	23	503.014		

RMA Analysis for Experiment 1 SRP 2

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP2	Dependent Variable
1	GEP
2	RGEP
3	ALGEP

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP2	Pillai's Trace	.612	17.339[b]	2.000	22.000	.000
	Wilks' Lambda	.388	17.339[b]	2.000	22.000	.000
	Hotelling's Trace	1.576	17.339[b]	2.000	22.000	.000
	Roy's Largest Root	1.576	17.339[b]	2.000	22.000	.000

a. Design: Intercept

Within Subjects Design: Success_Rate_SRP2

b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP2	.682	8.413	2	.015	.759	.801	.500

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.

a. Design: Intercept

Within Subjects Design: Success_Rate_SRP2

b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP2	Sphericity Assumed	511.000	2	255.500	26.813	.000
	Greenhouse-Geisser	511.000	1.518	336.695	26.813	.000
	Huynh-Feldt	511.000	1.602	318.884	26.813	.000
	Lower-bound	511.000	1.000	511.000	26.813	.000
Error(Success_Rate_SRP2)	Sphericity Assumed	438.333	46	9.529		
	Greenhouse-Geisser	438.333	34.907	12.557		
	Huynh-Feldt	438.333	36.857	11.893		
	Lower-bound	438.333	23.000	19.058		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP2	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP2	Linear	507.000	1	507.000	35.444	.000
	Quadratic	4.000	1	4.000	.841	.368
Error(Success_Rate_SRP2)	Linear	329.000	23	14.304		
	Quadratic	109.333	23	4.754		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	4050.000	1	4050.000	37.672	.000
Error	2472.667	23	107.507		

RMA Analysis for Experiment 1 SRP 3

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP3	Dependent Variable
1	GEP
2	RGEP
3	ALGEP

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP3	Pillai's Trace	.750	32.987[b]	2.000	22.000	.000
	Wilks' Lambda	.250	32.987[b]	2.000	22.000	.000
	Hotelling's Trace	2.999	32.987[b]	2.000	22.000	.000
	Roy's Largest Root	2.999	32.987[b]	2.000	22.000	.000

a. Design: Intercept

Within Subjects Design: Success_Rate_SRP3

b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP3	.941	1.349	2	.510	.944	1.000	.500

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.

a. Design: Intercept

Within Subjects Design: Success_Rate_SRP3

b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP3	Sphericity Assumed	108.694	2	54.347	39.079	.000
	Greenhouse-Geisser	108.694	1.888	57.579	39.079	.000
	Huynh-Feldt	108.694	2.000	54.347	39.079	.000
	Lower-bound	108.694	1.000	108.694	39.079	.000
Error(Success_Rate_SRP3)	Sphericity Assumed	63.972	46	1.391		
	Greenhouse-Geisser	63.972	43.418	1.473		
	Huynh-Feldt	63.972	46.000	1.391		
	Lower-bound	63.972	23.000	2.781		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP3	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP3	Linear	99.188	1	99.188	68.482	.000
	Quadratic	9.507	1	9.507	7.132	.014
Error(Success_Rate_SRP3)	Linear	33.313	23	1.448		
	Quadratic	30.660	23	1.333		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	193.389	1	193.389	123.745	.000
Error	35.944	23	1.563		

RMA Analysis for Experiment 2 SRP 1

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP1	Dependent Variable
1	BASIC_ALGEP
2	LENGTH ADJUSTMENT
3	SLICE_XO
4	COMBINATION

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP1	Pillai's Trace	.900	62.916[b]	3.000	21.000	.000
	Wilks' Lambda	.100	62.916[b]	3.000	21.000	.000
	Hotelling's Trace	8.988	62.916[b]	3.000	21.000	.000
	Roy's Largest Root	8.988	62.916[b]	3.000	21.000	.000

a. Design: Intercept
Within Subjects Design: Success_Rate_SRP1
b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP1	.173	38.147	5	.000	.568	.609	.333

