NOVA IMS Information Management School

MMAA

Mestrado em Métodos Analíticos Avançados

Master Program in Advanced Analytics

A STUDY OF GEOMETRIC SEMANTIC GENETIC PROGRAMMING WITH LINEAR SCALING

Berfin Sakallioglu

Dissertation presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Data Science

NOVA Information Management School
Instituto Superior de Estatística e Gestão da Informação

Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade NOVA de Lisboa

A STUDY OF GEOMETRIC SEMANTIC GENETIC PROGRAMMING WITH LINEAR SCALING

by

Berfin Sakallioglu

Dissertation presented as partial requirement for obtaining the Master’s degree in Data Science and Advanced Analytics, with a specialization in Data Science

Advisor:
Prof. Dr. Leonardo Vanneschi

February, 2023

A Study of Geometric Semantic Genetic Programming with Linear Scaling

Copyright © Berfin Sakallioglu, NOVA Information Management School, NOVA University Lisbon.
The NOVA Information Management School and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.12) [1].

To Ankara.

Acknowledgements

I would like to express my deepest gratitude to my esteemed professor, Leonardo Vanneschi, for his exceptional mentorship and constructive feedback throughout my academic journey. His remarkable intellect and dedication to the pursuit of knowledge have been a source of inspiration. I am deeply indebted to him for his patience, understanding, and kindness. His unwavering support and trust have given me strength to overcome challanges.

My heartfelt appreciation goes to every person in my beautiful big family with special thanks to my mom, dad and wonderful brother, Utku. Thank you for giving me life, your unconditional love, support, and your existence. I am lucky to have all of you.

And all the amazing people in my life, my friends, my people in Ankara, I am glad that I grew up with all of you. You have all played a significant role in shaping me. And to Lisbon, and the people who embraced and loved me in this city. Thank you for being a family to me in a place so far away from my home.

To all these people and everyone who directly or indirectly helped me with my thesis period, I am deeply greatful, this would have been impossible without you. And finally, the cumulative knowledge of humanity, thank you. Despite everything, ending up on this planet has been great!

"... a star among stars
and one of the smallest,
a gilded mote on blue velvet—
I mean this, our great earth."

(Nâzım Hikmet)

Abstract

Machine Learning (ML) is a scientific discipline that endeavors to enable computers to learn without the need for explicit programming. Evolutionary Algorithms (EAs), a subset of ML algorithms, mimic Darwin's Theory of Evolution by using natural selection mechanisms (i.e., survival of the fittest) to evolve a group of individuals (i.e., possible solutions to a given problem). Genetic Programming (GP) is the most recent type of EA and it evolves computer programs (i.e., individuals) to map a set of input data into known expected outputs. Geometric Semantic Genetic Programming (GSGP) extends this concept by allowing individuals to evolve and vary in the semantic space, where the output vectors are located, rather than being constrained by syntax-based structures. Linear Scaling (LS) is a method that was introduced to facilitate the task of GP of searching for the best function matching a set of known data. GSGP and LS have both, independently, shown the ability to outperform standard GP for symbolic regression. GSGP uses Geometric Semantic Operators (GSOs), different from the standard ones, without altering the fitness, while LS modifies the fitness without altering the genetic operators. To the best of our knowledge, there has been no prior utilization of the combined methodology of GSGP and LS for classification problems. Furthermore, despite the fact that they have been used together in one practical regression application, a methodological evaluation of the advantages and disadvantages of integrating these methods for regression or classification problems has never been performed. In this dissertation, a study of a system that integrates both GSGP and LS (GSGP-LS) is presented. The performance of the proposed method, GSGP-LS, was tested on six hand-tailored regression benchmarks, nine real-life regression problems and three real-life classification problems. The obtained results indicate that GSGP-LS outperforms GSGP in the majority of the cases, confirming the expected benefit of this integration. However, for some particularly hard regression datasets, GSGP-LS overfits training data, being outperformed by GSGP on unseen data. This contradicts the idea that LS is always beneficial for GP, warning the practitioners about its risk of overfitting in some specific cases.

