T.R.
ONDOKUZ MAYIS UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF COMPUTER ENGINEERING

SOLVING AUTOMATIC PROGRAMMING PROBLEMS WITH FIREFLY PROGRAMMING METHOD

Master's Thesis

Mohamed ALIWI

Supervisor

Asst. Prof. Dr. Sercan DEMİRCİ

II. Supervisor

Assoc. Prof. Dr. Selçuk ASLAN

SAMSUN
2022

ACCEPTANCE AND APPROVAL OF THE THESIS

The study entitled **"SOLVING AUTOMATIC PROGRAMMING PROBLEMS WITH FIREFLY PROGRAMMING METHOD"** prepared by **Mohamed ALIWI** and supervised by **Asst. Prof. Dr. Sercan DEMİRCİ** and **Assoc. Prof. Dr. Selçuk ASLAN** was found successful and unanimously accepted by committee members as master thesis of the Department of Computer Engineering, following the examination on the date 29/08/2022.

	Title Name SURNAME **University** **Department/Art**	**Signature**	**Final Decision**
Chairman	Assoc. Prof. Dr. Geylani KARDAŞ Ege University Department of Information Technologies		☒ Accept ☐ Reject
Member	Assoc. Prof. Dr. Sedat AKLEYLEK Ondokuz Mayıs University Department of Computer Engineering		☒ Accept ☐ Reject
Member	Asst. Prof. Dr. Sercan DEMİRCİ Ondokuz Mayıs University Department of Computer Engineering		☒ Accept ☐ Reject

This thesis has been approved by the committee members that already stated above and determined by the Institute Executive Board.

APPROVAL
..../..../2022
Prof. Dr. Ali BOLAT
Head of Institute of Graduate Studies

DECLARATION OF COMPLIANCE WITH SCIENTIFIC ETHIC

I hereby declare and undertake that I complied with scientific ethics and academic rules in all stages of my M.S. thesis, that I have referred to each quotation that I used directly or indirectly in the study, and that the works I have used consist of those shown in the references, that it was written in accordance with the institute writing guide and that the stated cases in article 3, section 9 of the Regulation for TÜBİTAK Research and Publication Ethics Board were not violated.

Is an Ethics Committee Necessary?

☐ Yes

☒ No

29/08/2022
Mohamed ALIWI

DECLARATION OF THE THESIS STUDY ORIGINALITY REPORT

Thesis Title: SOLVING AUTOMATIC PROGRAMMING PROBLEMS WITH FIREFLY PROGRAMMING METHOD

As a result of the originality report taken by me from the plagiarism detection program on 23/06/2022 for the thesis titled above:

Similarity ratio : %10

Single source rate : %1 has been released.

29/08/2022
Asst. Prof. Dr. Sercan DEMİRCİ

ÖZET

ATEŞBÖCEĞİ PROGRAMLAMA YÖNTEMİ İLE OTOMATİK PROGRAMLAMA PROBLEMLERİNİN ÇÖZÜLMESİ

Mohamed ALIWI
Ondokuz Mayıs Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Ana Bilim Dalı
Yüksek Lisans, Ağustos/2022
Danışman: Dr. Öğr. Üyesi Sercan DEMİRCİ
II. Danışman: Doç. Dr. Selçuk ASLAN

Klasik regresyon yöntemleri, verimlilik ve doğru çözümler bulma yetenekleri nedeniyle farklı problemlerin çözümünde yaygın olarak kullanılmaktadır. Klasik yöntemlerin çoğu özelleşmiş olarak kabul edilmesine rağmen, farklı problemleri çözmelerini engelleyen bazı kısıtlamalara sahiptir. Otomatik programlama veya sembolik regresyon yöntemleri, önceden belirlenmiş herhangi bir desen veya yapı kullanmadan karmaşık veriye dayalı problemleri çözebilen makine öğrenme teknikleridir. Otomatik programlama yöntemleri, optimal modeli oluşturmak için farklı matematiksel ifadeleri ve sabitleri birleştirir.

Bu tez, sembolik regresyon prensiplerini kullanarak karmaşık problemleri çözebilen yeni bir otomatik programlama yöntemi sunmayı amaçlamaktadır. Bu tez kapsamında, sembolik regresyon problemlerini çözebilmek için ateşböceği algoritmasını modifiye eden ilk otomatik programlama yöntemi olarak ateşböceği programlama önerilmiştir. Ateşböceği programlama süreçlerinin iyileştirilmesi sonucunda, farka dayalı ateşböceği programlama adlı geliştirilmiş bir versiyon da önerilmiştir.

Yeni önerilen yöntemler, performanslarını iki farklı problem kullanarak test etmeden önce ayrıntılı olarak açıklanmıştır. Önerilen ateşböceği programlama yöntemleri, farklı sembolik regresyon kıyaslama problemlerinin çözümlemesinde ve Box-Jenkins zaman serilerinin modellemesinde kullanılmıştır. Deneysel testlerin sonuçları, yeni önerilen yöntemlerin veriye dayalı farklı problemleri ne kadar iyi çözebileceğini göstermektedir.

Anahtar Sözcükler: Sembolik Regresyon, Otomatik Programlama, Ateşböceği Algoritması, Ateşböceği Programlama, Makine Öğrenme

ABSTRACT

SOLVING AUTOMATIC PROGRAMMING PROBLEMS WITH FIREFLY PROGRAMMING METHOD

Mohamed ALIWI
Ondokuz Mayıs University
Institute of Graduate Studies
Department of Computer Engineering
Master, August/2022
Supervisor: Asst. Prof. Dr. Sercan DEMİRCİ
II. Supervisor: Assoc. Prof. Dr. Selçuk ASLAN

Classical regression methods are used widely to solve different problems due to their efficiency and ability to find accurate solutions. Although most of the classical methods are considered specialized, they have some limitations that prevent them from solving various problems. Automatic programming or symbolic regression methods are machine learning techniques that can solve complex data-driven problems without using any pre-defined pattern or structure. Automatic programming methods are usually based on evolutionary computation algorithms that search among several candidate solutions to find the best one. Automatic programming methods combine different mathematical expressions and constants to form the optimal model.

This thesis aims to present a new automatic programming method that can solve complex problems using the principles of symbolic regression. In the scope of this thesis, firefly programming is proposed as the first automatic programming method that modifies the firefly algorithm to be able to solve symbolic regression problems. As a result of improving the processes of firefly programming, an improved version called difference-based firefly programming is proposed also.

The newly proposed methods are described in detail before testing their performance using two different problems. The proposed firefly programming methods were used in solving different symbolic regression benchmark problems, and modeling Box-Jenkins time series. The results of the experimental tests show how well the newly proposed methods can solve different data-driven problems.

Keywords: Symbolic Regression, Automatic Programming, Firefly Algorithm, Firefly Programming, Machine Learning

ACKNOWLEDGEMENT

First of all, I would like to say “Alhamdulillah”, Almighty God for his blessings and mercy that guided and helped me to finish this work. Following the Sunnah of our Prophet Muhammad, peace and blessings be upon him, who said: “He who does not thank the people is not thankful to Allah”, I would like to thank my family, my beloved mother, father, brothers, and sisters who helped and supported me, not only in this work but in all my life. I would like to thank my supervisors Asst. Prof. Dr. Sercan DEMİRCİ, and Assoc. Prof. Dr. Selçuk ASLAN for their encouragement, advice, and support throughout this work. Thanks to all my friends who also supported me to finish this work. Thanks to everyone who helped me directly or indirectly to complete this thesis.

Mohamed ALIWI

CONTENTS

SYMBOLS AND ABBREVIATIONS

ABCP	: Artificial Bee Colony Programming
ACO	: Ant Colony Optimization
AP	: Ant Programming
AI	: Artificial Intelligence
AIS	: Artificial Immune System
CGP	: Cartesian Genetic Programming
CSA	: Clonal Selection Algorithm
DE	: Differential Evolution
DFP	: Distance-based Firefly Programming
EC	: Evolutionary Computation
FP	: Firefly Programming
GA	: Genetic Algorithm
GEP	: Gene Expression Programming
GE	: Grammatical Evolution
GP	: Genetic Programming
IP	: Immune Programming
IPA	: Immune Plasma Algorithm
IPP	: Immune Plasma Programming
KNN	: K-Nearest Neighbour
LGP	: Linear Genetic Programming
MEP	: Multi Expression Programming
MHABCP	: Multi Hive Artificial Bee Colony Programming
ML	: Machine Learning
MSE	: Mean Square Error
NN	: Neural Network
QABCP	: Quick Artificial Bee Colony Programming
QSABCP	: Quick Semantic Artificial Bee Colony Programming
SA	: Simulated Annealing
SABCP	: Semantic Artificial Bee Colony Programming
SGP	: Stack-based Genetic Programming
SIHC	: Stochastic Iterated Hill Climbing
SVM	: Support Vector Machine

FIGURES LEGENDS

TABLES LEGENDS

1. INTRODUCTION

The role of technology in human daily lives increased during the last two decades significantly. Data is obtained from almost everything now. The increase in data collecting raised the need to develop different techniques that can handle it. The interest in artificial intelligence (AI) increased dramatically because of its ability to handle and solve problems efficiently. AI techniques are used widely for different purposes; data analysis, problem-solving, forecasting and modeling, and others (Goodfellow et al., 2016). Developing new AI and machine learning (ML) methods is essential to maximize the benefit of the obtained data as much as possible.

Data varies between structured and unstructured types (Eryurek et al., 2021). Structured data has a certain pattern in contrast with unstructured data. The type of data impacts the success of the used method. Methods manipulate data in different ways according to its structure in most cases (Goodfellow et al., 2016; Alpaydin, 2020). Performing and analyzing data is an important process to determine the most suitable technique to solve the problem (Goodfellow et al., 2016; Alpaydin, 2020). Data analysts confirm the major role of the analysis in obtaining good results. For instance, classification task methods are incompatible to be used for dimensionality reduction problems. Therefore, it is essential to take the previous points into consideration to ensure the success of the chosen method and to obtain desired results.

Regression analysis is a machine learning method concerned with analyzing data and making predictions. Most classical regression methods search for the relation between dataset variables using a specific structure. The pre-specified model limits the classical regression methods and prevents them from solving various problems since not all data has a recognizable pattern (Rusell & Norvig, 2003; Mundhenk et al., 2021). This issue motivated researchers to develop new methods that can overcome this limitation. Developers succeeded in developing methods with the ability to detect the pattern of the data without relying on any pre-specified model. Methods with pattern-detection abilities are called later symbolic regression or automatic programming methods (Mundhenk et al., 2021).

Most symbolic regression methods are developed using different evolutionary algorithms that use evolutional principles to improve different solutions through consecutive generations. Several algorithms used this mechanism to solve problems. The main issues of evolutionary algorithms are unreliability and randomness which are a result of the different basis that have been used in each algorithm. Thus, each one has strengths and weak points different than the others. Therefore, it is necessary to develop different automatic programming methods based on several evolutionary algorithms to overcome such problems. Various automatic programming methods were presented and used to solve different problems including classification (Loveard & Ciesielski, 2001; Espejo et al., 2009; Santoso et al., 2020), clustering (Dimopoulos & Mort, 2001; De Falco et al., 2006), feature selection (Muni et al., 2006; Arslan & Ozturk, 2019b), modeling and planning (Contreras-Cruz et al., 2015), predictions (Kaboudan, 2000; Fallah-Mehdipour et al., 2013), and more. Within the scope of this thesis, "automatic programming methods" and "symbolic regression methods" terms are used interchangeably since they both refer to the same meaning.

1.1. Research Objectives

This research aims to propose a new automatic programming method as a new machine learning technique that can detect the pattern of different types of data, and solve complex problems which are not solvable using the traditional methods. This work introduces firefly programming (FP) as an automatic programming method that extends the firefly algorithm and improvs it by using the principles of symbolic regression. Firefly programming is the first method to use the firefly algorithm as a basis to solve problems by detecting and recognizing the pattern of the data (Aliwi et al., 2020a). FP offers a new choice to solve complex problems efficiently. Within the scope of this thesis, firefly programming is proposed and described in detail. FP uses different functional and terminal searching space sets to create the structure of the problem solution. FP uses different metrics as objective functions to evaluate and improve the accuracy of the generated structure. In order to increase the efficiency of the newly introduced method, a new improved version of FP is proposed and discussed by explaining the difference between its operations and the operations of FP.

1.2. Thesis Structure

After excluding this introduction, the rest of the thesis consists of four other chapters. The second chapter "literature review" starts by giving an overview of machine learning by explaining the most important concepts related to it, and the main tasks achieved via machine learning methods. It also describes the learning process starting by gathering data and ending by exporting results of the different learning scenarios. Classical regression methods are briefly explained by listing the cases in which are used. Symbolic regression is also described by explaining the difference between it and classical regression. At the end of this chapter, the history of automatic programming methods is reviewed by mentioning and discussing the related works.

The third chapter: "materials and methods" starts by describing the firefly algorithm which is used later as a basis for developing both firefly programming (FP) and difference-based firefly programming (DFP) methods. It starts by describing the behavior of the fireflies in nature that inspired Yang to develop the firefly algorithm (Yang, 2008, 2009, 2010, 2017). It enumerates the characteristic properties of the firefly algorithm and clarifies its concepts. It also discusses the efficiency of the firefly algorithm and how it can deal with local traps and avoids them. Within the scope of this chapter, the newly introduced FP method is described by explaining the concepts of forming solution trees and the sharing operator. It also presents the improved version of the firefly programming method by explaining the most important differences between it and the standard firefly programming method.

In the fourth chapter, two different experiments are performed to evaluate the performance of FP and DFP. The first experiment used a set of symbolic regression benchmark equations to be modeled using FP and DFP. The results of FP and DFP are compared to the results of the genetic programming (GP) (Uy et al., 2013) and the artificial bee colony programming (ABCP) (Gorkemli & Karaboga, 2015) methods. A comparison between FP and DFP was performed also using different population size values to evaluate the efficiency of each method. In the second experiment, both FP and DFP were used in modeling the famous Box-Jenkins gas furnace time series (Box & Jenkins, 1976; Box et al., 2015). This experiment was performed using four different cases using different functional sets and depth limitation values. The results of

this experiment were compared to the results of different product-unit neural networks (Öztürk, 2011).

The last chapter discusses the results of FP and DFP and evaluates their performance in the pre-mentioned experiments. It investigates the strengths and weaknesses of FP and DFP and discusses the reasons behind each point. It gives the conclusion about this work and suggests recommendations to improve the proposed methods in future works.

2. LITERATURE REVIEW

Automatic programming techniques use the principles of symbolic regression to solve problems. Symbolic regression is a type of regression analysis that is used for finding the relation between problem's data records. In general, regression analysis is a part of machine learning tasks. Therefore, it is necessary to give a brief overview of machine learning by explaining its basic concepts before discussing the main tasks of machine learning methods. This section also explains the difference between classical and symbolic regression before giving an overview of the history of automatic programming methods.

2.1. Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that uses several techniques to learn from data. Machine learning techniques are used to solve various types of problems starting by analyzing data before performing different processes to give optimal solutions (Burkov, 2019; Mitchell, 1997). Mohri et al. in his book "Foundations of Machine Learning", stated that ML techniques are computational methods that can learn by repeating to gain experience in solving problems automatically (Mohri et al., 2018). This process is similar to the person that gets experience after performing a specific task frequently. Since ML methods use data to solve problems, they share some principles with statistical methods; therefore, ML methods are used for data analysis, optimization problems, and other data-driven problems (Mohri et al., 2018). It is essential to choose an appropriate algorithm to solve a specific problem to obtain good results. The efficiency of the chosen algorithm is determined by measuring the correctness and accuracy of the solutions at the end of the optimization process. Quality measurements such as space and time complexity must be considered before using any algorithm since these measurements play an important role in determining the overall efficiency (Mohri et al., 2018).

2.1.1. Machine Learning Tasks

Nowadays, machine learning methods are used to solve most data-driven problems. Defining the task of machine learning methods is required to get the expected results. The purpose of the learning process is determined by the type of data, and the expected goals. According to (Mohri et al., 2018; Alpaydin, 2020), ML tasks are classified into several main tasks:

- *Classification*: A process of categorization different data and determining the class that belongs. Data is classified into two or more categories according to the purpose of the classification process (Mohri et al., 2018). For instance, the process of evaluating students' academic status at the end of the year is a classification process since it has only two possible results, pass or fail.
- *Clustering*: A process that is almost similar to classification. In the clustering task, data is divided into an unspecified number of groups, where the algorithm searches for the common attributes between the data values and regroups them (Mohri et al., 2018). The difference between clustering and classification processes is that the number of classes is prespecified initially in the classification process. Grouping users according to their interests in social networks is an example of the clustering process.
- *Regression*: A process of modeling a system that predicts results depending on input values. Some of the regression types are used for classifying data where it generates a continuous model, in contrast to classification that generates a discrete model (Rusell & Norvig, 2003; Alpaydin, 2020). More details about regression methods are given within the scope of this chapter.
- *Association Rules*: A process that is concerned with finding the patterns that link data values to each other. It shows the probability $P(B|A)$ of occurring event B considering event A (Agrawal et al., 1993). Recommendation system in video streaming platforms uses association rules to predict user's watching lists.
- *Dimension Reduction*: A technique that is used for reducing the number of the problem features or input variables (Mohri et al., 2018). It helps to reduce the complexity of the problems and increases the performance of the used method.

2.1.2. Learning Process

The learning process starts by defining the dataset that will be used and analyzed. Usually, the dataset contains different features which represent data inputs. The model that is generated by the ML method depends on the problem features, thus, deciding which features are used in the learning process is important. Feature selection methods aim to reduce the number of features to the least possible limit without affecting the efficiency of the model (Kuhn et al., 2013). After analyzing data and selecting features, the dataset is split into training and testing sets. The selected method uses the training set to discover the model and optimize its parameters if any exist. On the other hand, the testing set is used for evaluating the generated model (Mitchell, 1997). In most cases, the ratio of the training set to the testing one ranges between 6:4 and 8:2. At the end of the evaluation process, the deployment process allows the model to be used later. Figure 2.1 shows the flowchart of the learning process.

The learning process scenario differs according to the type of data. Data records may either be labeled, unlabeled, or even a combination of them. The next example describes label data types. Think about a data table that describes different animals, Features are the columns that have the properties of each animal like the number of legs, food type, and others. Labels such as animal names depend directly on other features (Serrano, 2021). Back to learning scenarios, there are many scenarios according to the type of data. In most cases, there are three common types (Rusell & Norvig, 2003; Goodfellow et al., 2016):

- *Supervised Learning*: Data training set contains labeled data that can help the algorithm to generate better predictions. Classification and regression methods including K-Nearest Neighbour (KNN), Support Vector Machine (SVM), and Logistic Regression are considered supervised learning algorithms.
- *Unsupervised Learning*: Unlabeled datasets are used in this type of learning method. Since data is not labeled, the learning process usually will be more difficult than it is in supervised learning methods. K-Means algorithm which is used as a clustering method is an example of unsupervised learning techniques. The same is true for dimension reduction algorithms such as the popular matrix factorization methods. Association rule mining methods are also considered unsupervised.

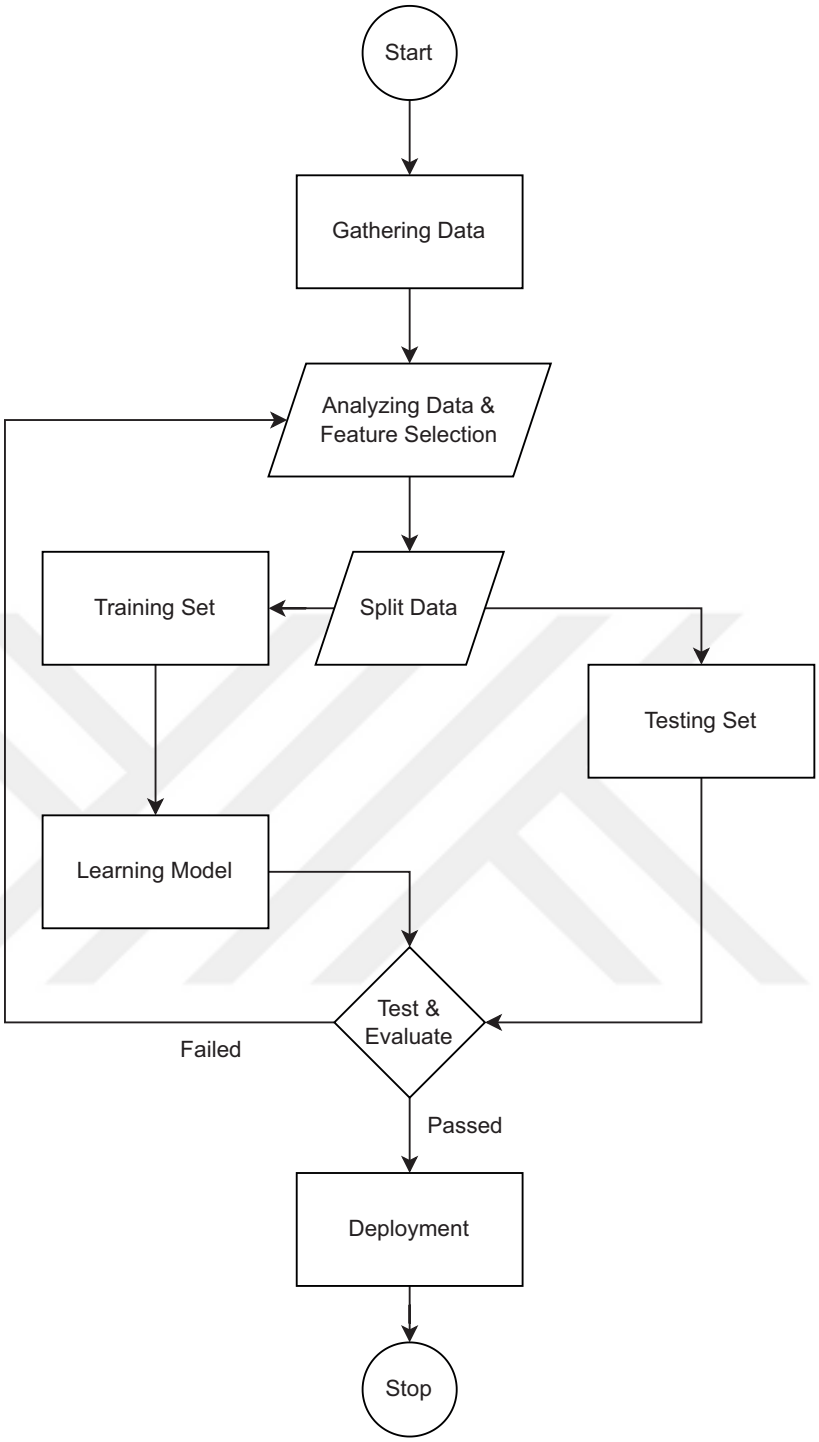


Figure 2.1. Flowchart of the learning process

- *Reinforcement Learning*: In this scenario, the learning method which generally uses unlabeled data gets either rewards or punishments to optimize the learning process. The Q-learning algorithm is an example of this learning scenario.