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.
a. Design: Intercept
Within Subjects Design: Success_Rate_SRP1
b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP1	Sphericity Assumed	16382.542	3	5460.847	144.730	.000
	Greenhouse-Geisser	16382.542	1.704	9612.668	144.730	.000
	Huynh-Feldt	16382.542	1.827	8968.044	144.730	.000
	Lower-bound	16382.542	1.000	16382.542	144.730	.000
Error(Success_Rate_SRP1)	Sphericity Assumed	2603.458	69	37.731		
	Greenhouse-Geisser	2603.458	39.198	66.418		
	Huynh-Feldt	2603.458	42.016	61.964		
	Lower-bound	2603.458	23.000	113.194		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP1	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP1	Linear	13188.033	1	13188.033	181.570	.000
	Quadratic	1.500	1	1.500	.091	.766
	Cubic	3193.008	1	3193.008	132.707	.000
Error(Success_Rate_SRP1)	Linear	1670.567	23	72.633		
	Quadratic	379.500	23	16.500		
	Cubic	553.392	23	24.061		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	664335.375	1	664335.375	6047.480	.000
Error	2526.625	23	109.853		

RMA Analysis for Experiment 2 SRP 2

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP2	Dependent Variable
1	BASIC_ALGEP
2	LENGTH ADJUSTMENT
3	SLICE_XO
4	COMBINATION

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP2	Pillai's Trace	.863	43.949[b]	3.000	21.000	.000
	Wilks' Lambda	.137	43.949[b]	3.000	21.000	.000
	Hotelling's Trace	6.278	43.949[b]	3.000	21.000	.000
	Roy's Largest Root	6.278	43.949[b]	3.000	21.000	.000

a. Design: Intercept
 Within Subjects Design: Success_Rate_SRP2
b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP2	.029	77.157	5	.000	.410	.422	.333

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.
a. Design: Intercept
 Within Subjects Design: Success_Rate_SRP2
b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP2	Sphericity Assumed	36295.208	3	12098.403	127.550	.000
	Greenhouse-Geisser	36295.208	1.231	29491.913	127.550	.000
	Huynh-Feldt	36295.208	1.265	28693.722	127.550	.000
	Lower-bound	36295.208	1.000	36295.208	127.550	.000
Error(Success_Rate_SRP2)	Sphericity Assumed	6544.792	69	94.852		
	Greenhouse-Geisser	6544.792	28.306	231.218		
	Huynh-Feldt	6544.792	29.093	224.960		
	Lower-bound	6544.792	23.000	284.556		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP2	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP2	Linear	29641.633	1	29641.633	138.036	.000
	Quadratic	8.167	1	8.167	.576	.455
	Cubic	6645.408	1	6645.408	119.410	.000
Error(Success_Rate_SRP2)	Linear	4938.967	23	214.738		
	Quadratic	325.833	23	14.167		
	Cubic	1279.992	23	55.652		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	90651.042	1	90651.042	111.969	.000
Error	18620.958	23	809.607		

RMA Analysis for Experiment 2 SRP 3

Within-Subjects Factors

Measure: MEASURE_1

Success_Rate_SRP3	Dependent Variable
1	BASIC_ALGEP
2	LENGTH ADJUSTMENT
3	SLICE_XO
4	COMBINATION

Multivariate Tests[a]

Effect		Value	F	Hypothesis df	Error df	Sig.
Success_Rate_SRP3	Pillai's Trace	.762	22.403[b]	3.000	21.000	.000
	Wilks' Lambda	.238	22.403[b]	3.000	21.000	.000
	Hotelling's Trace	3.200	22.403[b]	3.000	21.000	.000
	Roy's Largest Root	3.200	22.403[b]	3.000	21.000	.000

a. Design: Intercept
 Within Subjects Design: Success_Rate_SRP3

b. Exact statistic

Mauchly's Test of Sphericity[a]

Measure: MEASURE_1

Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Epsilon[b]		
					Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Success_Rate_SRP3	.814	4.471	5	.484	.881	1.000	.333