Keywords: Genetic Programming, Geometric Semantic Genetic Programming, Linear Scaling, Evolutionary Algorithms, Machine Learning.

Resumo

A Aprendizagem Automática (AA) é uma disciplina científica que se esforça por permitir que os computadores aprendam sem a necessidade de programação explícita. Algoritmos Evolutivos (AE), um subconjunto de algoritmos de ML, mimetizam a Teoria da Evolução de Darwin, usando a seleção natural e mecanismos de "sobrevivência dos mais aptos"para evoluir um grupo de indivíduos (ou seja, possíveis soluções para um problema dado). A Programação Genética (PG) é um processo algorítmico que evolui programas de computador (ou indivíduos) para ligar características de entrada e saída. A Programação Genética em Geometria Semântica (PGGS) estende esse conceito permitindo que os indivíduos evoluam e variem no espaço semântico, onde os vetores de saída estão localizados, em vez de serem limitados por estruturas baseadas em sintaxe. A Escala Linear (EL) é um método introduzido para facilitar a tarefa da PG de procurar a melhor função que corresponda a um conjunto de dados conhecidos. Tanto a PGGS quanto a EL demonstraram, independentemente, a capacidade de superar a PG padrão para regressão simbólica. A PGGS usa Operadores Semânticos Geométricos (OSGs), diferentes dos padrões, sem alterar o fitness, enquanto a EL modifica o fitness sem alterar os operadores genéticos. Até onde sabemos, não houve utilização prévia da metodologia combinada de PGGS e EL para problemas de classificação. Além disso, apesar de terem sido usados juntos em uma aplicação prática de regressão, nunca foi realizada uma avaliação metodológica das vantagens e desvantagens da integração desses métodos para problemas de regressão ou classificação. Nesta dissertação, é apresentado um estudo de um sistema que integra tanto a PGGS quanto a EL (PGGS-EL). O desempenho do método proposto, PGGS-EL, foi testado em seis benchmarks de regressão personalizados, nove problemas de regressão da vida real e três problemas de classificação da vida real. Os resultados obtidos indicam que o PGGS-EL supera o PGGS na maioria dos casos, confirmando o benefício esperado desta integração. No entanto, para alguns conjuntos de dados de regressão particularmente difíceis, o PGGS-EL faz *overfit* aos dados de treino, obtendo piores resultados em comparação com PGGS em dados não vistos. Isso contradiz a ideia de que a EL é sempre benéfica para a PG, alertando os praticantes sobre o risco de *overfitting* em alguns casos específicos.

Palavras-chave: Programação Genética, Programação Genética em Geometria Semântica, Escala Linear, Algorítmos Evolutivos, Aprendizagem Automática

Contents

List of Figures

List of Tables

1 Introduction

Genetic Programming (GP) is a general-purpose method to automatically breed computer programs to solve a task [2]. In the last decade, the use of GP [2] to tackle symbolic regression problems has gained popularity, possibly because of some qualities of GP, such as its ability to deal with problems where little or no information is known about the data, its ability to evolve models that do not have a previously fixed mathematical shape, and to perform automatic feature selection while learning the model [3–5]. GP is traditionally employed to tackle symbolic regression problems using well-known loss measures, such as the root mean square error (RMSE), to quantify the fitness of the evolving solutions. Even though this approach is still very popular, it has a drawback: some solutions may receive a bad fitness value, promising though they might be. It is the case, for instance, of solutions that have a similar shape to the one of the target function, but with different slope and/or location in the Cartesian space. Linear scaling (LS) was thus introduced by Keijzer [6] to tackle this issue and improve the performance of GP on symbolic regression. LS modifies the fitness function rescaling each individual by using their slope and intercept, two constants that can be easily calculated with a cost that is linear in the size of the training set. In this way, the burden of searching for these two constants is removed from the evolution, leaving GP with the only task of searching for functions whose shape is most similar to that of the target function. Since its introduction, the benefit of LS was demonstrated on many theoretical benchmark functions [6] and real-life applications [7–10]. These studies indicate that LS does not only improve standard GP on training data, but can also bestow on GP a better generalization ability, often outperforming standard GP also on unseen data. However, Costelloe and Ryan [11] pointed out that LS may not always improve GP's generalization ability.