2.2. Classical Regression

Regression analysis concerns finding the mathematical form of a specific system that can predict the dependent variables (outputs) after analyzing the independent variable set. In other words, regression analysis searches for the relationship between the independent inputs that give a particular output through a specific system (Rusell & Norvig, 2003; Alpaydin, 2020). Regression analysis methods are mostly used in modeling and prediction problems. The general form for different classical regression methods is shown in Eq. 2.1 where Y represents the dependent variable (predicted), X is the vector of independent variables, W is the X's weight vector, and ϵ represents the error term.

$$Y = f(X, W) + \epsilon \tag{2.1}$$

It is worth mentioning that the predicted output Y can be a vector of multiple dependent variables as we can see in Figure 2.2. The generated model is represented with $f(X, W)$ and it is different for each regression method.

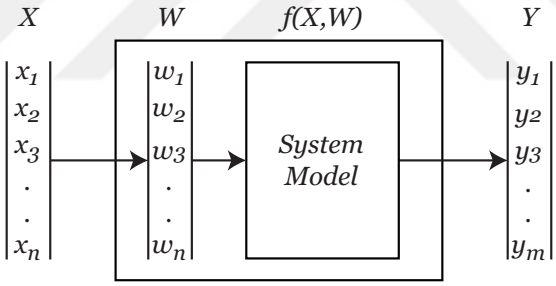


Figure 2.2. System model with n input variables and m output variables

Regression analysis methods have different prespecified expression forms, which make each method specialized in modeling specific types of problems (Mohri et al., 2018). The distribution of the data plays an important role in selecting the most suitable method. The following list contains the most common regression methods that are used to solve problems:

- *Linear Regression*: The most used regression analysis method duo to its efficiency in predicting linear models (Burkov, 2019). Linear regression has a prespecified model shown in Eq. 2.2:

$$f(X, W) = x_1\omega_1 + x_2\omega_2 + x_3\omega_3 + \cdots + x_n\omega_n + b \tag{2.2}$$

where x_n is the n^{th} input variable, ω_n is the weight of the n^{th} input variable, and b is the bias value. Figure 2.3 (a) shows an example graph of a linear regression generated model.

- *Logistic Regression*: This type is efficient in modeling two-values discrete datasets. Logistic regression is classified as a binary classification method since it's used only for solving two class-based problems (Burkov, 2019). The basic model has the same structure as the linear regression model, but the difference lies in using the sigmoid function to generate the final model with a curve that is similar to the one in Figure 2.3 (b). The mathematical expression form of this method is (Eq. 2.3):

$$\begin{aligned} z(X,W) &= x_1\omega_1 + x_2\omega_2 + x_3\omega_3 + \cdots + x_n\omega_n + b \\ f(X,W) &= sigmoid(z) = \frac{1}{(1+e^{-z})} \end{aligned} \tag{2.3}$$

- *Polynomial Regression*: In this method, the generated model's curve is not linear because of the non-linearity distribution of the data. This type of regression is used for both linear and non-linear datasets due to its ability to optimize inputs' weights to fit data values. Although this can be considered an advantage, this ability may leads to an overfitting learning case, therefore, it is not used for solving linear regression problems. Figure 2.3 (c) shows a polynomial regression model example which has an expression form as in Eq. 2.4:

$$f(X,W) = x_1\omega_1 + x_2\omega_2^2 + x_2\omega_3^3 + \cdots + x_n\omega_n^n + b \tag{2.4}$$

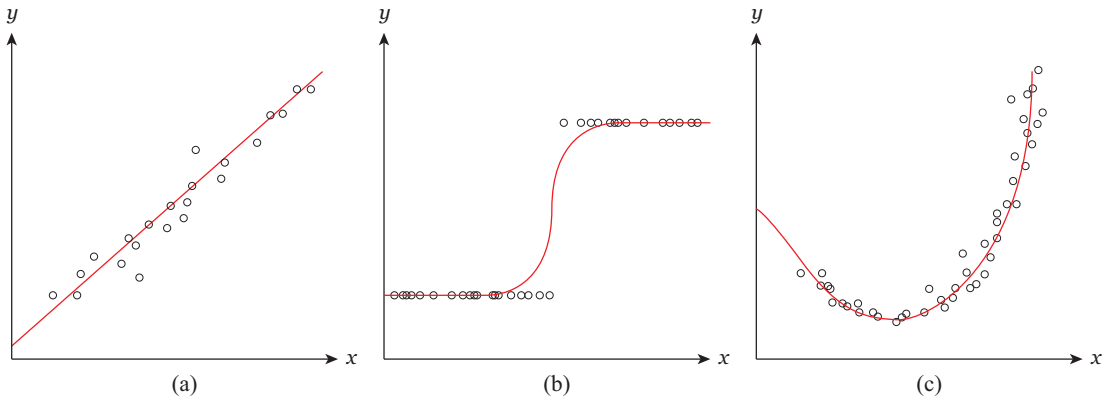


Figure 2.3. Examples of different regression analysis models' graphs: (a) Linear regression - (b) Logistic regression - (c) Polynomial regression

Alongside the previously listed regression methods, it's worthy to mention some other common methods such Ridge, Lasso, and ElasticNet regression that are used for solving complex regression problems (Mohri et al., 2018; Tibshirani, 1996; Ogutu et al., 2012).

2.3. Symbolic Regression

Symbolic regression is a method that is concerned with identifying the mathematical form that generates output variables using a set of independent inputs through a particular system (Mundhenk et al., 2021). The process starts by searching in the space of the different functional expression spaces and combining them to form a mathematical model. The previous Section 2.2. mentioned that different regression methods have a predefined form that is used as an initial form to be optimized. Therefore, data analysis process is important since it helps detect the general pattern of the data which in turn leads to obtaining good results (Shamoo & Resnik, 2009).

2.3.1. Difference Between Classical & Symbolic Regression Methods

In real-world problems, not all have a pattern that makes the problem predictable. the usage of the classical regression methods may not be efficient enough to give good results for different data-driven problems. In other words, classical regression methods are not capable to solve problems that do not have a well-known data pattern, or pattern is too complex. Symbolic regression is suitable for such problems since it does not have a predefined model structure, both structure and parameters are unknown. The methods that use the principles of symbolic regression try to recognize the pattern of the data in order to form a suitable structure and optimize its parameters at the same time (Billard & Diday, 2002). Symbolic regression methods use different mathematical expressions and combine them to form the structure of the solution (Schmidt & Lipson, 2006). The used expressions include simple mathematical operators (addition, subtraction, multiplication, and division), trigonometric functions (sine, cosine, tangent, . . . , etc.), logical (and, or, not, nor, if-else, . . . , etc.), and others (Koza, 1992, 1994). Besides the functional expressions, it uses also different variables and constants as terminals. While terminals represent coefficients (in case the terminal is a constant) or one of the problem features (terminal is a variable), the functional expressions describe the

relationship between the connected functional nodes and terminals. The combination of the functional and terminal sets (Figure 2.4) creates the searching space which is defined by analysts.

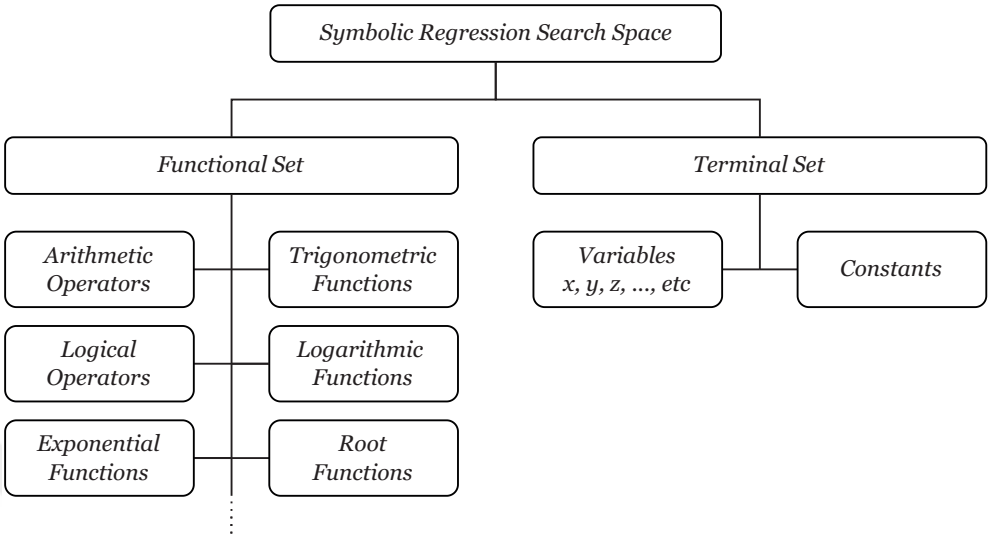


Figure 2.4. Functional and terminal sets in symbolic regression

2.3.2. Advantages & Disadvantages of Symbolic Regression

Smits and Kotanchek in their work discussed the benefits of symbolic regression by mentioning its contribution as a machine learning method. They also listed the motivations that push analysts and researchers to use and develop symbolic regression methods (Smits & Kotanchek, 2005). Vladislavleva et al. in their paper discussed the advantages of symbolic regression methods over other non-linear models such neural networks (Vladislavleva et al., 2008). The following points summarize the advantages of using symbolic regression methods:

- *Simple Expressions*: Most automatic programming methods are developed using different swarm-based methods, hence, the complexity of the generated expressions varies between complex and simple solutions.
- *Fewer Limitations*: Unlike classical methods which have predefined structures, symbolic regression does not have such limitations since it searches for both structure and its parameters by evaluating the solutions before exporting the best one.
- *Human-expected Results*: The generated expressions reflect the expectations of the

analysts because the model uses a combination of the pre-defined searching space set. This point makes generated model predictable and trustworthy.

- *Low-cost Preprocessing*: Symbolic regression methods generate different expressions without using any feature-selection methods since they are based on evolutionary algorithms which can detect the more important features and ignore the rest.
- *Diversity*: Symbolic regression methods generate different candidate solutions evolved through the evaluation process. Each one of these solutions has different properties.

Smits and Kotanchek also discussed the disadvantages of symbolic regression methods and the difficulties that face the users of them. The disadvantages are listed below:

- *Increased Time-execution*: The most important disadvantage is the slowness in exporting solutions compared to the classical methods. The main reason behind this is that symbolic regression does not have a predefined structure and it has to recognize the pattern of the given data. This process will increase the total execution time and show symbolic regression methods as slow techniques.
- *Difficulties in Exporting Best Solutions*: Since symbolic regression evaluates the solutions using a proper objective function, the selected best solution is not always the desired one because most evaluations do not consider the complexity of the generated solutions.
- *Unspecialized*: The results of using symbolic regression compared to other methods are not always the best since most machine learning methods are developed as specialized techniques for solving specific problems.

2.3.3. Genetic Programming (GP)

Symbolic regression methods try to find the fittest model of the problem by transforming it into an optimization problem that can be solved with evolutionary computation (EC) techniques such as the popular genetic algorithm (GA) (Koza, 1992; Langdon & Poli, 2002; Poli et al., 2008). GA was the basis for developing genetic programming (GP), the first automatic programming method that can solve problems using the principles of symbolic regression (Koza, 1992, 1994; Koza et al., 1999). In

GP, each individual is a program that represents a solution for the problem. Koza used a hierarchical parse-tree structure that consists of connected functional expressions and terminals to represent the individuals of the GP as in Figure 2.5.

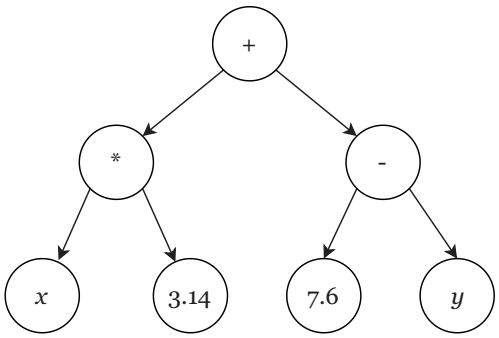


Figure 2.5. Individual representation in GP

Parse-tree structure is implemented easily as computer programs using the syntax of LISP which is also known as symbolic expression or S-expression. For instance, the computer program visualized in Figure 2.5 is represented using S-expression as in Eq. 2.5:

$$+(*x3.14)(-7.6y) \tag{2.5}$$

GP uses crossover, mutation, and reproduction operators to find the best program that describes the relationship between the problem's variables (Koza, 1992; Johnson, 2002; Uy et al., 2010). The crossover operator in GP is done by taking two subtrees of the hierarchically structured parent programs to create the new offspring (see Figure 2.6). In mutation, a randomly selected subtree is replaced with another formed one as Figure 2.7 shows. The reproduction operator copies a current-generation program into the next generation. Programs are evaluated using a suitable objective function to determine their fitnesses. At the end of the evaluation process, GP exports the best program among all candidate programs as a symbolic expression (Koza, 1992, 1994; Koza et al., 1999). This made GP able to solve problems without the need for any predefined structure, which gave it an advantage over other machine learning methods.

Koza used his model (Koza, 1994) in simulating an 11-input multiplexer (11-MX) where the searching space set included (and, or, not, if) expressions as functions, in addition to the 11-inputs as terminals. GP successfully simulated the multiplexer and

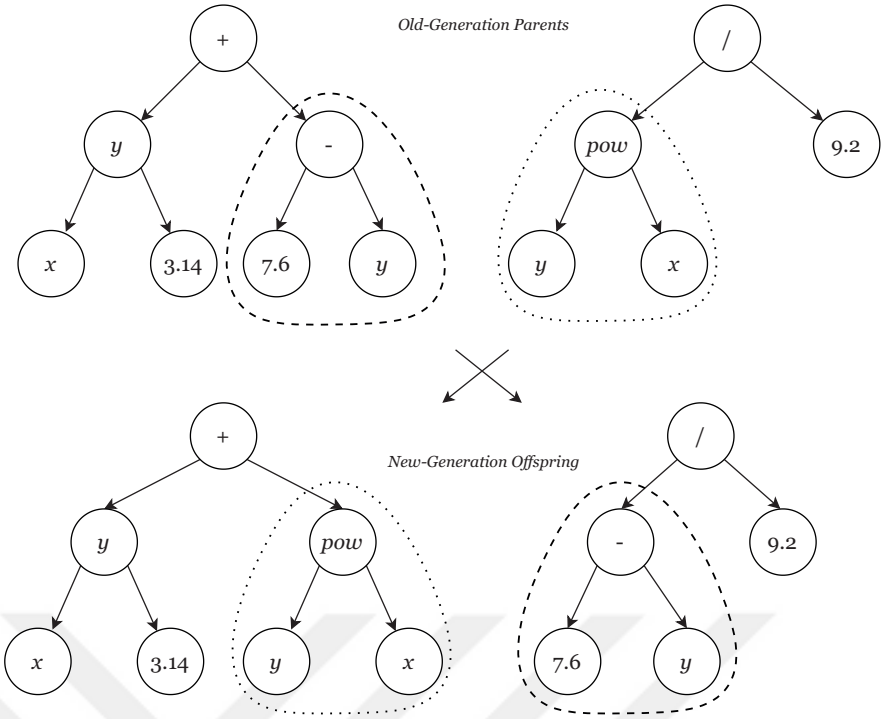


Figure 2.6. Crossver example in GP

generated a symbolic model that is equivalent to it. Koza also used its method to model empirical data as real-world data values, which resulted in excellent results.

Solving problems using symbolic regression methods started after Koza introduced his first GP model. As a result of being the first automatic programming technique, and due to its capability to solve different problems in different working fields, GP attracted researchers to inspect and work on improving it. The development process did not focus on developing the GP syntactically only, semantically developments were also done (Uy et al., 2010). Some imporovments focused on the basic GP operators such as crossover and mutation, whereas the others were working on developing the representation structure of the individuals.

2.3.3.1. Development of the Basic Operators

Koza in his initial model proposed a crossover operator with a probability of 90% to choose a functional node as a subtree root. The proposed crossover operator in GP produces new solutions easily, but also complex ones (Koza, 1992; Beadle & Johnson, 2008). On the other hand, the initial mutation operator was as simple as described in the previous paragraph. Researchers started to develop new operator variants to increase

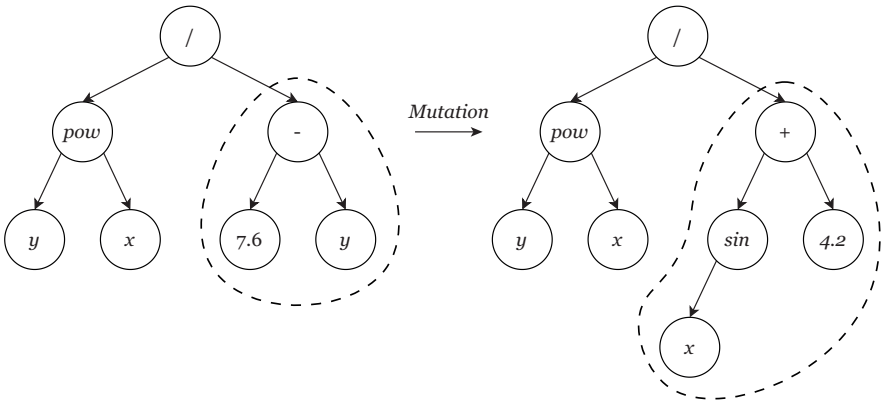


Figure 2.7. Mutation example in GP

the efficiency and performance of GP. O'Reilly and Oppacher proposed performing a height-fair crossover process that takes into consideration the height of the subtrees compared to their parent trees. O'Reilly and Oppacher implemented a new crossover operator and performed three experimental tests by simulating the 11-MX, 6-MX, and sorting problems. The results of GP with the modified crossover operators were compared to the results of simulated annealing (SA), standard genetic programming (GP), and stochastic iterated hill climbing (SIHC) algorithms. The results showed that GP with the modified crossover operator gave better results than standard GP. However, GP was not as successful as SA which was the best in all experiments. On the other side, GP gave better results than the SIHC for all experimental test cases (O'Reilly & Oppacher, 1994).

Harries et al. investigated the performance of seven different GP-crossover operators (GP-Standard, GP-NoBias, GP-SameDepths, GP-DiffDepths, GP-Std/Same, GP-Std/Diff, and GP-Half&Half) using four different experimental problems: the problem of the artificial ant, simple polynomial problem, boolean 4-PARITY, and 5-PARITY problems. The results showed that the standard same depth-based (GP-Std/Same) crossover operator gave the best performance compared to the others (Harries et al., 1997).

Ito et al. proposed two depth-dependent crossover operators which choose a node with the same depth value as the other selected one. Ito et al. performed several experiments to evaluate the performance of the new operators, in the first and second experiments (11-MX and 4-PARITY problem), both depth-dependent (DD) and revised depth-dependent (RDD) crossover operators outperformed the standard

crossover operator. Ito et al. used DD and RDD models to solve the artificial ANT problem where the performance of both DD and RDD was not as predicted and very similar to the results of the standard GP crossover operator (Ito et al., 1998).

In 1998, Poli and Langdon introduced two new crossover operators: one-point, and strict one-point crossover. These two operators inspect the shape of the parents' subtrees, then, find subtrees with the same arity before swapping them to generate the new offspring. Poli and Langdon investigated the behaviors of these operators using three problems: 3-PARITY, 4-PARITY, and 5-PARITY. The results showed that it will be good only if the initial tree's depth was selected correctly (Poli & Langdon, 1998; Langdon, 2000).

The research on GP basic operators continued and resulted in more studies that focused on the process of selecting nodes in both crossover and mutation operators. It is impossible to discuss all the research made about this subject, but it will be useful to mention some of them (Altenberg et al., 1994; Tackett, 1994; Tackett & Carmi, 1994; Hengpraprohm & Chongstitvatana, 2001; Majeed & Ryan, 2006; Uy et al., 2011, 2013).

2.3.3.2. Development of Individual's Representation Structure

GP uses a hierarchical parse-tree structure to represent solutions. This structure can be easily transformed into LISP expression using S-expression. Parse-tree representation was not the only one used in GP. Several approaches were developed and implemented to increase the performance, and decrease the execution time of GP. The developed structures varied between linear, stacked, binary, and other.

In 1994, a stack-based GP model was introduced with the ability to work in stack-based virtual machines. In this new model, functional terms are protected and will not be executed if its arity is not satisfied. For instance, if a *sin* functional node has a stack value equal to zero, the model will bypass it because it will not be executed correctly. Perkis used the reverse polish notation to make the calculations since the expressions in this notation are executed in its sequence unlike the syntax of parse trees. Perkis stated that this model can decrease the total execution time as it ignores the unsatisfied expressions. Figure 2.8 shows an example of the stack-based representation

where it helps to protect the unsatisfied functional terms. Perkis evaluated his model using the same problems used by Koza (Koza, 1992, 1994). The results showed that the model gave almost the same results using the same functional sets, but the performance increased when additional functions were added (Perkis, 1994).

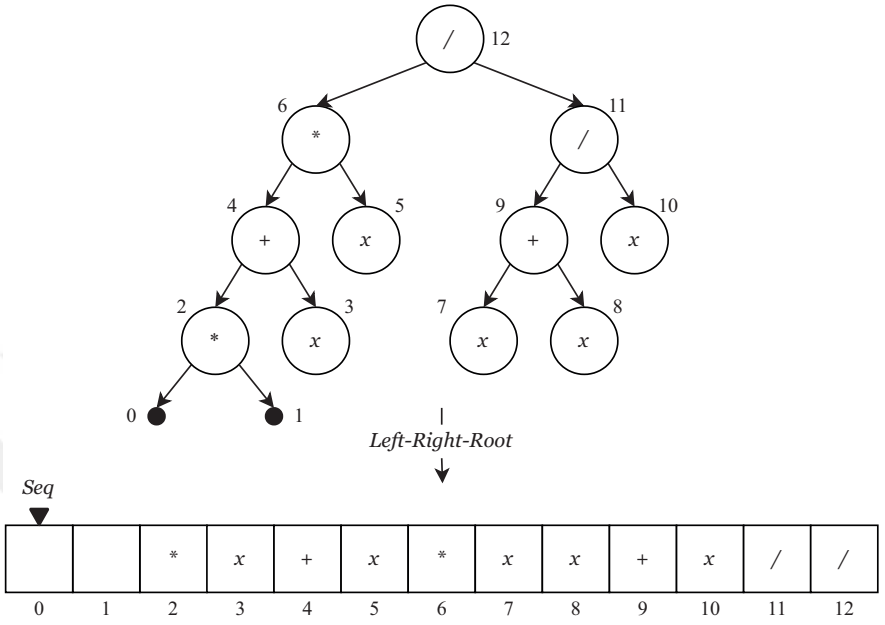


Figure 2.8. Stack-based individual representation in GP

Miller and Thomson introduced cartesian genetic programming (CGP) in which solutions are represented and indexed in a layered graph-based structure. Each node is replaced with an arity-inputs functional block. This structure will transform each individual into a digits-string as shown in Figure 2.9. The main advantage of this method is reducing the bloat in the tree-based methods which happens due to the continuous growth. The performance of this method was investigated using different sizes/lengths of individuals (Miller & Thomson, 2000).