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.

a. Design: Intercept
 Within Subjects Design: Success_Rate_SRP3

b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP3	Sphericity Assumed	523.542	3	174.514	32.592	.000
	Greenhouse-Geisser	523.542	2.644	198.039	32.592	.000
	Huynh-Feldt	523.542	3.000	174.514	32.592	.000
	Lower-bound	523.542	1.000	523.542	32.592	.000
Error(Success_Rate_SRP3)	Sphericity Assumed	369.458	69	5.354		
	Greenhouse-Geisser	369.458	60.804	6.076		
	Huynh-Feldt	369.458	69.000	5.354		
	Lower-bound	369.458	23.000	16.063		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	Success_Rate_SRP3	Type III Sum of Squares	df	Mean Square	F	Sig.
Success_Rate_SRP3	Linear	448.533	1	448.533	61.972	.000
	Quadratic	6.000	1	6.000	1.551	.226
	Cubic	69.008	1	69.008	13.924	.001
Error(Success_Rate_SRP3)	Linear	166.467	23	7.238		
	Quadratic	89.000	23	3.870		
	Cubic	113.992	23	4.956		

Tests of Between-Subjects Effects

Measure: MEASURE_1

Transformed Variable: Average

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Intercept	3015.042	1	3015.042	298.959	.000
Error	231.958	23	10.085		

Attachment 4: Paired t-test

Experiment 1:

t-Test: Paired Two Sample for Means for SRP 1

	GEP	*RGEP*
Mean	31.33333333	49.95833333
Variance	185.9710145	317.432971
Observations	24	24
Pearson Correlation	0.889961225	
Hypothesized Mean Difference	0	
df	23	
t Stat	-10.83317215	
P(T<=t) one-tail	8.26989E-11	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.65398E-10	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	GEP	*ALGEP*
Mean	31.33333333	69.91666667
Variance	185.9710145	112.0797101
Observations	24	24
Pearson Correlation	0.619971987	
Hypothesized Mean Difference	0	
df	23	
t Stat	-17.32470229	
P(T<=t) one-tail	5.39303E-15	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.07861E-14	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	RGEP	*ALGEP*
Mean	49.95833333	69.91666667
Variance	317.432971	112.0797101
Observations	24	24
Pearson Correlation	0.747742966	
Hypothesized Mean Difference	0	
df	23	
t Stat	-8.052534309	
P(T<=t) one-tail	1.91644E-08	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	3.83288E-08	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	GEP	*RGEP*
Mean	4.083333	7.833333
Variance	11.73188	46.49275
Observations	24	24
Pearson Correlation	0.849528	
Hypothesized Mean Difference	0	
df	23	
t Stat	-4.2662	
P(T<=t) one-tail	0.000145	
t Critical one-tail	1.713872	
P(T<=t) two-tail	0.00029	
t Critical two-tail	2.068658	

t-Test: Paired Two Sample for Means SRP 2

	GEP	*ALGEP*
Mean	4.083333	10.58333
Variance	11.73188	68.34058
Observations	24	24
Pearson Correlation	0.908758	
Hypothesized Mean Difference	0	
df	23	
t Stat	-5.95347	
P(T<=t) one-tail	2.27E-06	
t Critical one-tail	1.713872	
P(T<=t) two-tail	4.54E-06	
t Critical two-tail	2.068658	

t-Test: Paired Two Sample for Means SRP 2

	RGEP	*ALGEP*
Mean	7.833333	10.58333
Variance	46.49275	68.34058
Observations	24	24
Pearson Correlation	0.92971	
Hypothesized Mean Difference	0	
df	23	
t Stat	-4.25566	
P(T<=t) one-tail	0.000149	
t Critical one-tail	1.713872	
P(T<=t) two-tail	0.000298	
t Critical two-tail	2.068658	