Ten years after the introduction of LS, Moraglio et al. [12] introduced Geometric Semantic GP (GSGP), a different variant of GP. GSGP uses specific genetic operators, called Geometric Semantic Operators (GSOs), instead of the traditional crossover and mutation of GP. Although acting directly on the syntax of the GP individuals, GSOs have an indirect known effect on their semantics, and have the important property of inducing a unimodal error surface for any supervised learning problem [12]. Several results were presented revealing the ability of GSGP to effectively fit training data [12, 13]. At the same time, GSGP was also shown to limit overfitting, often outperforming standard GP on unseen data for several real-life symbolic regression problems [14].

Given that LS works by redefining the fitness and GSGP works by redefining the genetic operators, which are in general two independent parts of the GP algorithm, it is natural to imagine a system that joins these two methods, possibly capturing the advantages of both GSGP and LS. Following this idea, in 2015 Vanneschi et al. [15] combined GSGP and LS, achieving outstanding results on a challenging application based on AIS (Automatic Identification System) for the prediction of the positions of vessels at sea. The success of that system in that particular application domain, together with the previous achievements of GSGP and LS used in isolation, may induce researchers to think that the integration of GSGP and LS is always beneficial. However, Costelloe's and Ryan's [11] observations made on standard GP sound like an important warning and call for a methodological study aimed at investigating the pros and cons of integrating GSGP and LS.

This dissertation presents a GSGP algorithm and an investigation of a system that employs GSOs to explore the search space, guided by the LS fitness function, in a manner similar to that of the approach proposed in [15]. The proposed system is referred to as GSGP-LS. Furthermore, this work employs the GSGP and LS methods to address classification problems where no prior utilization has been implemented.

The document is structured as follows. Chapter 2, Theoretical Background, provides the reader with sufficient knowledge to comprehend the presented work, including a general understanding of Machine Learning (ML), optimization, bioinspired and Evolutionary Algorithms (EAs), as well as LS. Chapter 3, Literature Review, briefly reviews previous works relevant to this study, including both semantics in GP and LS in GP. Chapter 4, Methodology, presents a detailed description of the GSGP-LS algorithm developed for this work. A discussion of the software implementing GSGP-LS has been organized and developed. In Chapter 5, the experimental setup is described, starting with the parameter settings, followed by presentation of the case studies, the obtained experimental results and a discussion of the results. Finally, Chapter 6 concludes the work with an overview containing the results of the work, limitations, and ideas for future research.

2 Theoretical Background

The objective of this chapter is to present algorithms, techniques and fundamental concepts used within this work. More specifically, Section 2.1 introduces the field of ML, Section 2.2 presents an overview of Optimization while Section 2.3 summarizes Bioinspired and EAs, including GP in Subsection 2.3.1 and GSGP in Subsection 2.3.2. Lastly, Section 2.4 expounds LS.

2.1 Machine Learning

"I PROPOSE to consider the question, 'Can machines think?' [16]

The question was posed by Alan Turing in 1950 [16]. Although the origins of ML can be traced back to the 1950s, the question "Can machines think?" sparked a flurry of discoveries in computer science and human history. This question is followed by another: "How can we teach computers to learn like intelligent beings? How can a system automatically improve with experience? What types of methods and procedures can be followed to govern all learning processes?" [17]. The answers and research on these questions form the foundation of the ML definition.

Mitchell defines ML as the study of computer algorithms that improve themselves automatically through experience [18]. Instead of explicit programming, which can be impossible for too complex problems, the goal is to design algorithms that learn new behaviors and improve through experience. Because *learning* is ensured with data and initial algorithm structure, ML is considered a field of Statistics and Computer Science. However, it would be erroneous to suggest that ML is only related to these fields, as it is a field that arose from the idea of mimicking the learning mechanisms of humans or other animals. Furthermore, defining what thinking is and what is considered intelligent opens up discussions in psychology, neuroscience, and related fields. ML, which also aimes at defining systems that automatically adapt or optimize to their environment, has a wide scope of research and application fields ranging from vision systems to economics, exhibiting being a multi-disciplinary field [17].