In 2001, gene expression programming (GEP) was presented as a new method for representing programs/solutions (Ferreira, 2001). Parse trees are transformed into a linear combination structure with a head and tail using a breadth-first sequence. Figure 2.10 shows an example of a parse tree converted to gene expression. While the head part contains functional and terminal expressions, the tail contains only terminals. Both head and tail parts have fixed sizes to prevent expression from growing infinitely. The performance of GEP was investigated using a set of problems including symbolic

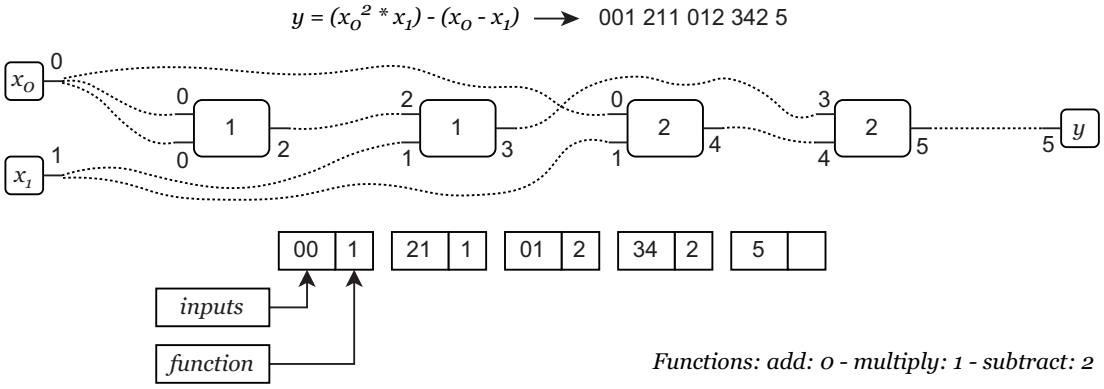


Figure 2.9. Individual representation in CGP

regression, classification, and block stacking problems. GEP was improved later in 2006 when a new modified version of it was presented by the Chinese researcher Xin Li (Li, 2006).

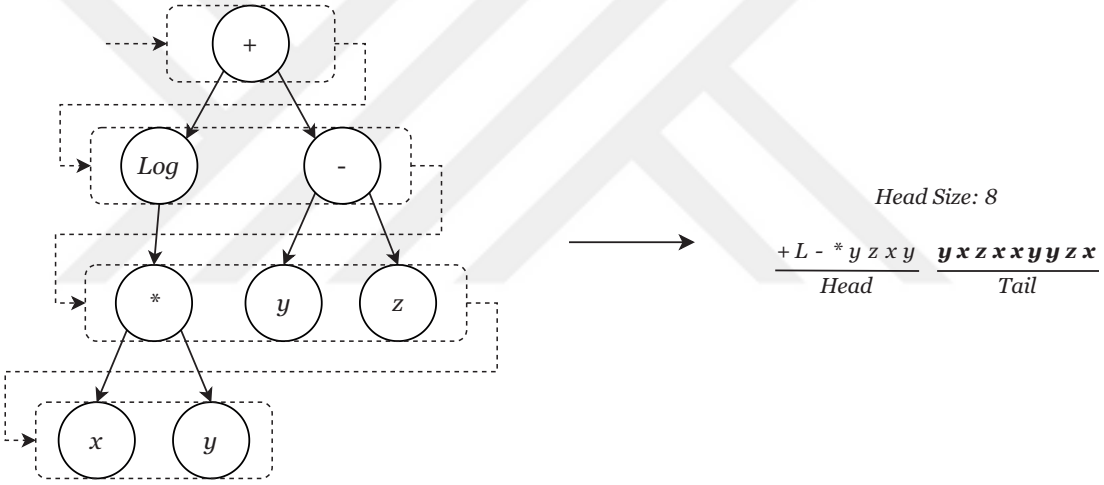


Figure 2.10. Gene expression representation in GEP

In linear genetic programming (LGP), programs are represented as simple code instructions with at most one operation. Every instruction is assigned to a variable. Each program in LGP consists of multiple instructions. Programs are limited to a specific size to prevent continuous growth problems (Brameier, 2004). Figure 2.11 shows an example of a solution tree represented linearly. Multi Expression Programming (MEP) is very similar to LGP since it uses fixed-size chromosomes that have different genes. MEP is considered one of the linear-based representation GPs (Oltean & Dumitrescu, 2021).

In the grammatical evolution (GE) method, expressions are represented in binary

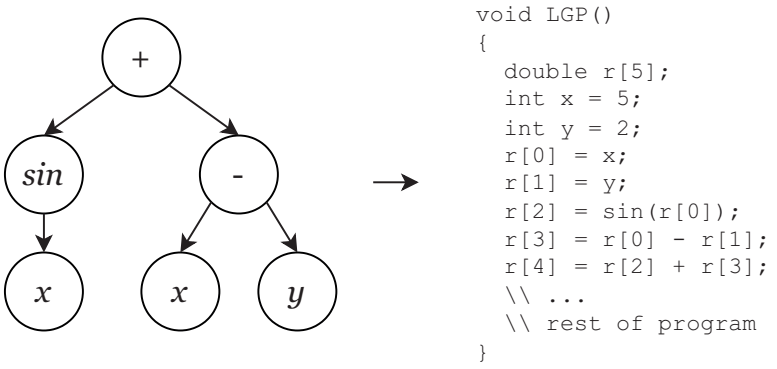


Figure 2.11. Linear representation in LGP

variable arrays. It uses the Backus Naur Form (BNF) notation to produce the rules for selecting the expressions that form the different individuals or programs. GE prunes the unnecessary genes to help the crossover operator be faster and more effective (Ryan & O'Neill, 1998; Ryan et al., 1998). (McKay et al., 2010) discussed the formalized grammar structure that is used in GP where it plays important role in improving efficiency while solving problems. Oltean and Grosan performed different experiments using five test problems to evaluate the performance of MEP, GEP, LGP, and GE (Oltean & Grosan, 2003).

2.3.4. Other Evolutionary Automatic Programming Methods

Developments and research were not limited to genetic algorithm-based automatic programming methods. Other nature-inspired algorithms proved their efficiency in solving optimization problems, especially numeric ones. Therefore, different swarm intelligence optimization algorithms were taken also as a starting point for developing new automatic programming methods.

Johnson modified the clonal selection algorithm (CSA) which is an artificial immune system (AIS) and added the ability to solve symbolic regression problems (De Castro & Von Zuben, 2000; De Castro & Timmis, 2002). CSA used a parse-tree structure for representing solution programs. The experimental tests showed that CSA outperformed the standard GP in solving a simple symbolic regression problem (Johnson, 2003). In 2006, Musilek et al. introduced the AIS-based immune programming (IP) method which uses stacks to represent the antibodies (programs) as in the stack-based GP. IP method uses a sequence of operations to evolve the

antibodies or the candidate programs such as cloning, replacement, and hypermutation. These operations help the IP method to develop and export several programs as problem solutions. Musilek et al. investigated the performance of IP using several test problems including polynomial and bivariant symbolic regression problems.

Dorigo et al. introduced the ant colony optimization (ACO) algorithm in 1996 as a new optimization system (Dorigo et al., 1996). In 2000, Roux and Fonlupt modified ACO to be able to solve symbolic regression problems. Roux and Fonlupt introduced the first model of ant programming (AP) method in which each ant creates a program using a parse-tree structure as in GP. The difference is that every node in the tree has a table to store the pheromone rate of each term in both functional and terminal sets. AP initializes pheromone rates equally before updating them at the end of each generation. Terms with higher pheromone rates are more likely to be selected to replace the current node's term. Figure 2.12 shows an example of pheromone rates tables.

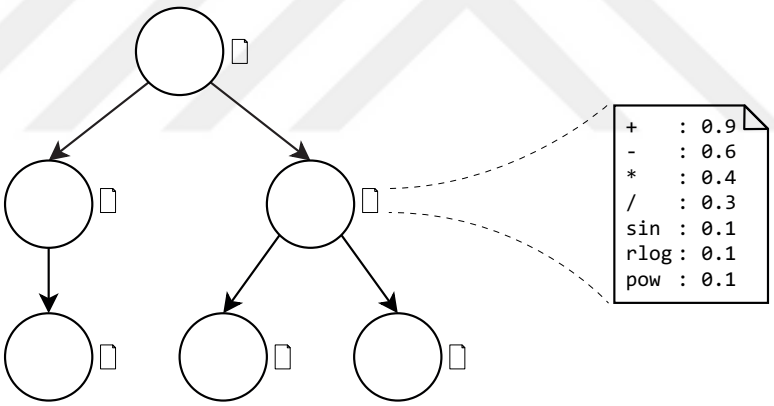


Figure 2.12. Pheromone rate tables in AP

The performance of AP was investigated using two symbolic regression problems, and an 11-MX simulation. Roux and Fonlupt compared the results of the experimental tests with the results of the standard GP (Roux & Fonlupt, 2000). In 2002, Keber and Schuster introduced the generalized ant programming (GAP) method which can solve problems using context-free-grammar represented solutions. Keber and Schuster used GAP in approximating the valuation of paying stock databases (Keber & Schuster, 2002). Shirakawa et al. presented the dynamic ant programming (DAP) method that uses a new tree construction method in which the artificial ants

create the solution tree using functional terms once, whereas terminals can be used repeatedly. Shirakawa et al. evaluated the performance of DAP by solving three various symbolic regression problems and compared the results to GP where the results of DAP showed its superiority over GP in all test cases (Shirakawa et al., 2008).

Another approach was introduced when Karaboga et al. presented the artificial bee colony programming (ABCP) method. ABCP used the artificial bee colony (ABC) algorithm which is a nature-inspired algorithm introduced in 2005 (Karaboga, 2005; Karaboga & Basturk, 2007, 2008). Solutions/Programs in ABCP are represented in a parse tree structure as in GP. In ABCP, bees are divided into three groups, employed, onlooker, and scout bees. While employed bees create the new programs by performing sharing process, the onlooker bees use the probability to select programs to modify it. Programs in ABCP have an expiration limit which determines the validity of the program. If a program became invalid, then APCP replaces it with a new one using scout bees. Karaboga et al. evaluated ABCP using different symbolic regression problems and compared the results to different automatic programming methods (Karaboga et al., 2012). Gorkemli and Karaboga presented three different improved versions of ABCP; quick ABCP (QABCP), semantic ABCP (SABCP), and quick-semantic ABCP (QSABCP). The three versions added improvements to the standard version, especially QSABCP which combined the advantages of the two other improved versions (Gorkemli & Karaboga, 2019). Arslan and Ozturk presented another version in 2019 which is the multi hive ABCP (MHABCP) and used it to solve high dimensional symbolic regression problems (Arslan & Ozturk, 2019b, 2019a).

Covid-19 treatment-inspired immune plasma algorithm (IPA) (Aslan & Demirci, 2020) was modified also to gain the ability to solve symbolic regression problems when Arslan introduced the immune plasma programming (IPP) method in 2021. In IPP, programs are evolving using the restoration operator that has the same characteristics as the sharing operator. The newly proposed IPP method was used in forecasting and modeling a time-series dataset and the results were compared to the results of ABCP (Arslan, 2021).

All developments led to a significant increase in the performance of the automatic programming methods. These methods are used in different working fields due to their

reliability and efficiency in solving real-world data-driven problems. The development process will continue to improve the performance of current methods or present new automatic programming techniques.

3. MATERIALS AND METHODS

Most automatic programming techniques are developed using swarm-based optimization algorithms. This chapter gives a brief overview of the nature-inspired firefly algorithm (FA) that is taken as a starting point to develop firefly programming (FP) and difference-based firefly programming (DFP) methods. The first section describes the characteristics of FA and the mechanisms that it uses to avoid local traps. The second section delves into the details of both FP and DFP by explaining their concepts and processes.

3.1. Firefly Algorithm

Firefly algorithm (FA) is a swarm-based nature-inspired algorithm introduced by Chinese mathematician Xin-She Yang in 2008 (Yang, 2008, 2009, 2010, 2017). Yang inspired the idea of FA by observing the behavior of fireflies in nature as in most of the nature-inspired algorithms (Fisher, 2009). FA is used for solving different optimization problems due to its efficiency in finding the optimal solutions. FA was used for UAV route planning (Henrio et al., 2019; Gharrad et al., 2021), 3D localization of UAV-based stations (Aliwi et al., 2020b), text summarization (Tomer & Kumar, 2021) and classification (Marie-Sainte & Alalyani, 2020), clustering (Senthilnath et al., 2011) and classifying data (HimaBindu et al., 2020), big data processing (Tian et al., 2019), objects tracking (Gao et al., 2013), and much more optimization problems.

3.1.1. The Behavior of the Fireflies in Nature

Firefly is an insect that belongs to the beetle insect group that is classified as a winged type and emits discrete lights at night (Branham et al., 2010). According to the living environment, more than two thousand types with different shapes and lights were registered around the world (Martin et al., 2019; Lewis & Cratsley, 2008). Fireflies usually live in high-temperature and tropical regions, especially in the forests where the light does not reach inside because of the high-density overlapping trees. Because of the high temperature of these regions, fireflies prefer to be active nightly when the temperature goes down. Since fireflies live in such environments, communicating using

lights becomes very effective. Fireflies use this ability for different purposes including mating with the other sex, prey catching, warning the swarm from a possible danger, or even telling each other about food sources.

3.1.2. The Describe of Firefly Algorithm

Communication way between fireflies within the swarm inspired Yang to develop FA. As in most swarm-based AI algorithms, each firefly in the swarm represents a possible solution to the problem (Yang, 2008, 2009, 2010, 2017). Yang stated that FA is based on two main concepts: *brightness* and *attractiveness*. While brightness in FA determines the performance of each firefly, attractiveness shows the extent of the effect a firefly has over the others (Yang, 2008, 2009, 2010, 2017). Yang summarized FA using the following characteristic points:

- Fireflies are unisex, there is no difference between fireflies.
- The more light-emitting firefly will attract the less emitting ones.
- With the increase of brightness, the attractiveness will increase too.
- The distance between fireflies impacts the amount of attractiveness. The increase in the distance will reduce the attractive power of the firefly due to the absorption of the emitted light by the surrounding environment.
- A suitable objective function determines the brightness of each firefly.

Figure 3.1 shows the flowchart of FA. It starts by defining the three control parameters; α, β_0, and γ. Control parameters are used to optimize the performance of FA. Then, a set of n fireflies is generated randomly as the initial generation after taking into consideration the limitations of the problem. Each firefly in this generation is a solution to the problem that is being solved. A proper objective function is selected to determine the brightness of each firefly (Yang, 2008, 2009, 2010, 2017). In case of maximization problems, brightness $I_0(x)$ is directly proportional to the value of objective function $f(x)$, while it is inversely proportional to it in case of minimization problems, see Eq. 3.1.

$$\begin{aligned} I_0(x) &\propto f(x) \quad , \quad \text{for } \textit{maximization} \text{ problems} \\ I_0(x) &\propto \frac{1}{f(x)} \quad , \quad \text{for } \textit{minimization} \text{ problems} \end{aligned} \tag{3.1}$$

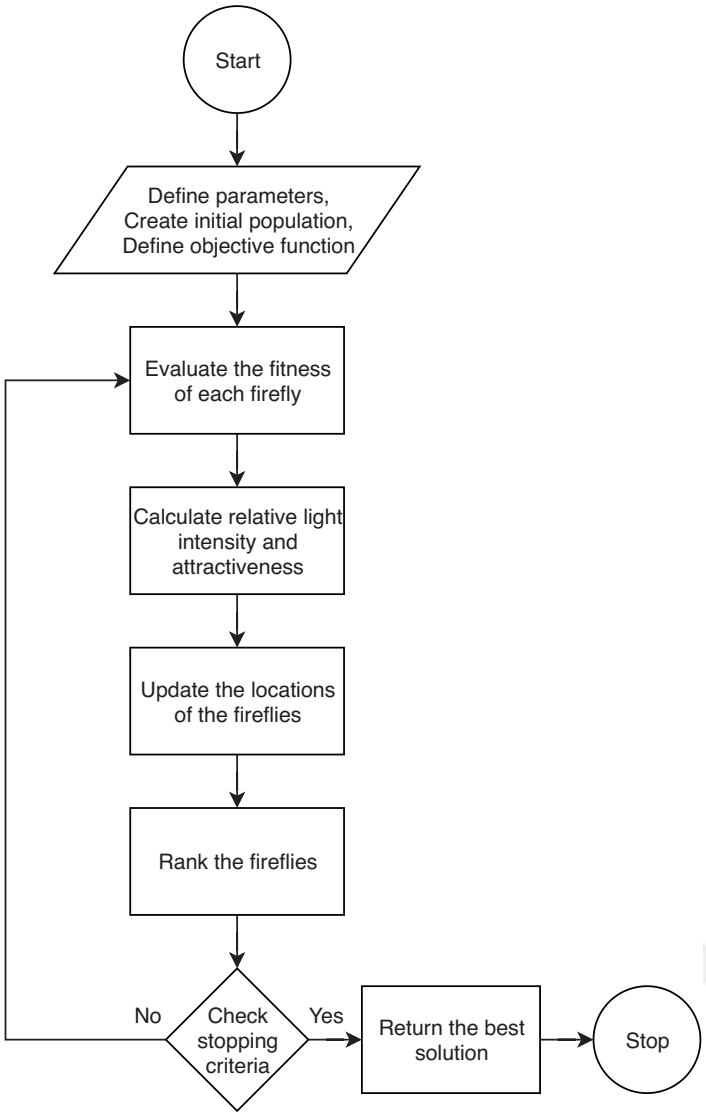


Figure 3.1. Flowchart of firefly algorithm

As long as termination criteria are not met, FA continues to compare the relative brightness of each firefly with all other fireflies as lines 5-7 show in Alg. 1, where FA compares the relative brightness of firefly i with firefly j. If the relative brightness of firefly i at distance r is more than the value of relative brightness of firefly j at the same distance, firefly i is considered brighter and it will start to attract firefly j towards itself (Yang, 2008, 2009, 2010, 2017). The relative brightness is defined as the amount of light that reaches a specific point, which can be calculated using Eq. 3.2,

$$I = I_0 e^{-\gamma r_{ij}^2} \tag{3.2}$$

where I_0 represents the value of the brightness, I is the amount of this brightness that

reaches the other firefly at r distance, and γ is the absorption coefficient. Eq. 3.2 shows that the relative brightness I is inversely proportional to the distance r between fireflies. Therefore, the light intensity at distance r is less than its intensity at the source. This is a real phenomenon where the light intensity gradually decreases due to the absorption of light from the surrounding medium. In FA, light absorption is determined using the control parameter γ (Yang, 2008, 2009, 2010, 2017). The distance between fireflies is the euclidean distance which is calculated using Eq. 3.3, where x_{ki}, x_{kj} represents the k^{th} location's parameter of fireflies i and j, D is the problem's dimension.

$$r_{ij} = \sqrt{\sum_{k=1}^{D} (x_{ki} - x_{kj})^2} \tag{3.3}$$

Light intensity affects the attractiveness of the firefly, where it is directly proportional to the light intensity (this is observed in real life when insects are attracted to brighter light sources). Similarly, the relative attractiveness at distance r is calculated using Eq. 3.4 where β_0 is the globally initialized attractiveness parameter.

$$\beta = \beta_0 e^{-\gamma r_{ij}^2} \tag{3.4}$$

If the light intensity I_i of firefly i near firefly j is more than the light intensity I_j of firefly j's near firefly i, then, firefly i starts to attract firefly j causing a change in the location of firefly j. The change in location is determined using Eq. 3.5,

$$x_{ki}^{new} = x_{ki}^{old} + \beta(x_{kj} - x_{ki}^{old}) + \alpha(\sigma - 1/2) \quad , \quad \forall k \in D \tag{3.5}$$

where σ is taken randomly between 0 and 1 once before every attraction process, and α is the randomness coefficient that is initialized at the beginning. Replacing Eq. 3.4 in Eq. 3.5 shows that the change in the location depends on the value of the β_0 parameter. When $\beta_0 = 0$, the attracted fireflies start to move randomly. Therefore, the value of β must be selected carefully (Yang, 2008, 2009, 2010, 2017).

As the pseudo-code of FA (Alg. 1) shows, the attraction process continues until a specific terminating criterion is met, which is the maximum evaluation number in our case studies.

Algorithm 1 Firefly Algorithm

```
Initialize all the parameters (α, β0, γ and MaxGenerations)
Initialize randomly a population of n fireflies
Evaluate the brightness of the initial population at xi by f(xi) for i = 1, ..., n
while g < MaxGenerations do
    for All fireflies (i = 1, ..., n) do
        for All fireflies (j = 1, ..., n) do
            if firefly j is better than i then
                Move firefly i towards j
                Evaluate the new solution and accept better solutions
            end if
        end for
    end for
    Rank and update the best solution found so far
    Update iteration counter g ⇐ g + 1
end while
Export the best solution
```

Although the basic suggested algorithm is efficient, Yang stated that tuning up some control parameters can help FA to speed up the process of optimization in addition to increasing its performance, which will give the algorithm flexibility in solving different problems (Yang, 2008, 2009, 2010, 2017). Yang proposed updating the randomness coefficient (α) at the end of every generation using two different ways, the first one as in Eq. 3.6, where α_0 is the initial value of α parameter, θ is a random value in range $(0, 1]$, g represents the current generation.

$$\alpha = \alpha_0 \theta^g \tag{3.6}$$

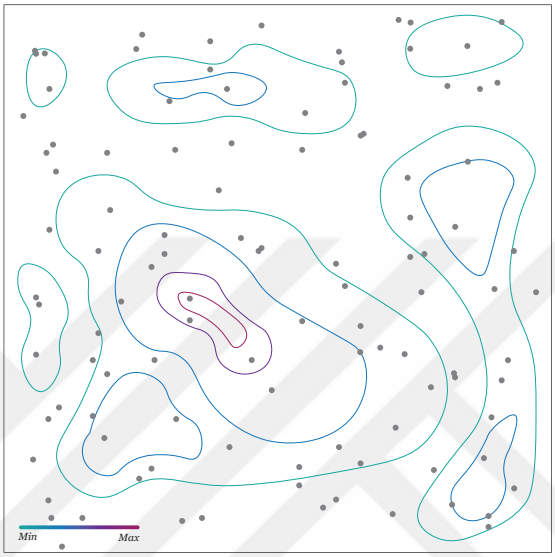
The second model updates the value of α parameter using both of the initial (α_0) and final (α_∞) values as in Eq. 3.7:

$$\alpha = \alpha_\infty + (\alpha_0 - \alpha_\infty) e^g \tag{3.7}$$

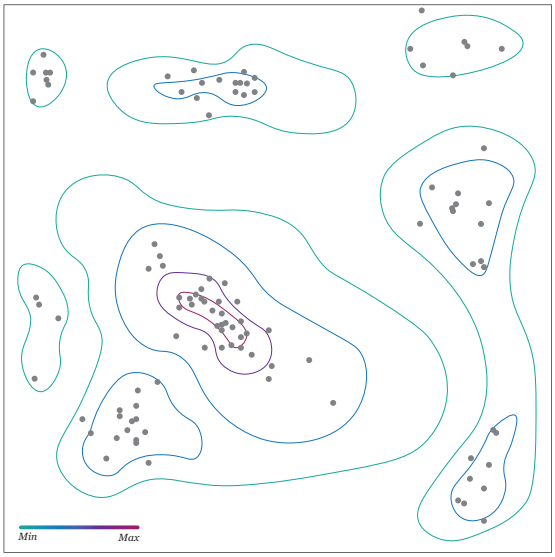
3.1.3. Escaping Local Optima

In swarm-based algorithms, escaping local traps is a very important or even an essential mechanism. Firefly algorithm uses a simple mechanism to avoid such traps. The firefly algorithm depends primarily on the total number of fireflies to find the best optimal solution. As a result of applying the attraction process, fireflies are divided

into small groups in which each group swarms around a local optimum. If the total number of the groups is enough to cover all local optimums, then, the global optimum is covered too (Yang, 2008, 2009, 2010, 2017). This helps FA escape local traps and makes it suitable to solve several optimization problem types. Figure 3.2 shows how the attraction process creates firefly groups in which each of them swarms around a local optimum.