t-Test: Paired Two Sample for Means SRP 3

	GEP	*RGEP*
Mean	0.458333	1.125
Variance	0.346014	1.592391
Observations	24	24
Pearson Correlation	-0.13911	
Hypothesized Mean Difference	0	
df	23	
t Stat	-2.23002	
P(T<=t) one-tail	0.017897	
t Critical one-tail	1.713872	
P(T<=t) two-tail	0.035793	
t Critical two-tail	2.068658	

t-Test: Paired Two Sample for Means SRP 3

	GEP	*ALGEP*
Mean	0.458333	3.333333
Variance	0.346014	2.405797
Observations	24	24
Pearson Correlation	-0.07942	
Hypothesized Mean Difference	0	
df	23	
t Stat	-8.2754	
P(T<=t) one-tail	1.19E-08	
t Critical one-tail	1.713872	
P(T<=t) two-tail	2.39E-08	
t Critical two-tail	2.068658	

t-Test: Paired Two Sample for Means SRP 3

	RGEP	*ALGEP*
Mean	1.125	3.333333
Variance	1.592391	2.405797
Observations	24	24
Pearson Correlation	0.177708	
Hypothesized Mean Difference	0	
df	23	
t Stat	-5.95314	
P(T<=t) one-tail	2.27E-06	
t Critical one-tail	1.713872	
P(T<=t) two-tail	4.54E-06	
t Critical two-tail	2.068658	

Experiment 2:

t-Test: Paired Two Sample for Means SRP 1

	Basic ALGEP	*Length adjustment Opr.*
Mean	69.91666667	70.33333333
Variance	112.0797101	91.79710145
Observations	24	24
Pearson Correlation	0.7315481	
Hypothesized Mean Difference	0	
df	23	
t Stat	-0.27406989	
P(T<=t) one-tail	0.39323867	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.786477339	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	Basic ALGEP	*Slice crossover*
Mean	69.91666667	96.29166667
Variance	112.0797101	8.476449275
Observations	24	24
Pearson Correlation	0.177146953	
Hypothesized Mean Difference	0	
df	23	
t Stat	-12.34018744	
P(T<=t) one-tail	6.31284E-12	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.26257E-11	

t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	Basic ALGEP	*Combination*
Mean	69.91666667	96.20833333
Variance	112.0797101	10.69384058
Observations	24	24
Pearson Correlation	0.211508039	
Hypothesized Mean Difference	0	
df	23	
t Stat	-12.38663675	
P(T<=t) one-tail	5.85344E-12	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.17069E-11	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	Length adjustment Opr.	*Slice crossover*
Mean	70.33333333	96.29166667
Variance	91.79710145	8.476449275
Observations	24	24
Pearson Correlation	0.269128197	
Hypothesized Mean Difference	0	
df	23	
t Stat	-13.77248033	
P(T<=t) one-tail	6.74654E-13	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.34931E-12	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 1

	Length adjustment Opr.	*Combination*
Mean	70.33333333	96.20833333
Variance	91.79710145	10.69384058
Observations	24	24
Pearson Correlation	0.266898041	
Hypothesized Mean Difference	0	
df	23	
t Stat	-13.68761027	
P(T<=t) one-tail	7.66262E-13	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	1.53252E-12	