2.1.1 Data

The term *experience* refers to the past information that the ML model has access to. This is usually in the form of electronic data that has been collected and made available for analysis. Data are typically represented in a structured collection known as a dataset [19]. This chapter will expand on the assumption of tabular data, emphasizing that, while it is a structured method that is widely used in science, it is not the only way to present data.

A *datapoint* or *instance* is a single observation from a dataset. This corresponds to each row in the context of data frames or tabular data, and it is described as a single experiment. Every instance has a value for an attribute known as a *feature*, which is also referred to as an experiment observation. Data with n instances and m number of features is represented as below.

$$x_{11}, x_{12}, x_{13} ... x_{1m}, y_1$$

$$x_{21}, x_{22}, x_{23} ... x_{2m}, y_2$$

$$...$$

$$x_{n1}, x_{n2}, x_{n3} ... x_{nm}, y_n$$

The values of various features of a particular instance are represented by a vector as $[x_{i1}, x_{i2}, x_{i3} ... x_{im}]$. This vector can be interpreted as the input of the algorithm. Along with features, data points can also have labels y_i, which are also known as *targets*. These are the expected outputs of a learning algorithm for each instance. The data type of the target and set of values can differ, and the name of the problem is based on this information as clarified in the following subsection.

2.1.2 Learning

According to Simon's definition, *learning* refers to changes in a system that enable it to perform the same task more efficiently the next time [20]. One of the primary practical goals of ML is to generate accurate *predictions* for previously unseen data [19].

However, a model that performs well on previously given data may not perform well on a new, unseen data. Learning is fundamentally about *generalization*. The generalization ability is critical for prediction. If a model performs poorly in terms of generalization, it indicates that the model only memorized the given data and did not learn in principle. This is known as *overfitting* [19]. The data are partitioned to test generalization in learning into the following sets.

Traning Set: A collection of data used to train a learning algorithm on the given data.

Test Set: A collection of data used to assess the generalization capability of the model and performance on previously unseen data.

Validation Set: A set of data used to select appropriate values for the parameters of the learning algorithm.

2.1.2.1 Learning Scenarios and Tasks

The learning scenarios vary depending on the type of data available to the learner.

Supervised Learning: The goal of supervised learning is to learn how to map feature values to targets. They were previously defined as inputs and outputs, respectively. It is called supervised because there is a given label/target for every instance by a supervisor [21]. The learned relationship between input and output is used to create a model that can predict on unseen data by leveraging its ability to generalize.

Unsupervised Learning: Unsupervised learning differs from this method in that no target values are assigned by a supervisor; in this case, only the input data exists. The goal is to find regularities in the input, which is known in statistics as density estimation [21]. The emphasis is on pattern recognition or structure recognition. This estimation is used in ML to find input clusters (clustering).

Some common machine learning tasks that have been studied in-depth and used in this work fall under the category of supervised learning for having target values.

Regression: Regression problems use datasets with target values that are continuous numbers. A real value for each item must be predicted to perform regression [19].

Classification: A classification problem is one where the target value is discrete (e.g., represents some groups). Each item is assigned to a category rather than predicting a real value.

2.1.2.2 The Quality of Learning

To obtain a quantitative analysis of the predictions, performance metrics or error measures are employed. This numerical value indicates the success of the algorithm. The metric used can differ depending on the problem type or context. The emphasis in this section is on error measures in supervised learning. Since the outputs of regression are continuous and the outputs of classification are discrete, the methods used as performance metrics vary between the two cases. Simply put, without knowing the target values, the model is allowed to make predictions, which are then compared to the actual values of the targets.