(a) Before optimization



(b) After optimization

Figure 3.2. Firefly subgroups positions while solving maximization problem with FA

3.2. Firefly Programming

Firefly programming (FP) is an automatic programming method that uses the principles of symbolic regression to solve problems (Aliwi et al., 2020a). As well as in most automatic programming methods, FP extends the firefly algorithm (FA) to gain the ability to solve different problems by optimizing several candidate solutions at a time. This section presents the standard version of FP before introducing an improved version too.

3.2.1. Standard Firefly Programming (FP)

Firefly programming (FP) is an extended version of the firefly algorithm presented in 2020 (Aliwi et al., 2020a). In FP, every individual represents a possible solution to the problem. FP uses a parse-tree structure to represent solutions as in the genetic programming method (Koza, 1992, 1994; Koza et al., 1999). Tree solutions consist of different-value connected nodes. Nodes are either functional or terminal nodes. Terminal nodes are leaf nodes that are found only on the edges, and the value of it is constant number or a problem-dependent variable $(x_1, x_2, x_3, \ldots)$. The others are functional nodes that link terminal or other functional nodes together and determine the relation between them. To make it clear to the reader, Figure 3.3, shows a possible solution for a symbolic regression problem represented in a parse-tree structure where nodes are connected and combined forming the mathematical expression $sin(x_1) - (1 + x_2)$. As shown in the figure, terminal nodes exist only at the ends, unlike functional nodes.

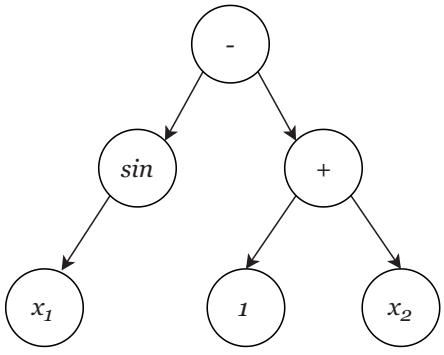


Figure 3.3. Expression representation in Koza tree

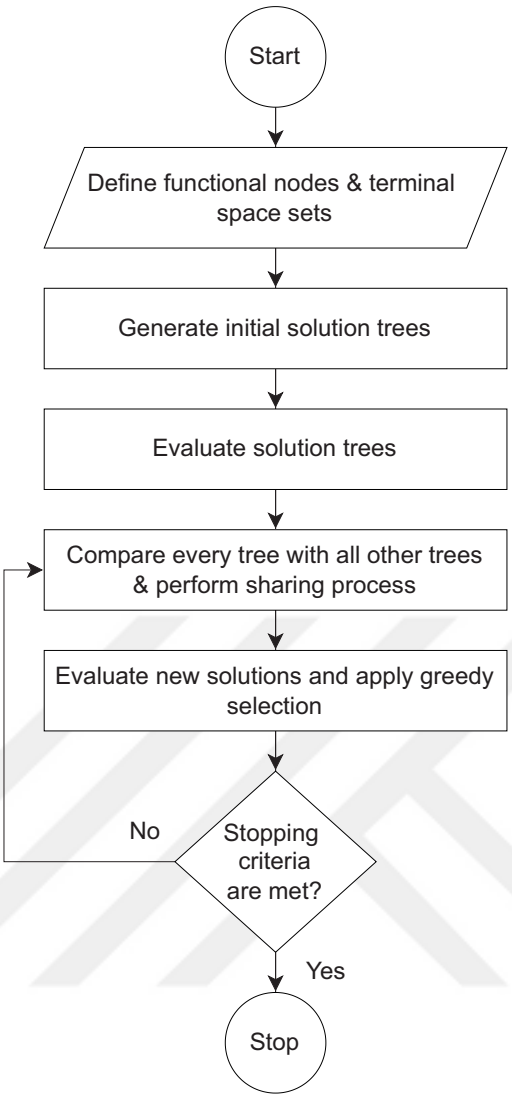


Figure 3.4. Flowchart of firefly programming (FP) method

The flowchart of FP in Figure 3.4, shows that it starts by generating initial tree solutions randomly using *ramped half and half* method which is a combination of two sub-methods; *full* and *grow*. Koza in his first model used this forming method to increase diversity and avoid similarity in the generated solutions (Koza, 1992, 1994; Koza et al., 1999). In both *full* and *grow* methods, trees are formed depending on two forming limits; initial minimum and maximum depth limits. These two limits are only used while generating trees. While the initial minimum depth determines the minimum node levels that the tree is consisted of, the initial maximum depth determines the maximum node levels. In both *full* and *grow* methods, all nodes that have a depth value equal to the minimum limit or less must be functional nodes. In the *full* method, all nodes are functional nodes except the nodes with a depth equal to the initial maximum

depth limit. On the other hand, in *grow* method, the solution tree doesn't have to reach the maximum depth limit in all branches. Figure 3.5 shows two different trees built with *full* and *grow* methods with a minimum depth of 2 and an initial maximum depth of 4. The previous figure shows that in *grow* method, terminal nodes can be found in level 3 and level 4 while they can be found only in level 4 nodes in *full* method.

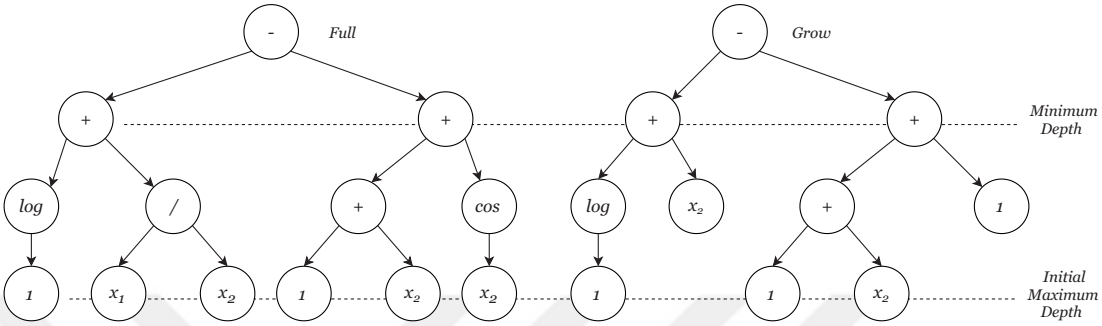


Figure 3.5. Full and Grow methods

After generating the initial solution trees, FP uses the selected objective function to determine the brightness of each tree (firefly). Then, FP compares the brightness of each tree to all other ones. Although FA stated that brighter firefly attracts the less bright one, this cannot happen in FP since fireflies (trees) are not moving on a geometric surface. Therefore, the attraction process in FP is replaced with another one called *sharing* operation. In sharing operation, a subtree instance is taken from the brighter tree and glued to an instance of the less bright tree. The subtree's instance is randomly selected, which increases the diversity in solution trees. Figure 3.6 shows an example of sharing operation. After applying the sharing operation, FP evaluates the new tree and compares it with the original one to choose the best among them in a greedy selection as in Eq. 3.8.

$$x_i = \begin{cases} x_i^{new} & \text{if } Brightness(x_i^{new}) \text{ is better than } Brightness(x_i^{old}) \\ x_i^{old} & \text{otherwise} \end{cases} \tag{3.8}$$

Due to the continuous application of the sharing operation, solution trees will continue to grow to unlimited depth values, which creates too complex solutions. To prevent this, FP uses two limitation parameters to handle this problem; the maximum tree depth, and the maximum node number.

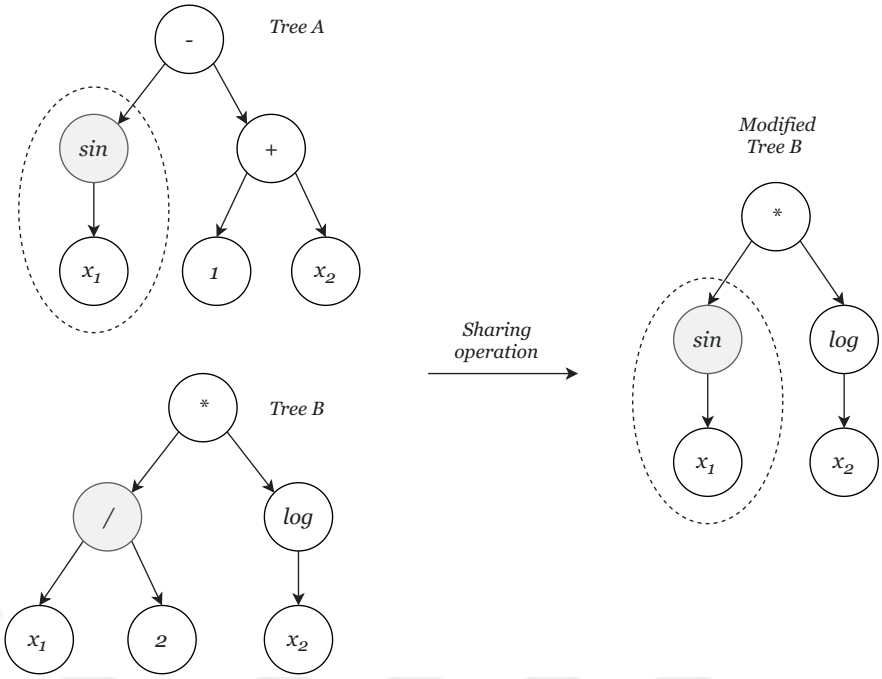


Figure 3.6. Sharing operation

Algorithm 2 Firefly Programming (FP)

```
Initialize parameters (initMinDepth, initMaxDepth, maxDepth and MaxGenerations)
Define functional and terminal sets
Generate initial n solution trees using ramped hald and half.
Evaluate trees using objective function
while g < MaxGenerations do
    for All trees (i = 1, ..., n) do
        for All trees (j = 1, ..., n) do
            if Tree_j is better than tree_i then
                Perform sharing operation from Tree_j to Tree_i
                Check new tree's depth
                Evaluate the new tree and accept if better
            end if
        end for
    end for
    Rank and update the best solution tree found so far
    Update iteration counter g ⇐ g + 1
end while
Export the best solution tree
```

If a solution tree exceeded one of the controlling limits, FP applies a proper mechanism to override this problem, like replacing the subtrees that exceeded the limits with high-value terminal nodes as suggested in (Gorkemli & Karaboga, 2015; Johnson,

2009), pruning the exceeded branches, or even generating a new tree to replace the old one. Alg. 2 shows the pseudo-code of FP.

3.2.2. Difference-based Firefly Programming (DFP)

In order to improve the performance of FP, a modified version of it is presented which is called difference-based firefly programming (DFP). In DFP, the main processes are modified in a way that increases the efficiency of FP. This section explains the main differences between FP and DFP according to the flowchart of the new version (Figure 3.7).

After generating the initial solution trees and before evaluating each one of them, DFP performs a new operation to decrease the complexity of the trees, called the *simplification* operation. The previous subsection mentioned that FP uses some parameters to prevent trees from growing continuously by limiting the maximum depth or the total number of nodes. Although these two parameters are still used in DFP, it uses the simplification operation to reduce such cases. The simplification operation merges subtree nodes into one with an equivalent value, and it's applied only to branches that do not have any variable nodes. Figure 3.8 shows an example of the simplification operation where the expression 1 + 1 is replaced with only one node with a value of 2. The simplification operation reduces the possibility of exceeding the limits and generates new terminal values too.

After evaluating solution trees using a proper objective function, DFP starts to compare each solution tree to all other trees before performing the sharing operation. According to Eq. 3.4 in Section 3.1., FA performs the attraction process depending on the distance between fireflies. Therefore, two close fireflies attract each other more than two far ones. In FP, sharing operation (which is equivalent to the attraction process in FA) is applied to all fireflies without taking into consideration the difference between fireflies, which can be unfair. This is the major difference between DFP and FP, where there are four different cases according to the difference in brightness values compared to the three control parameters: α, β, and γ. The four cases are not valid if the condition $\alpha < \beta < \gamma$ is not met. In the scope of this work, the number of repeated operations according to the different cases is shown in Table 3.1. It is worthy to mention that the four mentioned cases can have different operations than the listed operations in

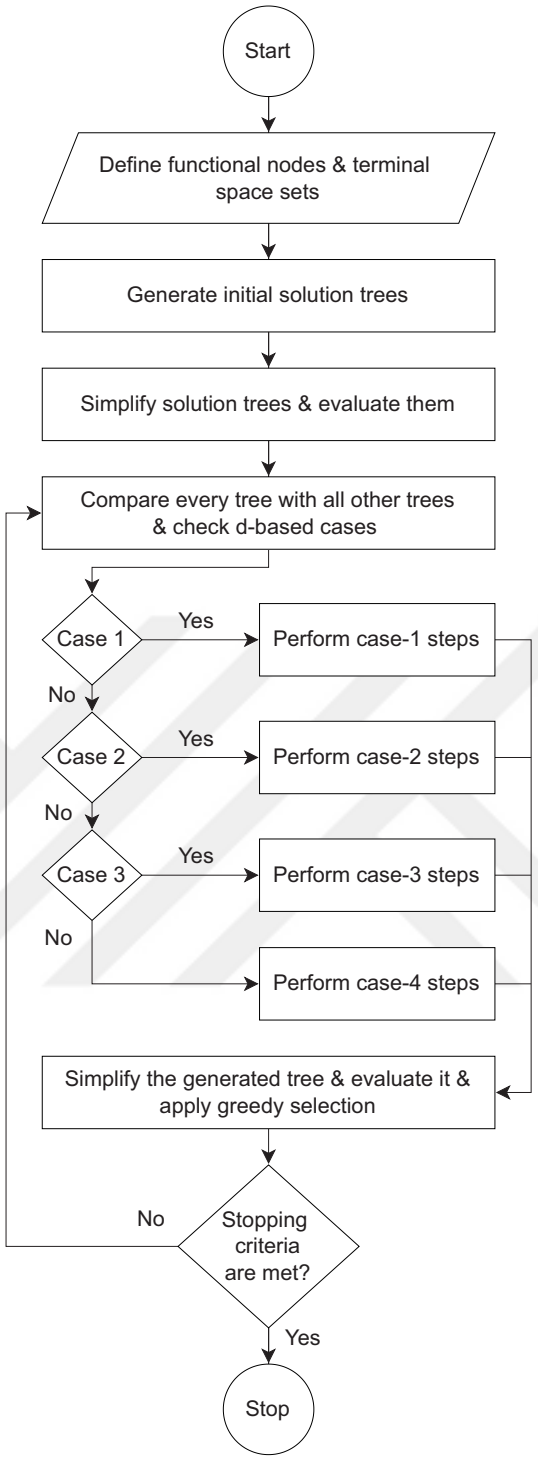


Figure 3.7. Flowchart of difference-based firefly programming (DFP) method

Table 3.1. These operations must be set previously by the analysts before starting the optimization process. The difference between two solution trees i and j is calculated using Eq. 3.9, where the brightness of each tree is determined using the objective function.

$$d = \left| Brightness_i - Brightness_j \right| \tag{3.9}$$

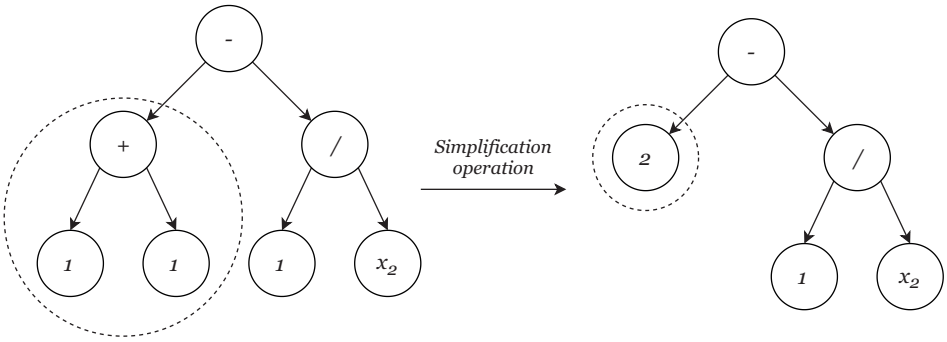


Figure 3.8. Simplification operation

Table 3.1. Attraction cases according to difference

	Attraction process	
Cases	Sharing operation	Substitution operation
Case-1: $d > \gamma$	2	1
Case-2: $\gamma > d > \beta$	1	1
Case-3: $\beta > d > \alpha$	1	0
Case-4: $\alpha > d$	0	1

Table 3.1 shows that the attraction process in DFP is consisted of two operations, *sharing*, and *substitution*. Unlike the sharing operation which was described in Subsection 3.2.1. previously, the substitution operation happens in just one tree. In this operation, a random node is chosen and replaced with a suitable one. To perform this without changing the structure of this node's subtree, nodes are classified into different classes: 0-child nodes that represent terminal nodes, 1-child nodes like trigonometric, logarithmic function nodes, 2-child nodes like addition, subtraction, multiplication, and division expressions, or the power function nodes, and so on. In the substitution operation, the chosen node is replaced randomly with a new node from the same class. Taking Figure 3.9 as an example, the logarithmic functional node is replaced with a trigonometric functional node since both of them are classified as 1-child nodes. Applying the substitution operation generates solutions with a minor difference from the original solution which is useful when the algorithm needs to make small changes to reach the optimal solution.

As an optional step, tuning up control parameters can be done according to the results of DFP at the end of each iteration or generation which will lead to speeding

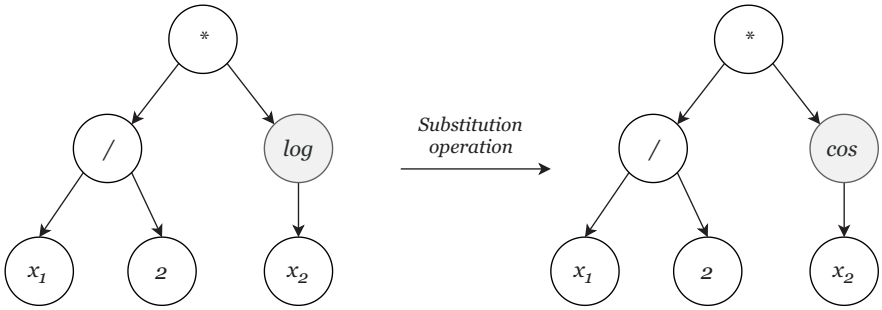


Figure 3.9. Substitution operation

up the method, and increases its performance too. Tuning up values must be done by calculating the maximum d_{max} and minimum d_{min} difference values, then using these values to update each control parameter value at the end of each generation as in Eq. 3.10:

$$\begin{aligned} d_{range} &= |d_{max} - d_{min}| \\ \alpha &= (1/4) * d_{range} \\ \beta &= (2/4) * d_{range} \\ \gamma &= (3/4) * d_{range} \end{aligned} \tag{3.10}$$

Algorithm 3 Difference-based Firefly Programming (DFP)

```
Initialize parameters (α, β, γ, DepthLimits, and MaxGenerations)
Define functional and terminal sets
Generate initial n solution trees using ramped hald and half.
Simplfy and evaluate trees using objective function
while g < MaxGenerations do
    for All trees (i = 1, ..., n) do
        for All trees (j = 1, ..., n) do
            if Tree_j is better than tree_i then
                Perform attraction operations
                Check new tree's depth
                Simplifiy, evaluate the new tree and accept if better
            end if
        end for
    end for
    Rank and update the best solution tree found so far
    Update iteration counter g ⇐ g + 1
    Tune up control parameters α, β, and γ
end while
Export the best solution tree
```

To summarize the difference between FP and DFP, two new operations were developed to improve the performance of FP. The simplification operation reduces the size of the tree without changing its actual value, which gives trees more flexibility within the same limits. The substitution operation performs minor changes to solution trees in a way that cannot be done using only the sharing operation. Although these operations slightly increase the execution time, they will increase the performance of the method as the results in Chapter 4. show. The pseudo-code of DFP is shown in Alg. 3.

4. EXPERIMENTAL RESULTS

In order to evaluate the performance of the newly proposed methods, FP and DFP were used in solving a set of symbolic regression benchmark problems, and modeling the Box-Jenkins gas furnace time series dataset. The results of both FP and DFP were compared to a set of different methods used to solve the same problems. The results of FP and DFP were taken after being coded and tested within Windows 10 WSL2.0 developing environment with 8.0 GB of memory and an Intel Core i7-7500U processor.

4.1. Solving Benchmark Problems

In this subsection, the results of FP and DFP are compared to the results of other methods that are used for solving symbolic regression problems which are genetic programming (GP) (Uy et al., 2013), and artificial bee colony programming (ABCP) (Gorkemli & Karaboga, 2015). To make sure that the introduced methods can solve different problems efficiently, a set of different problems were chosen including linear, polynomial, trigonometric, and logarithmic problems (Johnson, 2009; Keijzer, 2003; Hoai et al., 2002; Uy et al., 2013). These problems are listed in Table 4.1.

Table 4.1. Symbolic regression test problems

Problems	Number of random points	Domain
$F_1 = x^4 + x^3 + x^2 + x$	20	[-1, 1]
$F_2 = x^5 + x^4 + x^3 + x^2 + x$	20	[-1, 1]
$F_3 = sin(x) + sin(x + x^2)$	20	[0, 1]
$F_4 = sin(x^2)cos(x) - 1$	20	$[0, \frac{\pi}{2}]$
$F_5 = log(x + 1) + log(x^2 + 1)$	20	[0, 2]
$F_6 = \sqrt{x}$	20	[0, 4]
$F_7 = sin(x_0) + sin(x_1^2)$	100	$[0, 1] \times [0, 1]$
$F_8 = 2sin(x_0)cos(x_1)$	100	$[0, 1] \times [0, 1]$

Table 4.2 shows the parameters of the testing problems. In the functional set, the protected division function returns the value 1 if the denominator value is equal to zero (Gorkemli & Karaboga, 2015). In the same way, *rlog* function is a protected version of the natural logarithmic function which returns the value of zero if the data is equal to zero, otherwise, it returns the normal logarithmic value of the absolute given data

(Gorkemli & Karaboga, 2015).