t Critical two-tail	2.06865761

t-Test: Paired Two Sample for Means SRP 1

	Slice crossover	*Combination*
Mean	96.29166667	96.20833333
Variance	8.476449275	10.69384058
Observations	24	24
Pearson Correlation	0.559605675	
Hypothesized Mean Difference	0	
df	23	
t Stat	0.139908785	
P(T<=t) one-tail	0.444975022	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.889950043	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Basic ALGEP	*Length adjustment Opr.*
Mean	10.58333333	12
Variance	68.34057971	74.86956522
Observations	24	24
Pearson Correlation	0.942740093	
Hypothesized Mean Difference	0	
df	23	
t Stat	-2.403118449	
P(T<=t) one-tail	0.012358538	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.024717076	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Basic ALGEP	*Slice crossover*
Mean	10.58333333	50.04166667
Variance	68.34057971	469.1721014
Observations	24	24
Pearson Correlation	0.724403694	
Hypothesized Mean Difference	0	
df	23	
t Stat	-11.59192235	
P(T<=t) one-tail	2.19731E-11	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	4.39462E-11	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Basic ALGEP	*Combination*
Mean	10.58333333	50.29166667
Variance	68.34057971	481.7807971
Observations	24	24
Pearson Correlation	0.716898357	
Hypothesized Mean Difference	0	
df	23	
t Stat	-11.42410525	
P(T<=t) one-tail	2.9299E-11	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	5.85979E-11	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Length adjustment Opr.	*Slice crossover*
Mean	12	50.04166667
Variance	74.86956522	469.1721014
Observations	24	24
Pearson Correlation	0.782241738	
Hypothesized Mean Difference	0	
df	23	
t Stat	-11.76739613	
P(T<=t) one-tail	1.63161E-11	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	3.26322E-11	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Length adjustment Opr.	*Combination*
Mean	12	50.29166667
Variance	74.86956522	481.7807971
Observations	24	24
Pearson Correlation	0.781781652	
Hypothesized Mean Difference	0	
df	23	
t Stat	-11.64069917	
P(T<=t) one-tail	2.02216E-11	

t Critical one-tail	1.713871528	
P(T<=t) two-tail	4.04431E-11	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 2

	Slice crossover	*Combination*
Mean	50.04166667	50.29166667
Variance	469.1721014	481.7807971
Observations	24	24
Pearson Correlation	0.946109845	
Hypothesized Mean Difference	0	
df	23	
t Stat	-0.170952931	
P(T<=t) one-tail	0.432877908	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.865755815	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Basic ALGEP	*Length adjustment Opr.*
Mean	3.333333333	3.25
Variance	2.405797101	4.282608696
Observations	24	24
Pearson Correlation	-0.13545295	
Hypothesized Mean Difference	0	
df	23	
t Stat	0.148498396	
P(T<=t) one-tail	0.441621868	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.883243736	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Basic ALGEP	*Slice crossover*
Mean	3.333333333	7.458333333
Variance	2.405797101	8.780797101
Observations	24	24
Pearson Correlation	0.343701112	
Hypothesized Mean Difference	0	
df	23	
t Stat	-7.13260605	
P(T<=t) one-tail	1.44872E-07	
t Critical one-tail	1.713871528	

P(T<=t) two-tail	2.89744E-07	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Basic ALGEP	*Combination*
Mean	3.333333333	8.375
Variance	2.405797101	10.67934783
Observations	24	24
Pearson Correlation	-0.042888445	
Hypothesized Mean Difference	0	
df	23	
t Stat	-6.717266326	
P(T<=t) one-tail	3.74448E-07	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	7.48896E-07	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Length adjustment Opr.	*Slice crossover*
Mean	3.25	7.458333333
Variance	4.282608696	8.780797101
Observations	24	24
Pearson Correlation	0.179024462	
Hypothesized Mean Difference	0	
df	23	
t Stat	-6.253826966	
P(T<=t) one-tail	1.10798E-06	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	2.21596E-06	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Length adjustment Opr.	*Combination*
Mean	3.25	8.375
Variance	4.282608696	10.67934783
Observations	24	24
Pearson Correlation	0.075541128	
Hypothesized Mean Difference	0	
df	23	
t Stat	-6.724573075	
P(T<=t) one-tail	3.68175E-07	

t Critical one-tail	1.713871528	
P(T<=t) two-tail	7.3635E-07	
t Critical two-tail	2.06865761	

t-Test: Paired Two Sample for Means SRP 3

	Slice crossover	*Combination*
Mean	7.458333333	8.375
Variance	8.780797101	10.67934783
Observations	24	24
Pearson Correlation	0.470874098	
Hypothesized Mean Difference	0	
df	23	
t Stat	-1.396511822	
P(T<=t) one-tail	0.087946357	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.175892714	
t Critical two-tail	2.06865761	