The number of actual and predicted outputs for each class is used to evaluate the performance of a classification model. The True Positive, True Negative, False Positive, and False Negative cases are counted in this comparison. Below, a confusion matrix with visualizations of these values can be observed. Error measures exist not only in computer science, but also as statistical tools used in many other disciplines. Accuracy, sensitivity, precision, and F1 scores are some of the most commonly used measures [22].

The confusion matrix displays the numbers of True and False predictions made using known data. The blue and gray backgrounds show cases that were predicted to

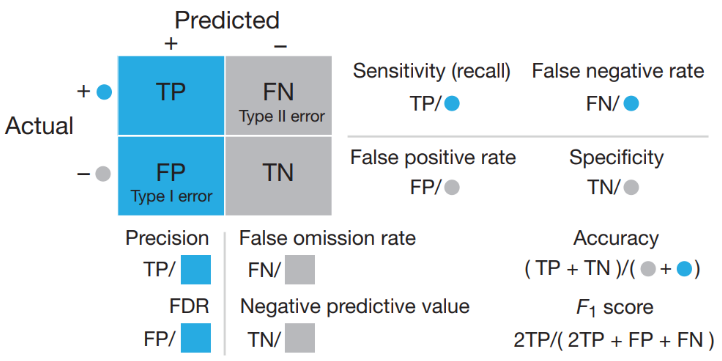


Figure 2.1: Confusion Matrix and Error Measures for Classification (Reprinted from [22])

be positive (TP+FP) and negative (FN+TN), while the blue and gray circles show cases that are known to be positive (TP + FN) and negative (FP + TN), respectively. FDR is false discovery rate. Equations for calculating each metric are graphically encoded.

Although the general preference for these metrics changes over time, some of the most commonly used metrics for regression problems are mean square error (MSE), RMSE, mean absolute error (MAE), and mean absolute percentage error (MAPE) [23]. In the following chapters, RMSE is employed as the error measure for regression problems. It is the standard deviation of the prediction errors. The differences between predicted and actual values are referred to as errors. Assume that after obtaining a specific model with a dataset of size n, n samples of unbiased model errors ϵ with $(e_i, i = 1, 2, ..., n)$ are collected. The following formula is used to calculate the RMSE of the predictions: [24].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

where $e_i = \hat{y}_i - y_i$ denotes each error with $\hat{y}_i$ as the predicted target value by the model and y_i as the actual target value.

2.2 Optimization

The field of optimization is associated with the methods or algorithms that can be used to solve complex problems by locating or approximating the *optima* [25]. Optima in plural or optimum in singular can be defined as the best solution to a problem. What is important to comprehend for this work is solving an optimization problem means, out of a typically very large set of possible solutions, finding the best solution(s) [26]. An instance of an optimization problem is defined as:

$$(S, f)$$

where:
S denotes the collection of all possible solutions, also known as the *solutions space* or *search space*.
f is a function defined on all elements of S that assigns each one a real number:

$$f : S \rightarrow \mathbb{R}$$

f is known as *fitness* and it quantifies the quality of the solution which makes it possible to interpret a solution good, bad or the best. Depending on the problem description, the term "best" can mean either the minimum or maximum value. Optimization seeks a solution that minimizes or maximizes f across the problem domain.

2.2.1 Fitness Landscapes

Fitness landscapes depict the fitness of each potential solution in a search space. Only having the solutions and their fitness does not connect these solutions. Having a sense of connectedness among all possible solutions and their fitness is beneficial in gaining an understanding of the problem and its characteristics [27]. The neighborhood concept ensures that the solutions are linked. A fitness landscape is a plot in which all of the solutions in S are represented horizontally, consistently sorted by the neighborhood structure N, and the fitness value $f(i)$ for each solution $i \in S$ is indicated vertically [26]. It is represented as follows:

$$(S, f, N)$$

The extent to which solutions are considered neighbors depends on the particular way in which possible solutions are represented and N is defined differently for each optimization problem. Hence, the triple (S, f, N) is defined differently for each problem at the outset.