Table 4.2. Symbolic regression benchmark problems' common parameters

Parameter	Value
Initial maximum depth	6
Maximum depth	15
Functional nodes	+, −, ×, ÷ (protected ver.), sin, cos, rlog, exp
Terminal nodes	1-variable: $[x, 1]$, 2-variables: $[x_0, x_1]$
Number of runs	100
Maximum evaluations	25,000

Table 4.3. FP, DFP, ABCP, and GP special parameters

FP / DFP		ABCP		GP	
Parameter	Value	Parameter	Value	Parameter	Value
Pop. size	50	Colony size	500	Pop. size	500
α	0.1	Limit	500	$P_{crossover}$	0.9
β	0.5			$P_{mutation}$	0.5
γ	1.0			Tournament	3

While forming the initial trees, the initial minimum depth limit was not used. All methods used *ramped half and half* as a tree forming method. In order to obtain better results, each problem was solved 100 times independently before taking the median value of these runs. The maximum evaluation number for each run is 25,000. The sum of difference between the targeted ($f(x)$) and the generated ($g(x)$) functions for all k testing points is used as an objective function as in Eq. 4.1. Other methods' special parameter values are shown in Table 4.3.

$$error_i = \sum_{n=1}^{k} |f(x_n) - g_i(x_n)| \quad , \quad \forall i \in [0, PopSize) \tag{4.1}$$

Table 4.4 shows the mean error of the testing problems after running all methods 100 times. DFP has the best result for F_1 while ABCP came in second place before FP and GP respectively. In F_2, the same results were obtained as DFP showed the best performance. The best performance in these two problems shows the efficiency of DFP in solving polynomial symbolic regression problems. Taking a look into Figure 4.1 and Figure 4.2, it is clear that the DFP-generated graphs are just the same as the targeted

Table 4.4. Mean error values of GP, ABCP, FP, and DFP

Method	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
GP	0.30	0.40	**0.11**	**0.24**	**0.15**	0.25	1.67	1.19
ABCP	0.11	0.19	0.21	0.55	0.24	**0.17**	0.62	0.69
FP	0.25	0.34	0.26	0.53	0.33	0.34	0.48	0.67
DFP	**0.08**	**0.17**	**0.11**	0.48	**0.15**	0.33	**0.21**	**0.34**

Lower is better

ones, whereas the FP-generated graph has an observable difference in F_1.

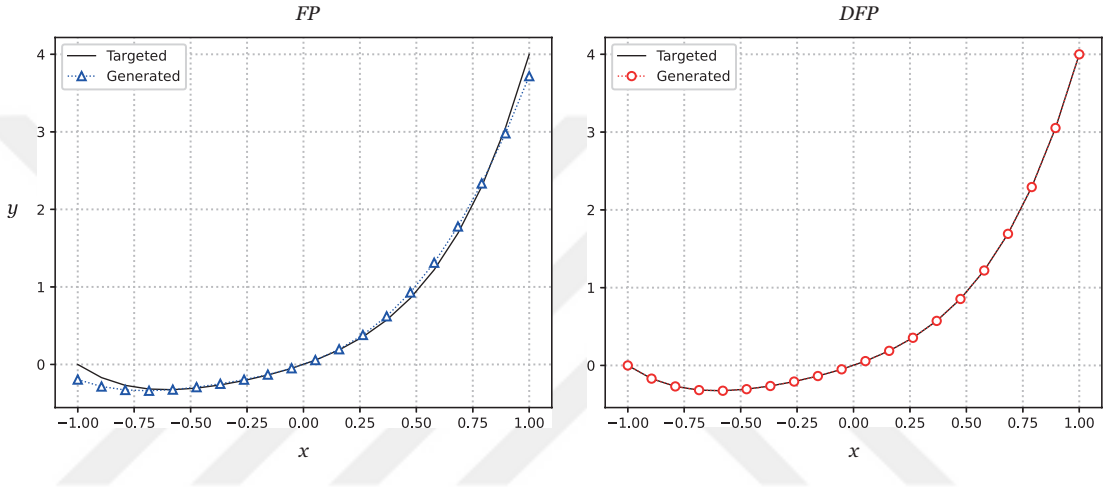


Figure 4.1. FP/DFP-generated and targeted graphs of F_1

In F_3, while FP gave the worst result, both DFP and GP gave the best error value with just 0.11. ABCP gave an average result which is not far from the result of FP. Figure 4.3 shows the FP/DFP-generated graphs of F_3 which clearly shows that DFP is more accurate than FP. In F_4 which is a trigonometric expression with two different functions, GP is the only one that performed well in this problem. All other methods gave below-average results while solving this problem. Taking a look into Figure 4.4 that shows the generated functions of F_4 using FP and DFP, the generated graphs have a significant difference from the targeted ones. This difference is unobservable in the DFP-generated graphs of F_1, F_2, and F_3.

In F_5, while DFP and GP performed well in solving the logarithmic function problem, the other two methods gave average results. The generated graphs are very similar to the targeted graph of F_5 as in Figure 4.5.

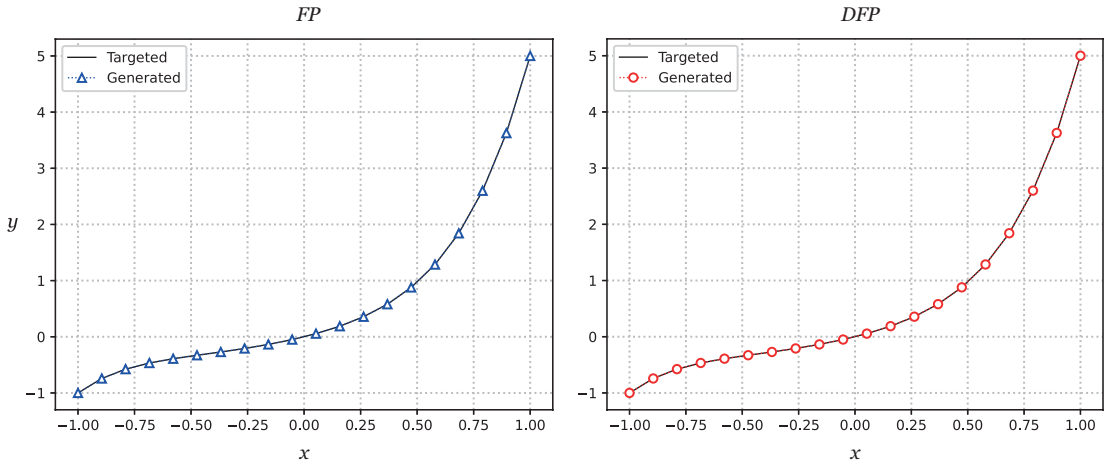


Figure 4.2. FP/DFP-generated and targeted graphs of F_2

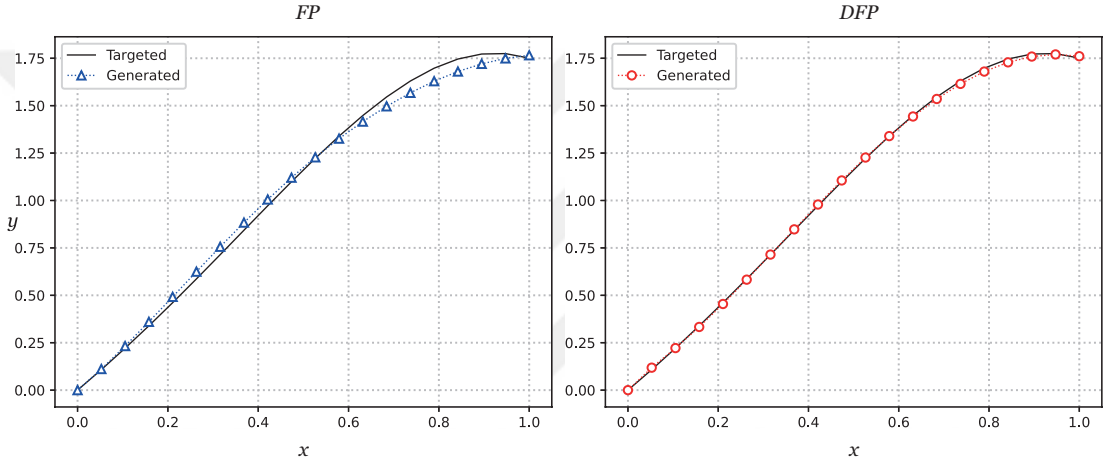


Figure 4.3. FP/DFP-generated and targeted graphs of F_3

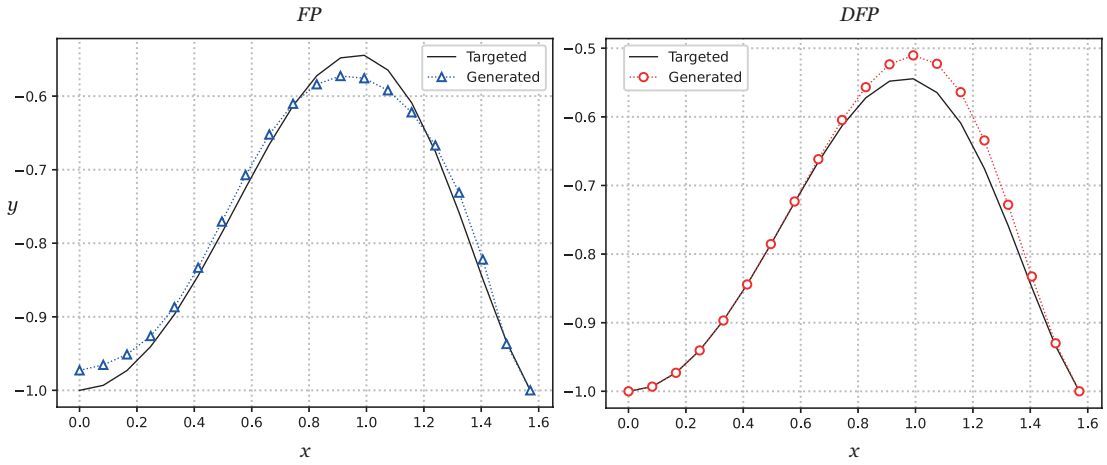


Figure 4.4. FP/DFP-generated and targeted graphs of F_4

Solving root function problems was not efficient enough using both FP and DFP where they both fell behind ABCP and GP respectively, and the graph of F_6 (Figure 4.6)

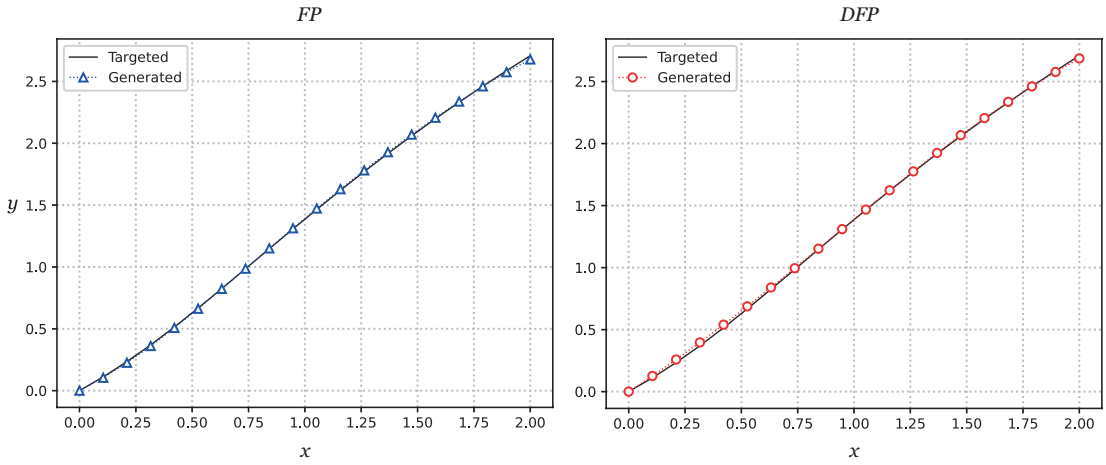


Figure 4.5. FP/DFP-generated and targeted graphs of F_5

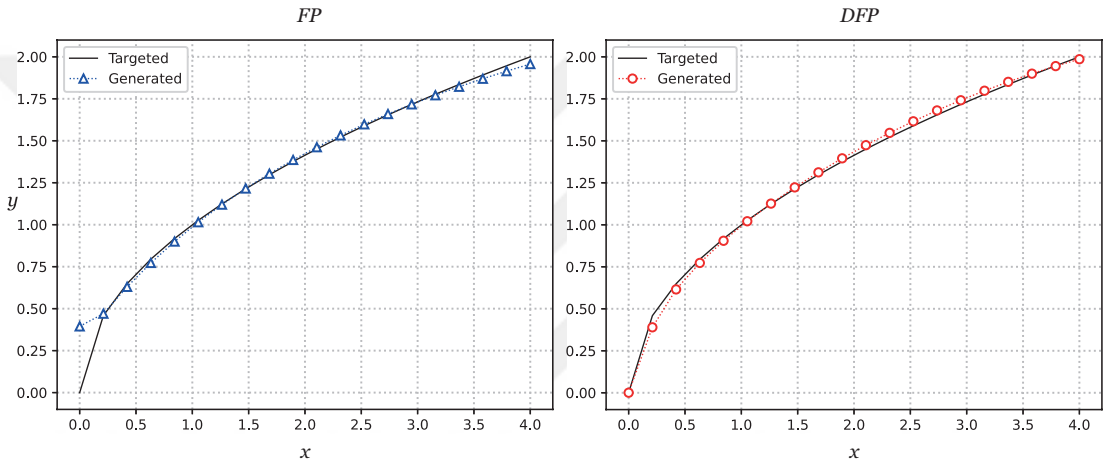


Figure 4.6. FP/DFP-generated and targeted graphs of F_6

is not as good as the graphs of F_1,F_2,F_3, and F_5. For the last two testing problems, F_7 and F_8 which are bivariant trigonometric functions, both FP and DFP performed better than ABCP which came third in solving this problem. On the other hand, GP failed to detect the pattern of these problems where the results were too far from the results of the other methods. The generated graphs of F_7 and F_8 with both FP and DFP were too close and accurate to the original graphs as Figure 4.7 and Figure 4.8 show.

Result comparison between the previous methods showed the superiority of DFP over other methods. Comparing DFP results to GP, DFP performed better in four problems, has the same performance in two, and its results were worse only in two. In the same way, DFP performed better than ABCP in all problems except F_6, where it was difficult to solve using FP and DFP. On the other hand, FP gave average results compared to GP and ABCP.

The generated solution is either an exact mathematical expression or a complex expression that gives almost the same values as the original expression. Table 4.1 shows that F_6 contains a square root function which is not included in the functional searching set. This means that it's impossible to generate the same expression using FP or DFP. The expression in Figure 4.9 represents a DFP-generated expression that is too close to F_6 within the domain of [0,4] with taking into consideration the protected division and logarithmic functions.

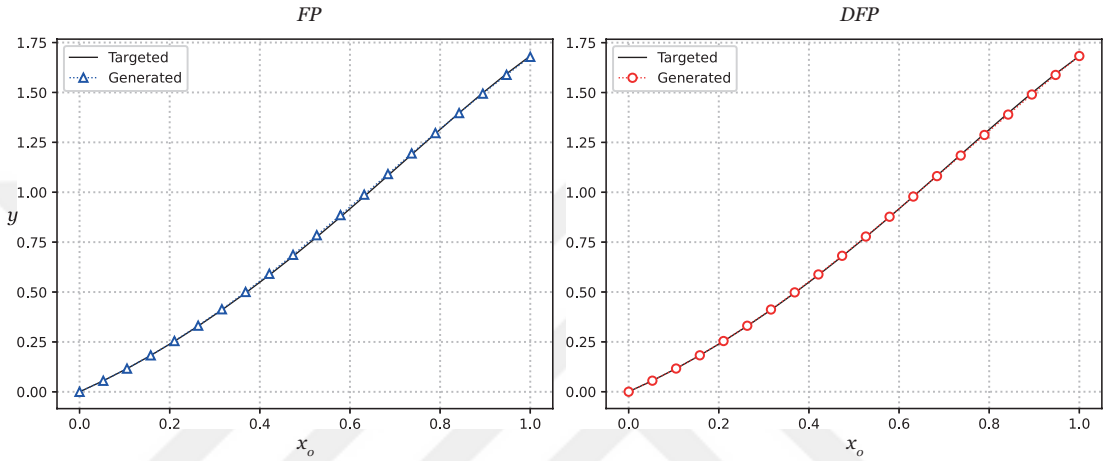


Figure 4.7. FP/DFP-generated and targeted graphs of F_7

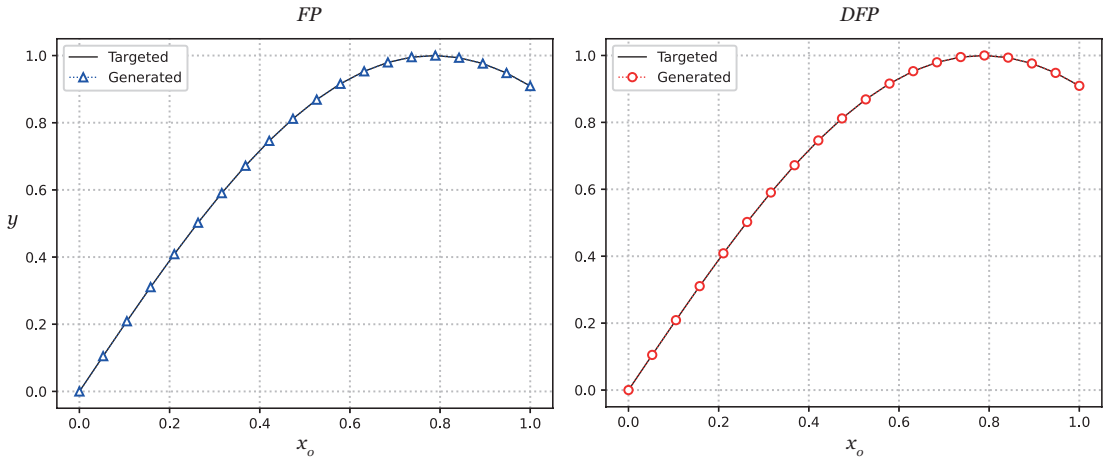


Figure 4.8. FP/DFP-generated and targeted graphs of F_8

```
y=(((x)-(rlog(((x)-(rlog(cos(((cos(1))*((cos(rlog(x)))+(x)))+(0)))))*(cos(1))
)))exp((x)/(((x)/(rlog(x)))*(((x)/(x))+(1)))))
```

Figure 4.9. DFP-generated expression of F_6

4.1.1. FP & DFP Performance Comparison

Both FP and DFP were tested also using different population size values. Table 4.5 and Table 4.6 show the results of FP and DFP after solving the same benchmark problem set in Table 4.1. The results confirm the superiority of DFP over FP in almost all cases. Table 4.6 showed that the best results were obtained in almost all cases when the population size equals 50. With the increase in population size value, the results decreased too. The main reason behind this point is the total number of evaluations. According to Alg. 3, FP and DFP have two nested `for loops` in lines 6-7 where each tree (firefly) is compared to all other trees. The total number of evaluations in each iteration equals $PopSize^2$ which is 62,500 when the population size equals 250. This number exceeds the total evaluation limit in this test which is 25,000. When the number of evaluations exceeds the limit, it means that not all trees will be evaluated during the execution time. On the other hand, the results decreased also when the population size is equal to 25. The convincing explanation of this case is that the total number of solution trees is insufficient to cover all possible solutions, which led DFP to fall into a local trap as mentioned in Subsection 3.1.3..

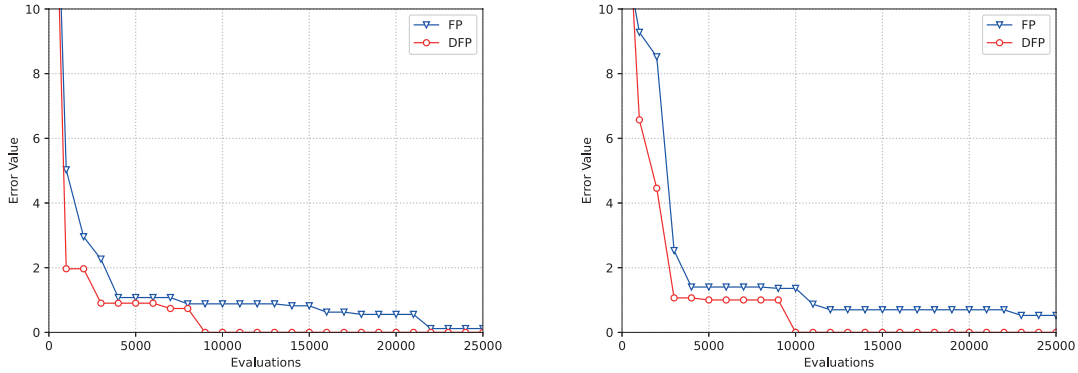
Table 4.5. FP results with different population size values

Pop. size	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
25	0.35	0.58	0.39	0.61	0.39	0.42	0.68	0.96
50	0.25	0.34	0.26	**0.53**	**0.33**	0.34	0.48	0.67
75	0.19	**0.32**	0.26	0.58	0.35	**0.33**	0.41	0.51
100	**0.18**	0.37	**0.24**	0.64	0.36	0.36	**0.40**	**0.47**
250	0.22	0.48	**0.24**	0.72	0.49	0.35	0.42	0.71
500	0.37	0.57	0.33	0.77	0.58	0.45	0.54	0.66

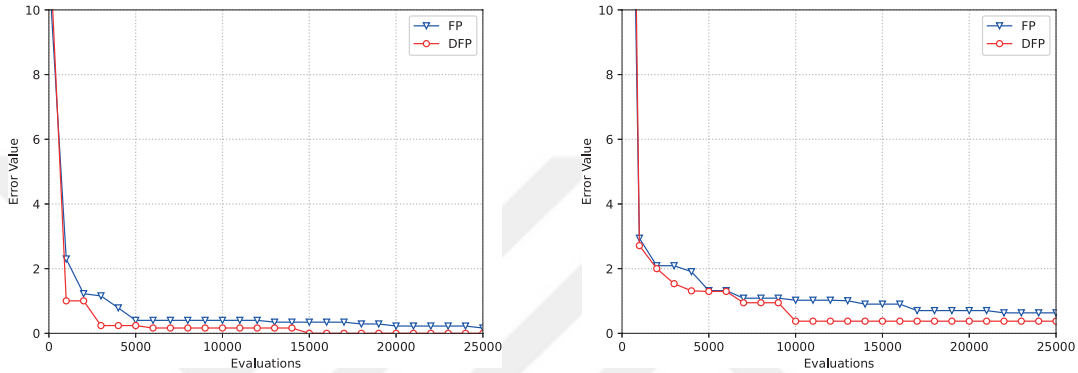
Lower is better

After taking a look at Figure 4.10 which shows the change in the error value through evaluations, DFP needed fewer evaluations to minimize the error values than FP in all test cases. The graphs proved that both simplification and difference-based attraction processes increased the performance of FP significantly.

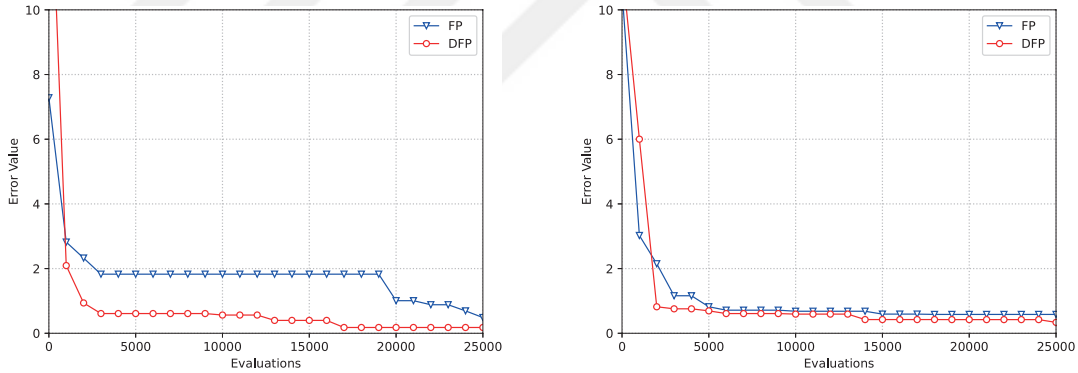
The chart in Figure 4.11 shows the total number of the generated expressions that is 100% identical to the targeted functions within testing domains after running each method (FP/DFP) 100 times independently. It shows that DFP is better than FP in



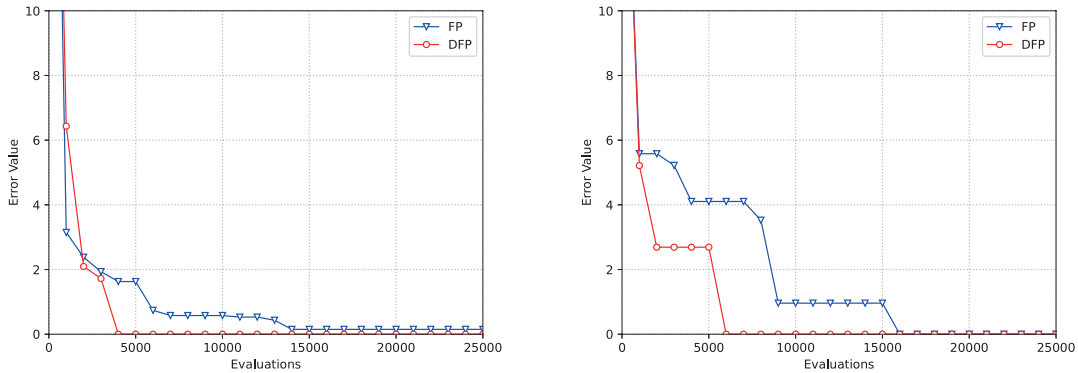
(a) F_1 (b) F_2



(c) F_3 (d) F_4



(e) F_5 (f) F_6



(g) F_7 (h) F_8

Figure 4.10. The change in the error value through evaluations using FP and DFP

Table 4.6. DFP results with different population size values

Pop. size	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
25	0.19	0.23	0.13	**0.42**	0.19	**0.33**	0.30	0.53
50	**0.08**	**0.17**	**0.11**	0.48	**0.15**	**0.33**	**0.21**	0.34
75	0.13	0.23	0.16	0.53	0.20	0.38	**0.21**	**0.31**
100	0.16	0.29	0.14	0.58	0.22	0.35	0.28	0.35
250	0.18	0.40	0.18	0.73	0.31	0.42	0.27	0.43
500	0.22	0.46	0.25	0.82	0.36	0.51	0.31	0.62

Lower is better

six functions: F_1, F_2, F_3, F_4, F_5 and F_7, whereas FP is better only in two: F_6, and F_8. Figure 4.12 shows the numbers of the generated models with an error equal to or less than 0.2. Again, DFP is better in six functions, while FP is in only two.

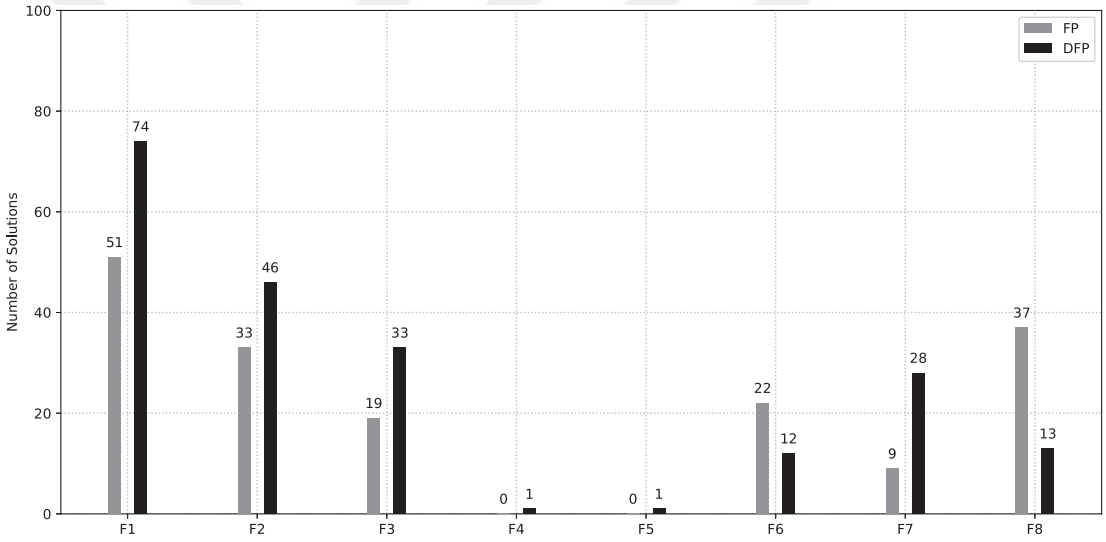


Figure 4.11. Number of exact generated solutions by FP/DFP

The three control parameters in DFP (α, β, and γ) can be optimized to increase the performance of the optimization process. While performing the pre-experiments, the difference in the error values for most cases was between 0 and 2. Taking into consideration the condition $\alpha < \beta < \gamma$, several tests were performed using different controlling values. Before starting the experiments, 0.1, 0.75, and 1.5 were chosen as initial values to α, β, and γ respectively. Tables 4.7, 4.8, and 4.9 show that the most successful results were obtained when $\alpha = 0.1$, $\beta = 0.5$, and $\gamma = 1.0$.

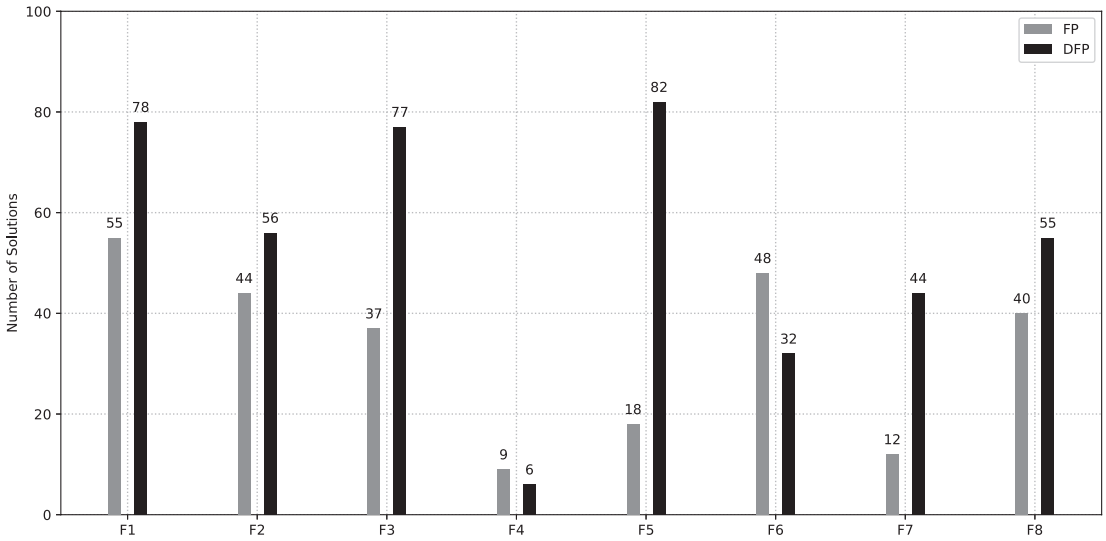


Figure 4.12. Number of the generated solutions with an error $\leqslant 0.2$ by FP/DFP

Table 4.7. DFP results with different α parameter values ($\beta : 0.75, \gamma : 1.5$)

α	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
0.05	**0.13**	0.24	**0.13**	0.46	0.21	0.30	0.24	0.31
0.10	**0.13**	**0.17**	0.17	0.48	**0.19**	**0.29**	0.23	0.28
0.15	0.17	0.25	0.14	**0.44**	0.20	0.35	0.27	0.32
0.20	0.15	0.28	**0.13**	0.46	0.21	0.37	0.27	0.41
0.25	0.14	0.24	0.16	0.51	0.21	0.32	**0.22**	0.48
0.30	0.15	0.26	0.15	0.51	0.23	0.39	0.24	0.28
0.35	0.15	0.22	**0.13**	0.53	0.24	0.36	0.25	0.46
0.40	0.17	0.26	0.14	0.47	0.21	0.37	0.27	**0.23**
0.45	0.18	0.25	0.14	0.51	0.25	0.32	0.26	0.31
0.50	**0.13**	0.30	0.19	0.54	0.21	0.46	0.26	0.42

Table 4.8. DFP results with different β parameter values ($\alpha : 0.1, \gamma : 1.5$)

β	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
0.40	0.15	0.18	0.17	0.52	0.21	0.31	0.24	0.52
0.50	0.16	**0.17**	**0.13**	0.48	**0.19**	0.28	**0.19**	**0.28**
0.60	0.14	0.19	0.16	0.49	0.20	**0.23**	0.23	0.37
0.70	0.10	0.23	0.14	0.50	0.21	0.26	0.27	0.44
0.80	0.13	0.21	**0.13**	0.48	0.20	0.30	0.21	0.36
0.90	0.12	0.30	0.14	0.43	**0.19**	0.33	0.21	0.31
1.00	0.16	0.24	0.14	0.52	**0.19**	0.24	0.26	0.35
1.10	0.14	0.22	0.15	**0.42**	**0.19**	0.26	0.21	0.34
1.20	**0.08**	0.23	0.16	0.43	0.20	0.32	0.26	0.30
1.30	0.11	0.21	**0.13**	0.46	**0.19**	0.31	0.27	0.45

Table 4.9. DFP results with different γ parameter values ($\alpha : 0.1, \beta : 0.75$)

γ	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
0.90	0.19	0.21	0.16	0.46	**0.19**	**0.27**	0.22	0.44
1.00	**0.10**	0.23	**0.13**	**0.45**	**0.19**	0.37	0.24	0.34
1.10	0.13	0.27	0.15	0.47	0.20	0.32	0.26	0.54
1.20	0.12	0.22	0.17	0.49	**0.19**	0.32	**0.19**	**0.31**
1.30	0.12	0.25	0.16	0.47	0.21	0.29	0.25	0.54
1.40	0.14	**0.19**	0.16	0.50	0.24	0.28	0.26	0.41
1.50	0.12	0.24	0.16	0.47	0.21	0.32	0.27	0.50
1.60	0.13	0.27	0.16	0.48	0.20	0.32	0.24	0.50
1.70	0.13	0.26	0.15	**0.45**	0.20	**0.27**	0.20	0.36
1.80	0.19	0.26	**0.13**	0.49	0.23	0.36	0.27	0.41

4.2. Box-Jenkins Gas Furnace Time Series Modeling and Forcasting

A time-series dataset is a sequence of data values that are tracked and recorded after performing experimental measurements within a system in a specific period (Box & Jenkins, 1976; Box et al., 2015). The statistical data are used in analyzing the system that generated this data to understand how it works, improve it, or even predict the outputs. In this sub-section, FP and DFP are used in modeling and forecasting the system of time series. Box-Jenkins gas furnace time series dataset was obtained by George Box and Gwilym Jenkins in 1970 (Box & Jenkins, 1976; Box et al., 2015). This dataset has 296 pairs of data that represent the amount of carbon dioxide-CO_2 (Y_t) concentration produced by burning gas in a furnace at a specific rate (U_t). In this experimental test, U_t and Y_{t-1} are used as inputs to the system that outputs Y_t.

Table 4.10 shows the different parameters used in FP and DFP. In the scope of this work, four different experiments were performed using two different searching sets and maximum depth values. Since the maximum number of nodes in a solution tree (which has the same structure as the binary tree) is calculated by $2^h - 1$, the number of the nodes is exponentially increased with the height of the tree. This helped us define two different values of the maximum tree depth. When the maximum height is less than 6, the solution tree can contain at most 31 nodes, which will generate only simple mathematical expressions. Therefore, the first value equals the initial maximum depth in experiments 1 and 3. On the other hand, the second value is equal to 17 in

Table 4.10. FP and DFP parameters

Parameter	Experiment	Value
Population Size	1, 2, 3, 4	50
Evaluations Number	1, 2, 3, 4	1,000,000
α, β, γ	1, 2, 3, 4	$10^{-4}, 10^{-2}, 10^{-1}$ respectively
Functional nodes	1, 2	+, −, ×, ÷ (protected ver.), pow
	3, 4	+, −, ×, ÷ (protected ver.), sin, cos, rlog, exp, pow
Terminal nodes	1, 2, 3, 4	$U_t, Y_{t-1}, C \in [-1, 1]$
Initial maximum depth	1, 2, 3, 4	6
Maximum depth	1, 3	6
	2, 4	17

experiments 2 and 4. This value allows trees to contain up to 131071 nodes which are more than enough to generate complex solutions. The terminal set in all experiments contains the two input variables $U_t(x_0), Y_{t-1}(x_1)$, and C which is a random float number between -1 and 1.

Table 4.11. FP/DFP results of forecasting Box-Jenkins-GFTS model - Exp 1

		MSE*	STD.*	Best MSE*	Worst MSE*
Training Set	FP	0.00069363	**0.00010782**	0.00048284	**0.00084566**
	DFP	**0.00062661**	0.00012619	**0.00039488**	0.00086772
Testing Set	FP	0.00220236	0.00110347	**0.00102831**	0.00468918
	DFP	**0.00196375**	**0.00087678**	0.00111447	**0.00420571**

* Lower is better

The dataset used for all these experiments is split into two subsets, the training set with 200 data pairs, and the testing set with 90 data pairs (Öztürk, 2011). The testing set represents 30% of the total data pairs used, which is suitable for evaluating the performance of the method. Dataset values are listed in Appendix A. In all experiments, the mean square error (MSE) in Eq. 4.2 is used as an objective function,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}((y_i - Y_i)^2) \quad , \quad \forall i \in samples \tag{4.2}$$

where Y_i is the i^{th} records' prediction, y_i is the i^{th} truth value.

```
y=(((x1)pow(((x0)+(x1))pow((x1)+((x0)+(x1)))))pow(((x0)+(x0))pow(-0.14)))
```

Figure 4.13. FP-best generated model - Exp 1

```
y=(((((x0)+(x1))pow((0.5)/(x1)))-((1.0)*(x0)))+((0.03)*((((x0)+(x1))-(x0))-(x0))))
```

Figure 4.14. DFP-best generated model - Exp 1

Table 4.11 shows the results of FP and FP in the first experiment which shows close results for both methods. The reason behind these close results is the limitation of the maximum tree depth which limits the trees from growing and producing complex models (Figure 4.13 and Figure 4.14). The predicted graphs of the generated models are shown in Figure 4.15.

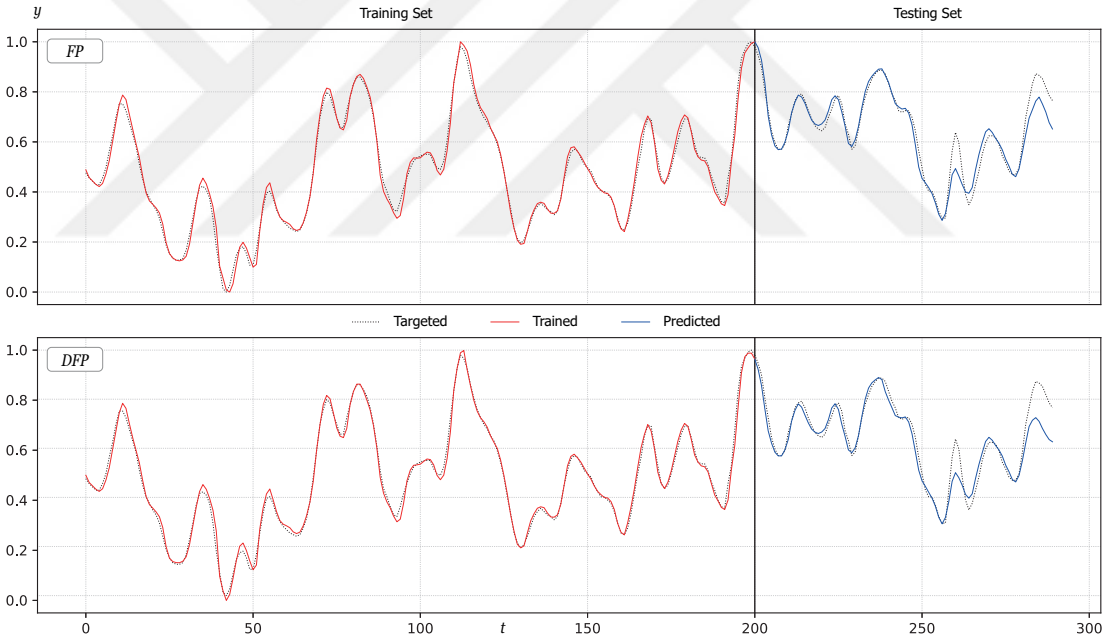


Figure 4.15. FP/DFP-best generated graph of Box-Jenkins-GFTS model (Exp. 1)

In the second experiment, DFP performed well as the results in Table 4.12 show. Both MSE mean value and the standard deviation were far better than the results of FP. By taking a look at the generated models in Figure 4.16 and Figure 4.17, DFP was able to generate a very complex model with only a 0.00036475 MSE value. The predicted graphs of experiment-2 are shown in Figure 4.18.

Table 4.12. FP/DFP results of forecasting Box-Jenkins-GFTS model - Exp 2

		MSE*	STD.*	Best MSE*	Worst MSE*
Training Set	FP	0.00057583	0.00012647	0.00040541	0.00095067
	DFP	**0.00042820**	**0.00003565**	**0.00034540**	**0.00047759**
Testing Set	FP	0.00629257	0.02079564	**0.00113152**	0.11816179
	DFP	**0.00216890**	**0.00057936**	0.00136828	**0.00371233**

* Lower is better

```
y=(((((-0.09)*((0.71)+(0.42)))-((-0.49)/(x1)))-((((x1)+(x0))pow(((-0.09)*(((0.71)+((-0.49
)/(x1)))pow((x1)pow(-0.76))))-((-0.49)/(x1))))+(((0.96)pow((x0)pow(((x1)pow(-0.76))*(
(-0.49)/(x1)))))/(-0.48))))-(((((-0.09)*(((0.71)+(0.42))pow((x1)pow(-0.76))))-((-0.49)/(x
1)))+((((x1)-(0.39))-(-0.42))/(((-0.36)+(-0.48))-(x1))))/((((x0)*(x0))*((x0)*(x1)))+(((0.71
)+(0.42))pow((x1)pow(-0.76))))))
```

Figure 4.16. FP-best generated model - Exp 2

```
y=(((x1)-((-0.01)*((((((1.41)/(x1))-((-0.77)+(x1)))pow((-0.57)*((-0.77)+(((((((x1)+(x0))/((
1.41)pow(x1)))*(x0))*((x0)*(((((x1)+(x1))*(x1))*(x1))+(x0))))+((-0.77)+(((-0.77)+(((1.41)/
((x1)-(x0)))*((0.55)pow(x0))))*(x1))))+(((((((-0.77)+((x1)*(x1)))*(((x1)+(x1))-((x1)+(x0)))
)+(((x1)*((x1)+(x0)))*(x0)))+(x1))-((-4.51)-(((1.41)pow(((x1)+(x0))+((1.0)*(x0))))+((((1.4
1)-(x0))*((x1)-(x0)))+((-0.77)+(x1))))))*(((((1.41)*(x1))*((1.0)*(x0)))-((0.94)-(x0)))*(((0.5
5)-((-0.77)+((x1)*(x1))))-(x0)))))))*((((1.41)/(x1))-(x0))-((((1.41)/(x1))*(((x0)+((-0.77)+((
(x1)-((-0.01)*((1.41)/(x1))))*((x1)*(x1)))))-(x1)))+((2.38)*(((x0)+((-0.77)+(((x1)+((((0.55)-
(x1))+((1.0)*(x0)))*((x0)/(x1))))*((x1)+(((1.0)/((1.41)pow(x1)))*((1.0)*(x0)))))))-((0.55)-(
x1)))))))+((((x0)*((x1)*(((x1)+((((-0.01)*(((x1)-(x1))-((1.0)*(x0))))-(x1))*(x1)))*((x1)*(x1)))
))pow(((((((x1)+(x0))*(x1))*(x1))+((-0.77)+(x1)))+(((x1)-(((-0.01)*((1.41)pow(((1.41)pow(
x1))+((1.41)pow(x1)))))*((((((x1)+(x1))+((x1)-(x0)))+((0.55)pow(x0)))*((1.41)pow(x1)))-((
(((x1)+(x0))/((1.41)pow(x1)))+(((x0)+(x0))pow(x1)))*(((1.0)*(x0))*(((x1)+(x0))+((-0.77)
+(x1)))))))*(x0)))+((-0.77)+(((((1.41)pow((x1)+(x1)))*(x0))*(((1.41)pow(x1))+(x1)))*(x1))
)))*((x1)*((1.41)pow(((1.41)pow((1.5)-((-0.77)+((x0)*((x1)+((((x0)/(x1))*((0.55)pow(x0)))
*(((1.41)pow(x1))*(x0)))))))-(((1.0)*(((x1)+(x0))*(x0)))+((-0.77)+((((((1.41)pow(x1))*((0.
55)pow(x0)))+(((0.55)pow(x0))+(((x1)+(x0))+((1.0)*(x0)))))*(((((0.94)-(x0))-(x1))-(x0))
+((1.41)pow((0.55)pow(x0)))))*((-0.77)+((x1)*(x1))))))))))-((-0.01)+(((1.0)*(((x1)+(((1.
0)/((1.41)pow(x1)))*((1.0)*(x0))))*((1.0)*((1.0)*(x0)))))*((((1.41)/(x1))pow(((x1)*((((0.5
5)pow(x0))pow((((((-0.01)-(x0))-((1.0)*((0.34)*(x0))))-(x0))+((1.0)*(((0.55)pow((x0)+(x
0)))-(x0))))-((((x0)+((-0.77)+((x1)*(x1))))+(((-0.01)-(x0))+((x1)*(x1))))+((x0)+((-0.77)+((
x1)*(x1)))))))*(x1)))*(((((1.41)pow(((((0.55)pow(x0))*(((0.55)pow(x0))/(x1)))-(x0))*((1.4
1)pow(((1.41)pow((0.55)pow(x0)))pow(((1.41)pow(x1))+(x1))))))/((((1.0)pow((1.41)pow(
(x1)+((1.41)pow(x1)))))-(x1))-((x1)pow((1.0)+((2.0)+(x0))))))-((x1)*((((((1.41)pow((x1)*(x
1)))+((0.55)pow(x0)))+(x1))pow((-0.77)+(((x1)*((x1)*(x1)))*(x1))))*((((1.41)*(x1))+((0.94
)-(x0)))+(x1)))))/(x1))))+((1.0)*(x0)))*(((0.34)*(((1.0)/((1.41)pow(x1)))*((1.0)*(x0))))*((1
.0)*(((((0.94)-(((1.0)*(x0))+((-0.77)+((x1)*(x1)))))pow((((x1)+(x0))*((1.0)*((x0)*((x1)+(x
0)))))*(((((((0.55)pow(x0))*((0.55)pow(x0)))*(x1))pow(((x1)*((x1)+(x0)))pow(((-0.01)-(
x0))-((1.0)*(x0)))))-((x1)+(((x1)+(x0))*(x0))))-(x0))))*(x1))-(((1.0)-(((x1)-((-0.01)*((1.41)
pow(x1))))*(x1)))-(x0)))))))))
```

Figure 4.17. DFP-best generated model - Exp 2

Although the functional node-set has more expressions in experiment 3 than in experiment 1, the results of experiment 3 in Table 4.13 are still close to each other

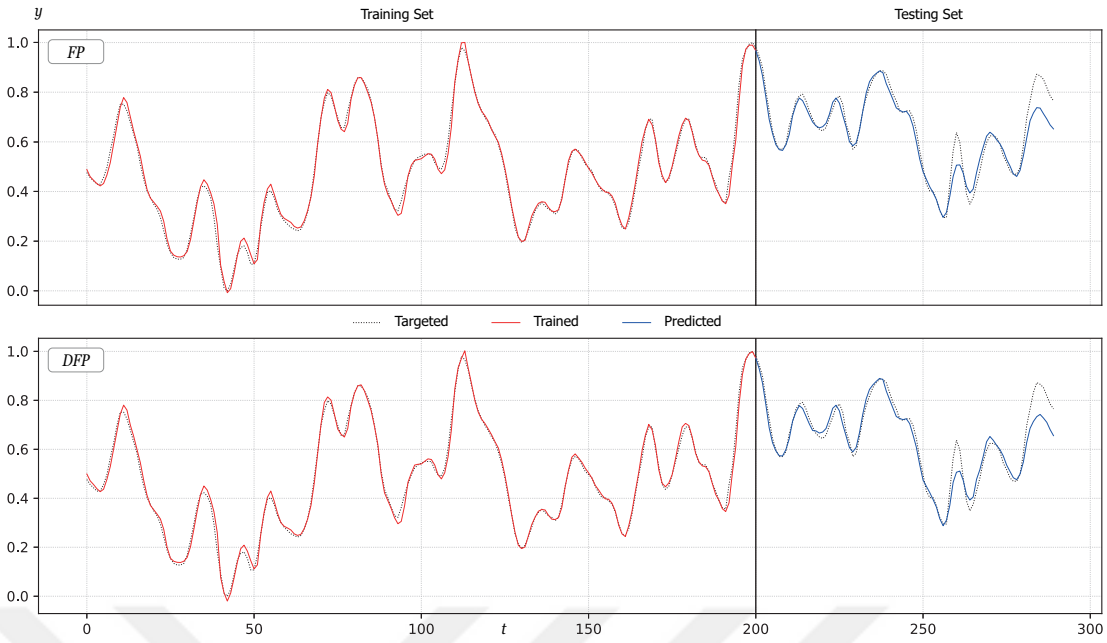


Figure 4.18. FP/DFP-best generated graph of Box-Jenkins-GFTS model (Exp. 2)

Table 4.13. FP/DFP results of forecasting Box-Jenkins-GFTS model - Exp 3

		MSE*	STD.*	Best MSE*	Worst MSE*
Training Set	FP	0.00072172	0.00016841	**0.00045719**	0.00106048
	DFP	**0.00067008**	**0.00012331**	0.00049263	**0.00084566**
Testing Set	FP	0.00237389	0.00164109	0.00108379	0.00867992
	DFP	**0.00218937**	**0.00105215**	**0.00105584**	**0.00519639**

* Lower is better

due to the limitation of the maximum depth. This shows the important role of these limits in generating models. The graphs of the best-generated models in Figure 4.19 and Figure 4.20 are shown in Figure 4.21.

```
y=(((rlog(exp(x1)))pow(((x0)+(x1))pow((exp(x1))pow(x1))))pow(((x0)+(x0))pow(((0.23)
*(-0.65))*((x0)+(x1)))))
```

Figure 4.19. FP-best generated model - Exp 3

```
y=(((((0.03)+(x1))pow((sin(x1))-((0.03)-(x0))))*(cos(((x0)-(x1))*((-0.0)-(x0)))))*(cos((co
s((-0.0)-(x0)))*(((x0)+(x1))*((-0.0)-(x0))))))
```

Figure 4.20. DFP-best generated model - Exp 3

The results of experiment 4 in Table 4.14 show that the DFP method has good

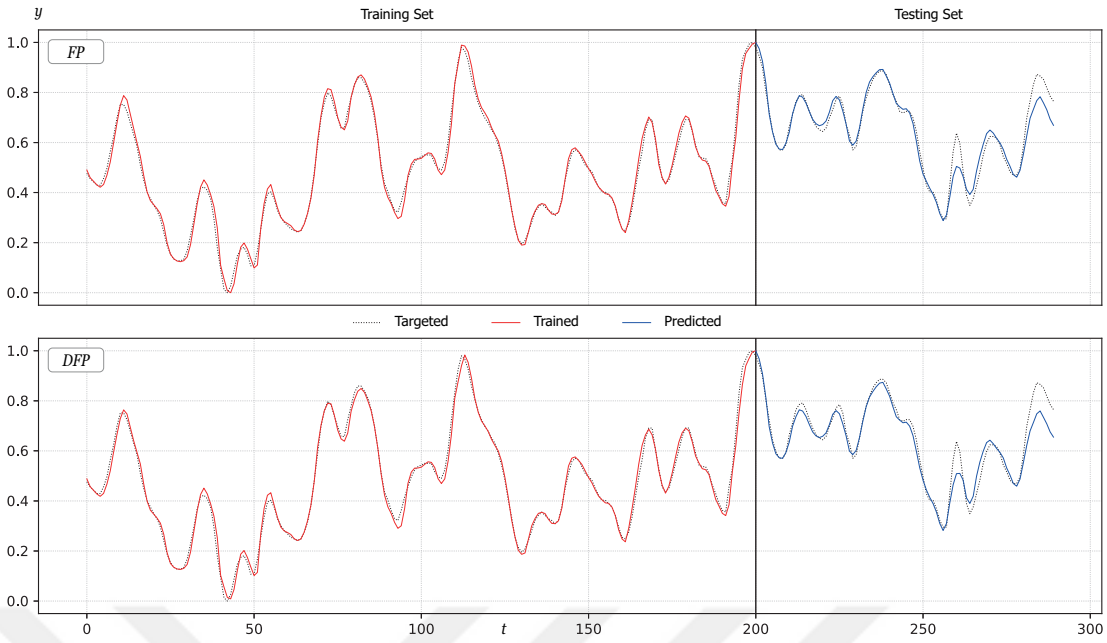


Figure 4.21. FP/DFP-best generated graph of Box-Jenkins-GFTS model (Exp. 3)

Table 4.14. FP/DFP results of forecasting Box-Jenkins-GFTS model - Exp 4

		MSE*	STD.*	Best MSE*	Worst MSE*
Training Set	FP	0.00054704	0.00037606	0.00038181	0.00255220
	DFP	**0.00042635**	**0.00005534**	**0.00036475**	**0.00063770**
Testing Set	FP	**0.00242676**	**0.00098444**	**0.00124539**	**0.00542701**
	DFP	0.00266292	0.00106410	0.00133588	0.00747385

* Lower is better

performance in the training set only where FP performed better in the testing set. Figure 4.23 and Figure 4.24 shows the generated models by FP and DFP respectively. The complexity of the generated models is too close in both FP and DFP. The predicted graphs of the generated models are shown in Figure 4.22 which shows that the generated graphs were too similar to the targeted ones in the training set with an observable difference in the testing set.

One more comparison was performed between the results of FP/DFP, and the results in (Öztürk, 2011) which used the same dataset to train different product-unit neural networks (PUNN) and additive units neural networks (ANN) using different meta-heuristic algorithms like ABC, PSO, and DE. Although standard deviation results of PSO-trained neural networks were better than the others, DFP showed the best MSE values in experiment 4 in the training set, and experiment 1 in the testing set.

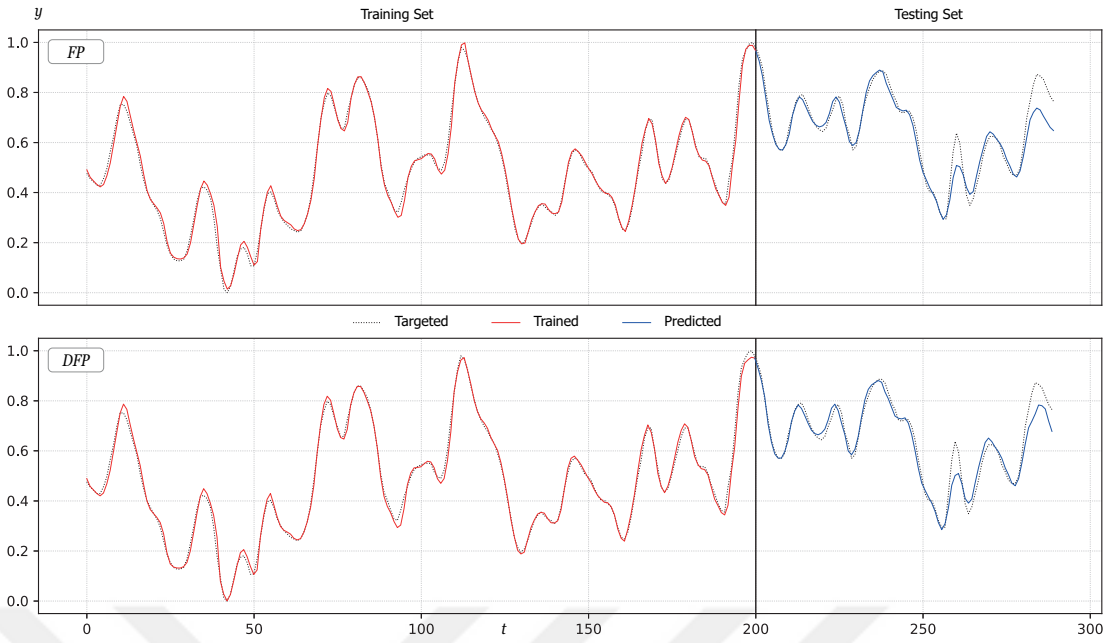


Figure 4.22. FP/DFP-best generated graph of Box-Jenkins-GFTS model (Exp. 4)

Table 4.15. Results comparison between FP, DFP, PUNN, and ANN

	MSE.		STD.	
Method	Train	Test	Train	Test
PSO-trained PUNN (2-6-1)	0.000436	0.002586	0.000025	**0.000142**
PSO-trained ANN (2-6-1)	0.010611	0.015363	0.001932	**0.000142**
DE-trained PUNN (2-6-1)	0.000438	0.002650	**0.000015**	0.000150
DE-trained ANN (2-6-1)	0.007835	0.011906	0.001395	0.000150
ABC-trained PUNN (2-6-1)	0.000637	0.003589	0.000072	0.000711
ABC-trained ANN (2-6-1)	0.006626	0.010572	0.000029	0.000711
FP Exp. 1	0.000694	0.002202	0.000108	0.001103
FP Exp. 2	0.000576	0.006293	0.000126	0.020796
FP Exp. 3	0.000722	0.002374	0.000168	0.001641
FP Exp. 4	0.000547	0.002427	0.000376	0.000984
DFP Exp. 1	0.000627	**0.001964**	0.000126	0.000877
DFP Exp. 2	0.000428	0.002169	0.000036	0.000579
DFP Exp. 3	0.000670	0.002189	0.000123	0.001052
DFP Exp. 4	**0.000426**	0.002663	0.000055	0.001064

y=(((sin(0.51))pow(((sin(-0.9))/(rlog(0.74)))pow(((exp((0.42)pow(((0.42)pow((x0)+(x1))+(x1))))((x0)+(x1)))*((x1)/(x0)))))*(((exp(rlog(((x0)+(x1))-(x0))))+(cos((x0)+(cos(cos(x1))))))pow((exp((-0.34)*(rlog(0.18))))*(((0.42)pow((x0)+(x1)))pow((sin(0.51))pow(((sin(-0.9))/(rlog(0.74)))pow((((exp(rlog(((x0)+(x1))-(x0))))+(cos((x0)+(cos(cos(x1))))))*((0.42)+(sin(0.51))))*((cos((x0)+(cos(cos(x1)))))/(x0)))))))))*

Figure 4.23. FP-best generated model - Exp 4

```
y=(((((x1)*(cos(x0)))-((((x0)-(((0.99)-(x1))/((0.99)-(x1))))*(exp(rlog(((x1)*(sin((x1)*(cos(
((x1)*(x1))*((x1)/(cos(x0))))))))-((((x0)-((cos(((0.99)-(x1))-((x1)/(cos(x0)))))-(x1)))*((((x1
)*(exp((x1)*(x0))))/((x1)/(x1)))*(x0)))+((-0.17)*(((cos(cos((0.99)-((x1)*(x0)))))pow(((x1)*
(exp((x1)*(x0))))/((x1)/(x1))))*(((x0)+((0.99)-(x1)))+(((cos((0.99)-(x1)))pow(x1))-(x1)))))
)))))+((-0.17)*(((sin(sin(sin(x1))))/((0.99)-((0.99)-(x1))))*((x0)+((cos(((x0)+(((x1)/(x1))*
((0.99)-(x1))))pow((exp(((x0)/(exp((x1)*(x0))))*(x0)))/(exp((x1)*(x0))))))*((x0)/(exp((x1
)*(x0))))))))))-(x0))+((x0)/(exp((x1)*(x0)))))
```

Figure 4.24. DFP-best generated model - Exp 4

5. CONCLUSION AND RECOMMENDATIONS

5.1. Conclusion

Symbolic regression or automatic programming methods are ML techniques that generate programs automatically to solve a specific problem. Automatic programming methods can recognize the pattern of the data without using any pre-defined structure. Therefore, automatic programming methods are used in solving complex data-driven problems. In the scope of this thesis, a new automatic programming method is introduced which is firefly programming (FP). FP is developed based on the firefly algorithm which is an evolutionary swarm-based algorithm introduced by Yang in 2008 and inspired by the behavior of fireflies in nature (Yang, 2008, 2009, 2010, 2017). To increase the efficiency of FP, an improved version is introduced called difference-based firefly programming (DFP) which depends on the difference between solutions.

While FP used sharing operation to exchange nodes between two trees, DFP used two operations to perform the attraction process according to the difference between trees. The first one "Sharing operation", takes an instance of a randomly selected subtree and glues it to another tree. The other operation "substitution" changes the value of one tree's node with another value without changing the structure. It is impossible to perform the attraction process in DFP without using the three control parameters: α, β, and γ. The difference between FP and DFP is not limited to the application of the attraction process, another operation was added also which reduces the complexity of the solution trees by simplifying its subtrees and combining them into one equivalent node.

To evaluate the performance of FP and DFP, two different experimental tests were performed. In the first one, FP and DFP were used to solve a set of symbolic regression benchmark problems that varied between polynomial, trigonometric, root, and multi-variable functions. The results of FP and DFP compared to the results of GP (Uy et al., 2013) and ABCP (Gorkemli & Karaboga, 2015) (Table 4.4) showed the superiority of DFP over the other methods as it gave the lowest mean error values in

four of eight problems: F1, F2, F7, and F8. DFP also shared the lowest mean error values with GP in two problems: F3 and F5. In F4 and F6, the performance of FP and DFP was not good as they both failed in giving good results. Generally, FP showed an average performance compared to GP, ABCP, and DFP. The same experiment compared the performance of the FP and DFP, where Table 4.5 and Table 4.6 showed that DFP gave better results than FP using different population size values. It also confirmed that DFP needed fewer evaluations to minimize error values as the graphs in Figure 4.10 show.

In the second experiment, FP and DFP were used in forecasting and modeling the Box-Jenkins gas furnace time series dataset. The experiment tested 2 different maximum depth limits: 6 and 17, where the experiments with a maximum depth of 17 gave better results. It also tested two different sets of the functional space sets, the first one contained "+, −, ×, ÷ (protected ver.), pow" and the second set added "sin, cos, rlog, exp" to the first set. The difference in the defined functional set did not impact the results of FP and DFP as shown in tables 4.11, 4.12, 4.13, and 4.14. Comparing the results of FP and DFP showed that DFP was better than FP in all cases. The results of FP and DFP were compared also to the results of different neural networks trained using evolutionary algorithms such as ABC, PSO, and DE (Öztürk, 2011). The results of DFP in the 2^{nd} and 4^{th} experiment were better than the results of neural networks in the training set, and the results of almost all experiments were better also in the testing set as Table 4.15 shows.

In general, FP and DFP performed well in the previously mentioned experiments with a remarkable superiority in performance for DFP. Nevertheless, the good performance was accompanied by an increase in time as well as in most evolution-based automatic programming methods (Smits & Kotanchek, 2005). Time complexity in such methods depends directly on the value of population size (n). For instance, the pseudo-code of the firefly algorithm in Alg. 1 contains two nested `for loops` in lines 5 and 6. This means that the firefly algorithm has a time complexity of $O(n^2)$ which is the same for FP and DFP. This time complexity is common among the automatic programming methods since most of them use evolutionary principles. In other words, in swarm-based methods, the time needed to improve the best model is multiplied by n since it improves n models before selecting the best one.

5.2. Recommendations

For future works and developments, it is recommended to focus on reducing the total execution time of FP. This can be achieved by developing a mini-version of the firefly algorithm that uses a minimal number of fireflies to complete the optimization process. Implementing additional operations and testing different structures to represent solutions can be useful to obtain better results. It is important to use firefly programming in solving different machine learning task problems including classifications, clustering, and more.

REFERENCES

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 acm sigmod international conference on management of data* (pp. 207–216).

Aliwi, M., Aslan, S., & Demirci, S. (2020a). Firefly programming for symbolic regression problems. In *2020 28th signal processing and communications applications conference (siu)* (pp. 1–4).

Aliwi, M., Aslan, S., & Demirci, S. (2020b). Solving uav localization problem with firefly algorithm. In *2020 28th signal processing and communications applications conference (siu)* (pp. 1–4).

Alpaydin, E. (2020). *Introduction to machine learning*. USA: MIT press.

Altenberg, L., et al. (1994). The evolution of evolvability in genetic programming. *Advances in genetic programming*, *3*, 47–74.

Arslan, S. (2021). Solving the problem of time series prediction using immune plasma programming. *European Journal of Science and Technology*(29), 219–224.

Arslan, S., & Ozturk, C. (2019a). Feature selection for classification with artificial bee colony programming. In *Swarm intelligence-recent advances, new perspectives and applications.* IntechOpen.

Arslan, S., & Ozturk, C. (2019b). Multi hive artificial bee colony programming for high dimensional symbolic regression with feature selection. *Applied Soft Computing*, *78*, 515–527.

Aslan, S., & Demirci, S. (2020). Immune plasma algorithm: A novel meta-heuristic for optimization problems. *IEEE Access*, *8*, 220227–220245.

Beadle, L., & Johnson, C. G. (2008). Semantically driven crossover in genetic programming. In *2008 ieee congress on evolutionary computation (ieee world congress on computational intelligence)* (pp. 111–116).

Billard, L., & Diday, E. (2002). Symbolic regression analysis. In *Classification, clustering, and data analysis* (pp. 281–288). Springer.

Box, G. E., & Jenkins, G. M. (1976). Time series analysis. forecasting and control. *Holden-Day Series in Time Series Analysis*.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Brameier, M. (2004). *On linear genetic programming* (Unpublished doctoral dissertation). Citeseer.

Branham, M. A., Leschen, R., Beutel, R., & Lawrence, J. (2010). Lampyridae latreille, 1817. *Morphology and Systematics*.

Burkov, A. (2019). *The hundred-page machine learning book* (Vol. 1). Andriy Burkov Quebec City, QC, Canada.

Contreras-Cruz, M. A., Ayala-Ramirez, V., & Hernandez-Belmonte, U. H. (2015). Mobile robot path planning using artificial bee colony and evolutionary programming. *Applied Soft Computing*, *30*, 319–328.

De Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach* (1st ed.). Springer Science & Business Media.

De Castro, L. N., & Von Zuben, F. J. (2000). The clonal selection algorithm with engineering applications. In *Proceedings of gecco* (Vol. 2000, pp. 36–39).

De Falco, I., Tarantino, E., Cioppa, A. D., & Fontanella, F. (2006). An innovative approach to genetic programming—based clustering. In *Applied soft computing technologies: The challenge of complexity* (pp. 55–64). Springer.

Dimopoulos, C., & Mort, N. (2001). A hierarchical clustering methodology based on genetic programming for the solution of simple cell-formation problems. *International Journal of Production Research*, *39*(1), 1–19.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(1), 29–41.

Eryurek, E., Gilad, U., Lakshmanan, V., Kibunguchy-Grant, A., & Ashdown, J. (2021). *Data governance: The definitive guide*. " O'Reilly Media, Inc.".

Espejo, P. G., Ventura, S., & Herrera, F. (2009). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(2), 121–144.

Fallah-Mehdipour, E., Haddad, O. B., & Mariño, M. (2013). Prediction and simulation of monthly groundwater levels by genetic programming. *Journal of Hydro-Environment Research*, *7*(4), 253–260.

Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. *arXiv preprint cs/0102027*.

Fisher, L. (2009). *The perfect swarm: The science of complexity in everyday life*. Basic Books.

Gao, M.-L., He, X.-H., Luo, D.-S., Jiang, J., & Teng, Q.-Z. (2013). Object tracking using firefly algorithm. *IET Computer Vision*, *7*(4), 227–237.

Gharrad, H., Jabeur, N., Yasar, A. U.-H., Galland, S., & Mbarki, M. (2021). A five-step drone collaborative planning approach for the management of distributed spatial events and vehicle notification using multi-agent systems and firefly algorithms. *Computer Networks*, *198*, 108282.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gorkemli, B., & Karaboga, D. (2015). *Developing new artificial bee colony programming (abcp) methods and symbolic regression applications* (PhD dissertation). Erciyes University.

Gorkemli, B., & Karaboga, D. (2019). A quick semantic artificial bee colony programming (qsabcp) for symbolic regression. *Information Sciences*, *502*, 346–362.

Harries, K., Smith, P., et al. (1997). Exploring alternative operators and search strategies in genetic programming. *Genetic Programming*, *97*, 147–155.

Hengpraprohm, S., & Chongstitvatana, P. (2001). Selective crossover in genetic programming. *Population*, *400*, 500.

Henrio, J., Deligne, T., Nakashima, T., & Watanabe, T. (2019). Route planning for multiple surveillance autonomous drones using a discrete firefly algorithm and a bayesian optimization method. *Artificial Life and Robotics*, *24*(1), 100–105.

HimaBindu, G., Kumar, C. R., Hemanand, C., & Krishna, N. R. (2020). Hybrid clustering algorithm to process big data using firefly optimization mechanism. *Materials Today: Proceedings*.

Hoai, N. X., McKay, R. I., Essam, D., & Chau, R. (2002). Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. In *Proceedings of the 2002 congress on evolutionary computation. cec'02 (cat. no. 02th8600)* (Vol. 2, pp. 1326–1331).

Ito, T., Iba, H., & Sato, S. (1998). Depth-dependent crossover for genetic programming. In *1998 ieee international conference on evolutionary computation proceedings. ieee world congress on computational intelligence (cat. no. 98th8360)* (pp. 775–780).

Johnson, C. G. (2002). Deriving genetic programming fitness properties by static analysis. In *Genetic programming: 5th european conference* (pp. 299–308). Berlin, Germany: Springer.

Johnson, C. G. (2003). Artificial immune system programming for symbolic regression. In *European conference on genetic programming* (pp. 345–353).

Johnson, C. G. (2009). Genetic programming crossover: Does it cross over? In *European conference on genetic programming* (pp. 97–108). Berlin, Heidelberg.

Kaboudan, M. A. (2000). Genetic programming prediction of stock prices. *Computational Economics*, *16*(3), 207–236.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization..

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, *39*(3), 459–471.

Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (abc) algorithm. *Applied soft computing*, *8*(1), 687–697.

Karaboga, D., Ozturk, C., Karaboga, N., & Gorkemli, B. (2012). Artificial bee colony programming for symbolic regression. *Information Sciences*, *209*, 1–15.

Keber, C., & Schuster, M. G. (2002). Option valuation with generalized ant programming. In *Proceedings of the 4th annual conference on genetic and evolutionary computation* (pp. 74–81).

Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In *European conference on genetic programming* (pp. 70–82). Berlin, Heidelberg.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection* (Vol. 1). USA: MIT Press.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, *4*(2), 87–112.

Koza, J. R., Andre, D., Bennett, F. H., & Keane, M. A. (1999). *Genetic programming iii: Darwinian invention and problem solving* (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kuhn, M., Johnson, K., et al. (2013). *Applied predictive modeling* (Vol. 26). Springer.

Langdon, W. B. (2000). Size fair and homologous tree genetic programming crossovers. *Genetic programming and evolvable machines*, *1*(1/2), 95–119.

Langdon, W. B., & Poli, R. (2002). *Foundations of genetic programming* (1st ed.). Berlin, Heidelberg: Springer Science & Business Media.

Lewis, S. M., & Cratsley, C. K. (2008). Flash signal evolution, mate choice, and predation in fireflies. *Annu. Rev. Entomol.*, *53*, 293–321.

Li, X. (2006). *Self-emergence of structures in gene expression programming*. University of Illinois at Chicago.

Loveard, T., & Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Proceedings of the 2001 congress on evolutionary computation (ieee cat. no. 01th8546)* (Vol. 2, pp. 1070–1077).

Majeed, H., & Ryan, C. (2006). A less destructive, context-aware crossover operator for gp. In *European conference on genetic programming* (pp. 36–48).

Marie-Sainte, S. L., & Alalyani, N. (2020). Firefly algorithm based feature selection for arabic text classification. *Journal of King Saud University-Computer and Information Sciences*, *32*(3), 320–328.

Martin, G. J., Stanger-Hall, K. F., Branham, M. A., Da Silveira, L. F., Lower, S. E., Hall, D. W., ... Bybee, S. M. (2019). Higher-level phylogeny and reclassification of lampyridae (coleoptera: Elateroidea). *Insect Systematics and Diversity*, *3*(6), 11.

McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., & O'neill, M. (2010). Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, *11*(3), 365–396.

Miller, J. F., & Thomson, P. (2000). Cartesian genetic programming. *Proc. European Conference on Genetic Programming*, *1802*, 121–132.

Mitchell, T. M. (1997). *Machine learning*. New York, United States: McGraw-Hill Education.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. USA: MIT press.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., & Petersen, B. K. (2021). Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*.

Muni, D. P., Pal, N. R., & Das, J. (2006). Genetic programming for simultaneous

feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *36*(1), 106–117.

Musilek, P., Lau, A., Reformat, M., & Wyard-Scott, L. (2006). Immune programming. *Information Sciences*, *176*(8), 972–1002.

Ogutu, J. O., Schulz-Streeck, T., & Piepho, H.-P. (2012). Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions. In *Bmc proceedings* (Vol. 6, pp. 1–6).

Oltean, M., & Dumitrescu, D. (2021). Multi expression programming. Research Square.

Oltean, M., & Grosan, C. (2003). A comparison of several linear genetic programming techniques. *Complex Systems*, *14*(4), 285–314.

O'Reilly, U.-M., & Oppacher, F. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *International conference on parallel problem solving from nature* (pp. 397–406).

Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the first ieee conference on evolutionary computation. ieee world congress on computational intelligence* (pp. 148–153).

Poli, R., & Langdon, W. B. (1998). Genetic programming with one-point crossover. In *Soft computing in engineering design and manufacturing* (pp. 180–189). Springer.

Poli, R., Langdon, W. B., McPhee, N. F., & Koza, J. R. (2008). *A field guide to genetic programming*. United Kingdom: Lulu Press.

Roux, O., & Fonlupt, C. (2000). Ant programming: or how to use ants for automatic programming. In *Proceedings of ants* (Vol. 2000, pp. 121–129).

Rusell, S., & Norvig, P. (2003). Artificial intelligence: A modern approach. *Pretice Hall Series in Artificial Intelligence*, *1*, 649–789.

Ryan, C., Collins, J. J., & Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *European conference on genetic programming* (pp. 83–96).

Ryan, C., & O'Neill, M. (1998). Grammatical evolution: A steady state approach. *Late Breaking Papers, Genetic Programming*, *1998*, 180–185.

Santoso, L., Singh, B., Rajest, S., Regin, R., & Kadhim, K. (2020). A genetic programming approach to binary classification problem. *EAI Endorsed Transactions on Energy Web*, *8*(31), e11.

Schmidt, M. D., & Lipson, H. (2006). Co-evolving fitness predictors for accelerating evaluations and reducing sampling. *Genetic Programming Theory and Practice IV*, *5*.

Senthilnath, J., Omkar, S., & Mani, V. (2011). Clustering using firefly algorithm: performance study. *Swarm and Evolutionary Computation*, *1*(3), 164–171.

Serrano, L. (2021). *Grokking machine learning*. Simon and Schuster.

Shamoo, A. E., & Resnik, D. B. (2009). *Responsible conduct of research* (2nd ed.). Oxford University Press.

Shirakawa, S., Ogino, S., & Nagao, T. (2008). Dynamic ant programming for automatic construction of programs. *IEEJ transactions on electrical and electronic engineering*, *3*(5), 540–548.

Smits, G. F., & Kotanchek, M. (2005). Pareto-front exploitation in symbolic regression. In *Genetic programming theory and practice ii* (pp. 283–299). Springer.

Tackett, W. A. (1994). *Recombination, selection, and the genetic construction of computer programs* (Unpublished doctoral dissertation). University of Southern California Los Angeles.

Tackett, W. A., & Carmi, A. (1994). The unique implications of brood selection for genetic programming. In *Proceedings of the first ieee conference on evolutionary computation. ieee world congress on computational intelligence* (pp. 160–165).

Tian, M., Bo, Y., Chen, Z., Wu, P., & Yue, C. (2019). A new improved firefly clustering algorithm for smc-phd filter. *Applied Soft Computing*, *85*, 105840.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

Tomer, M., & Kumar, M. (2021). Multi-document extractive text summarization based on firefly algorithm. *Journal of King Saud University-Computer and Information Sciences*.

Uy, N. Q., Hoai, N. X., O'Neill, M., & McKay, B. (2010). The role of syntactic and semantic locality of crossover in genetic programming. In *International conference on parallel problem solving from nature* (pp. 533–542).

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., & Galván-López, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, *12*(2), 91–119.

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., & Phong, D. N. (2013). On the roles of semantic locality of crossover in genetic programming. *Information Sciences*, *235*, 195–213.

Vladislavleva, E. J., Smits, G. F., & Den Hertog, D. (2008). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, *13*(2), 333–349.

Yang, X.-S. (2008). *Nature-inspired metaheuristic algorithms*. United Kingdom: Luniver Press.

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms* (pp. 169–178).

Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International journal of bio-inspired computation*, *2*(2), 78–84.

Yang, X.-S. (2017). *Nature-inspired algorithms and applied optimization* (Vol. 744). USA: Springer.

Öztürk, C. (2011). *Training artificial neural networks with artificial bee colony algorithm* (PhD dissertation). Erciyes University.

APPENDICES

Appendix A. Box-Jenkins Gas Furnace Dataset

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
1	0.52144204	0.50335040	0.47650480
2	0.55045102	0.47650480	0.45637060
3	0.55657714	0.45637060	0.44294780
4	0.56882938	0.44294780	0.42952500
5	0.57243298	0.42952500	0.42952500
6	0.55207264	0.42952500	0.45637060
7	0.51225286	0.45637060	0.49663900
8	0.45693760	0.49663900	0.56375300
9	0.38342416	0.56375300	0.62415560
10	0.29928010	0.62415560	0.69798100
11	0.23333422	0.69798100	0.75167220
12	0.21549640	0.75167220	0.75167220
13	0.25477564	0.75167220	0.72482660
14	0.34270348	0.72482660	0.67784680
15	0.40378450	0.67784680	0.63086700
16	0.45459526	0.63086700	0.58388720
17	0.50522584	0.58388720	0.51006180
18	0.56774830	0.51006180	0.44965920
19	0.62828878	0.44965920	0.40267940
20	0.64540588	0.40267940	0.37583380
21	0.64702750	0.37583380	0.34898820
22	0.64991038	0.34898820	0.32885400
23	0.66720766	0.32885400	0.29529700
24	0.71693734	0.29529700	0.24160580
25	0.80918950	0.24160580	0.18791460
26	0.84540568	0.18791460	0.15435760
27	0.83783812	0.15435760	0.13422340
28	0.82558588	0.13422340	0.12751200
29	0.81945976	0.12751200	0.12751200
30	0.80774806	0.12751200	0.13422340
31	0.77909944	0.13422340	0.16778040
32	0.71729770	0.16778040	0.22818300
33	0.63171220	0.22818300	0.29529700
34	0.55423480	0.29529700	0.36912240
35	0.51009070	0.36912240	0.41610220
36	0.50522584	0.41610220	0.42281360
37	0.54900958	0.42281360	0.40939080
38	0.60558610	0.40939080	0.37583380

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
39	0.66234280	0.37583380	0.29529700
40	0.74324362	0.29529700	0.18120320
41	0.97045060	0.18120320	0.09395500
42	1.00000012	0.09395500	0.01341820
43	0.99603616	0.01341820	0.00000000
44	0.93675694	0.00000000	0.02684100
45	0.83693722	0.02684100	0.08724360
46	0.75693730	0.08724360	0.14764620
47	0.70810852	0.14764620	0.17449180
48	0.71261302	0.17449180	0.18120320
49	0.77909944	0.18120320	0.15435760
50	0.83261290	0.15435760	0.10737780
51	0.85387414	0.10737780	0.10737780
52	0.81639670	0.10737780	0.16778040
53	0.58576630	0.16778040	0.25502860
54	0.51135196	0.25502860	0.33556540
55	0.49099162	0.33556540	0.39596800
56	0.51891952	0.39596800	0.40267940
57	0.61027078	0.40267940	0.37583380
58	0.67297342	0.37583380	0.32885400
59	0.69585628	0.32885400	0.30200840
60	0.69747790	0.30200840	0.28187420
61	0.68973016	0.28187420	0.26845140
62	0.69135178	0.26845140	0.25502860
63	0.70973014	0.25502860	0.24831720
64	0.71585626	0.24831720	0.24160580
65	0.69783826	0.24160580	0.24831720
66	0.65387434	0.24831720	0.27516280
67	0.60108160	0.27516280	0.31543120
68	0.53531590	0.31543120	0.38254520
69	0.43891960	0.38254520	0.48321620
70	0.29477560	0.48321620	0.59059860
71	0.20991082	0.59059860	0.69798100
72	0.16522618	0.69798100	0.75838360
73	0.16054150	0.75838360	0.79865200
74	0.22702792	0.79865200	0.78522920
75	0.31928008	0.78522920	0.73824940
76	0.38666740	0.73824940	0.69798100
77	0.41171242	0.69798100	0.65771260
78	0.38540614	0.65771260	0.65771260
79	0.31639720	0.65771260	0.72482660
80	0.19819912	0.72482660	0.77851780
81	0.15153250	0.77851780	0.83220900
82	0.14864962	0.83220900	0.85905460
83	0.17477572	0.85905460	0.85905460

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
84	0.22378468	0.85905460	0.83892040
85	0.27297382	0.83892040	0.81207480
86	0.32234314	0.81207480	0.76509500
87	0.39495568	0.76509500	0.69798100
88	0.49657720	0.69798100	0.61073280
89	0.63135184	0.61073280	0.51006180
90	0.65927974	0.51006180	0.43623640
91	0.65693740	0.43623640	0.40267940
92	0.67063108	0.40267940	0.36241100
93	0.69423466	0.36241100	0.32885400
94	0.70522564	0.32885400	0.32214260
95	0.67927972	0.32214260	0.36241100
96	0.59657710	0.36241100	0.41610220
97	0.47495560	0.41610220	0.45637060
98	0.43279348	0.45637060	0.49663900
99	0.43747816	0.49663900	0.52348460
100	0.46180246	0.52348460	0.53690740
101	0.46973038	0.53690740	0.54361880
102	0.45567634	0.54361880	0.55033020
103	0.44342410	0.55033020	0.55033020
104	0.44810878	0.55033020	0.55033020
105	0.48810874	0.55033020	0.51677320
106	0.53513572	0.51677320	0.49663900
107	0.54882940	0.49663900	0.48992760
108	0.50774836	0.48992760	0.52348460
109	0.41315386	0.52348460	0.60402140
110	0.28414498	0.60402140	0.72482660
111	0.07946050	0.72482660	0.83220900
112	0.02198308	0.83220900	0.92616860
113	0.00000000	0.92616860	0.97985980
114	0.03711820	0.97985980	0.96643700
115	0.16684780	0.96643700	0.92616860
116	0.24684772	0.92616860	0.85905460
117	0.29459542	0.85905460	0.80536340
118	0.32540620	0.80536340	0.75838360
119	0.33153232	0.75838360	0.72482660
120	0.32991070	0.72482660	0.69798100
121	0.34522600	0.69798100	0.67784680
122	0.39135208	0.67784680	0.65100120
123	0.41441512	0.65100120	0.63086700
124	0.44054122	0.63086700	0.59059860
125	0.48937000	0.59059860	0.54361880
126	0.56198254	0.54361880	0.48321620
127	0.64090138	0.48321620	0.40267940
128	0.72090130	0.40267940	0.33556540

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
129	0.77891926	0.33556540	0.25502860
130	0.80396428	0.25502860	0.21476020
131	0.79261294	0.21476020	0.19462600
132	0.75693730	0.19462600	0.20804880
133	0.66828874	0.20804880	0.24160580
134	0.60612664	0.24160580	0.28187420
135	0.59333386	0.28187420	0.32214260
136	0.59333386	0.32214260	0.34227680
137	0.60324376	0.34227680	0.35569960
138	0.62396446	0.35569960	0.34227680
139	0.65153200	0.34227680	0.32885400
140	0.66828874	0.32885400	0.32214260
141	0.66378424	0.32214260	0.30871980
142	0.63171220	0.30871980	0.32214260
143	0.56126182	0.32214260	0.37583380
144	0.46036102	0.37583380	0.44965920
145	0.38973046	0.44965920	0.51006180
146	0.38072146	0.51006180	0.55704160
147	0.41297368	0.55704160	0.57046440
148	0.45441508	0.57046440	0.56375300
149	0.48054118	0.56375300	0.53690740
150	0.50018080	0.53690740	0.51006180
151	0.51837898	0.51006180	0.49663900
152	0.54360418	0.49663900	0.48321620
153	0.58252306	0.48321620	0.44965920
154	0.59135188	0.44965920	0.42281360
155	0.59027080	0.42281360	0.40267940
156	0.59261314	0.40267940	0.40267940
157	0.59603656	0.40267940	0.38925660
158	0.61027078	0.38925660	0.37583380
159	0.65747794	0.37583380	0.34227680
160	0.73027066	0.34227680	0.29529700
161	0.75243280	0.29529700	0.25502860
162	0.73315354	0.25502860	0.24831720
163	0.62846896	0.24831720	0.27516280
164	0.52864924	0.27516280	0.33556540
165	0.44666734	0.33556540	0.41610220
166	0.36072148	0.41610220	0.49663900
167	0.29135218	0.49663900	0.56375300
168	0.26072158	0.56375300	0.65100120
169	0.27765850	0.65100120	0.69126960
170	0.36756832	0.69126960	0.69126960
171	0.49531594	0.69126960	0.60402140
172	0.58955008	0.60402140	0.53019600
173	0.60522574	0.53019600	0.45637060

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
174	0.57657712	0.45637060	0.43623640
175	0.50900962	0.43623640	0.44965920
176	0.43351420	0.44965920	0.49663900
177	0.36378454	0.49663900	0.55033020
178	0.30072154	0.55033020	0.60402140
179	0.26991076	0.60402140	0.65771260
180	0.27621706	0.65771260	0.69126960
181	0.33207286	0.69126960	0.69126960
182	0.42882952	0.69126960	0.64428980
183	0.50072134	0.64428980	0.59059860
184	0.50450512	0.59059860	0.54361880
185	0.48937000	0.54361880	0.53690740
186	0.48955018	0.53690740	0.53690740
187	0.52702762	0.53690740	0.51006180
188	0.58955008	0.51006180	0.46308200
189	0.63027076	0.46308200	0.42952500
190	0.64396444	0.42952500	0.38925660
191	0.65477524	0.38925660	0.36241100
192	0.64468516	0.36241100	0.35569960
193	0.56432488	0.35569960	0.45637060
194	0.42882952	0.45637060	0.53019600
195	0.31657738	0.53019600	0.67113540
196	0.16270366	0.67113540	0.83220900
197	0.06090196	0.83220900	0.93288000
198	0.03910018	0.93288000	0.96643700
199	0.04378486	0.96643700	0.99328260
200	0.06955060	0.99328260	0.99999400
201	0.11946046	0.99999400	0.97985980
202	0.17603698	0.97985980	0.94630280
203	0.26216302	0.94630280	0.89932300
204	0.38684758	0.89932300	0.80536340
205	0.46468534	0.80536340	0.72482660
206	0.48504568	0.72482660	0.64428980
207	0.48036100	0.64428980	0.59731000
208	0.46504570	0.59731000	0.57046440
209	0.43964032	0.57046440	0.57046440
210	0.39315388	0.57046440	0.59059860
211	0.33243322	0.59059860	0.66442400
212	0.26540626	0.66442400	0.71140380
213	0.23009098	0.71140380	0.76509500
214	0.23315404	0.76509500	0.78522920
215	0.27765850	0.78522920	0.79194060
216	0.34288366	0.79194060	0.76509500
217	0.37513588	0.76509500	0.72482660
218	0.38450524	0.72482660	0.69126960

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
219	0.37675750	0.69126960	0.66442400
220	0.36090166	0.66442400	0.65100120
221	0.33657736	0.65100120	0.64428980
222	0.30216298	0.64428980	0.65771260
223	0.24684772	0.65771260	0.69798100
224	0.19603696	0.69798100	0.73153800
225	0.19765858	0.73153800	0.77180640
226	0.28234318	0.77180640	0.78522920
227	0.40144216	0.78522920	0.75167220
228	0.46054120	0.75167220	0.67113540
229	0.48810874	0.67113540	0.63086700
230	0.47279344	0.63086700	0.57046440
231	0.37765840	0.57046440	0.58388720
232	0.29369452	0.58388720	0.65100120
233	0.21459550	0.65100120	0.72482660
234	0.15459556	0.72482660	0.77851780
235	0.12378478	0.77851780	0.81878620
236	0.12468568	0.81878620	0.85234320
237	0.13603702	0.85234320	0.87247740
238	0.13765864	0.87247740	0.88590020
239	0.16612708	0.88590020	0.88590020
240	0.25477564	0.88590020	0.87247740
241	0.30378460	0.87247740	0.83220900
242	0.32396476	0.83220900	0.79194060
243	0.34558636	0.79194060	0.76509500
244	0.33315394	0.76509500	0.72482660
245	0.30072154	0.72482660	0.71811520
246	0.28702786	0.71811520	0.72482660
247	0.33153232	0.72482660	0.72482660
248	0.41819890	0.72482660	0.69798100
249	0.52270330	0.69798100	0.64428980
250	0.60864916	0.64428980	0.56375300
251	0.61711762	0.56375300	0.49663900
252	0.59837890	0.49663900	0.42952500
253	0.57964018	0.42952500	0.40267940
254	0.59801854	0.40267940	0.40267940
255	0.65927974	0.40267940	0.36912240
256	0.70973014	0.36912240	0.32214260
257	0.71441482	0.32214260	0.29529700
258	0.63783832	0.29529700	0.29529700
259	0.50774836	0.29529700	0.42952500
260	0.49387450	0.42952500	0.56375300
261	0.55819876	0.56375300	0.63757840
262	0.65549596	0.63757840	0.59731000
263	0.67531576	0.59731000	0.48321620

Table A.1. Box-Jenkins Gas Furnace Dataset

t	U_t	Y_{t-1}	Y_t
264	0.64540588	0.48321620	0.38925660
265	0.58432486	0.38925660	0.34898820
266	0.50612674	0.34898820	0.37583380
267	0.40684756	0.37583380	0.42952500
268	0.35459536	0.42952500	0.48321620
269	0.31873954	0.48321620	0.55033020
270	0.30396478	0.55033020	0.59731000
271	0.32216296	0.59731000	0.62415560
272	0.37315390	0.62415560	0.62415560
273	0.41297368	0.62415560	0.61744420
274	0.43964032	0.61744420	0.59059860
275	0.46090156	0.59059860	0.54361880
276	0.48342406	0.54361880	0.51677320
277	0.50774836	0.51677320	0.48321620
278	0.53459518	0.48321620	0.46979340
279	0.53982040	0.46979340	0.46979340
280	0.48937000	0.46979340	0.49663900
281	0.40054126	0.49663900	0.58388720
282	0.35261338	0.58388720	0.69798100
283	0.34090168	0.69798100	0.76509500
284	0.35603680	0.76509500	0.83220900
285	0.39423496	0.83220900	0.87247740
286	0.45261328	0.87247740	0.86576600
287	0.49549612	0.86576600	0.85234320
288	0.52612672	0.85234320	0.81878620
289	0.53495554	0.81878620	0.78522920
290	0.52450510	0.78522920	0.76509500

CURRICULUM VITAE

Mohamed ALIWI graduated in 2019 as a computer engineer from Engineering Faculty / Ondokuz Mayıs University. Started master program in 2019 at the Institute of Graduate Studies / Ondokuz Mayıs University. Speaks Arabic (native), English and Turkish. Worked as a developer with different organizations. (August, 2022).

Contact Information

ORCID NO : 0000-0002-9876-3430

Publications

1. M. Aliwi, S. Aslan and S. Demirci, "Firefly Programming For Symbolic Regression Problems," 2020 28th Signal Processing and Communications Applications Conference (SIU), 2020, pp. 1-4, doi: 10.1109/SIU49456.2020.9302201.

2. M. Aliwi, S. Aslan and S. Demirci, "Solving UAV Localization Problem with Firefly Algorithm," 2020 28th Signal Processing and Communications Applications Conference (SIU), 2020, pp. 1-4, doi: 10.1109/SIU49456.2020.9302366.