There are local and global optimum points on a fitness landscape. The global optimum is the point at which the solution with the best fitness is observed, whereas the local optimum is the point at which the solution with the best fitness for a specific neighborhood is observed [27]. If there are no local optima in a fitness landscape, and the global optimum is unique, then it is classified as a *unimodal fitness landscape*, implying that there is only global optimum.

2.3 BioInspired and Evolutionary Algorithms

Heuristic Optimization is a subset of optimization methods named after the Greek word for "discovery". Heuristic methods are desirable and advantageous for optimization problems when a reliable exact method is unavailable, the exact method is computationally inefficient, or an approximate solution is sufficient [28]. It specifies a computational practice that selects an optimal solution by iteratively trying to improve a candidate solution with regard to a given quality measure [29].

As heuristic optimization methods, Bioinspired and EAs function by mimicking the process of computation and solution finding of generations of biological organisms [30]. There are a variety of Bioinspired Algorithms in the literature. For instance, Artificial Neural Networks are inspired by the structure of the human brain, EAs are inspired by the Theory of Evolution of Darwin [31], Particle Swarm is inspired by the collective intelligence of the animal groups [32]. There are also tens of animal-inspired algorithms including ants, bees, bacteria, cuckoo, firefly, frog, bat etc. as well as plant-based algorithms such as Flower Pollination Algorithm and Artificial Plant Optimization. The ability to return an approximated, or imprecise, solution to problems whenever they are unable to find a perfect one is a trait shared by the Bioinspired algorithms. The capacity of the system for generalization is enhanced by this feature.

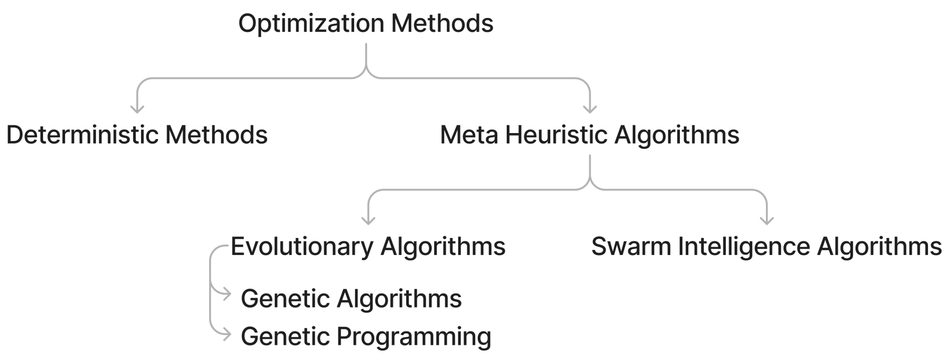


Figure 2.2: Briefly, optimization methods and the position of EAs in this context.

2.3.1 Genetic Programming

Turing proposed that a successful "learning machine" can be built by mimicking the learning process of a child, where the child has hereditary material and an education course. With this as a foundation, he proposed an accelerated *evolution* in which the concept of the *survival of the fittest* applied. The concept is based on *natural selection*, which is a mechanism in Charles Darwin's Theory of Evolution [31].

GP is an ML technique that falls under the category of Bioinspired Algorithms, more specifically Evolutionary Computation. The basic idea behind GP is to genetically breed a population of potential solutions [33]. An individual is a possible solution, and a population is made up of several individuals. Starting with a population in which each individual represents a computer program on its own, the population is allowed to evolve in order to find the most successful among them to solve a problem at hand. With sexual recombination (*crossover*) of the parental computer programs, GP generates a new population of offspring [2]. Because parental programs are chosen in proportion to their fitness measures, the Darwinian principle of survival of the fittest is applied. The goal of using natural selection and genetic operators to create a computer program that solves or approximates a given problem is, in fact, the definition of optimization. The concepts identifying natural selection are *reproduction*, *ability of adaptation to the environment*, *hereditariness*, *variation* and *competition*. These are duplicated to an extent in the algorithm [26]. Over many generations, GP algorithms may exhibit increasing average fitness and effectively adapt to environmental changes [2].

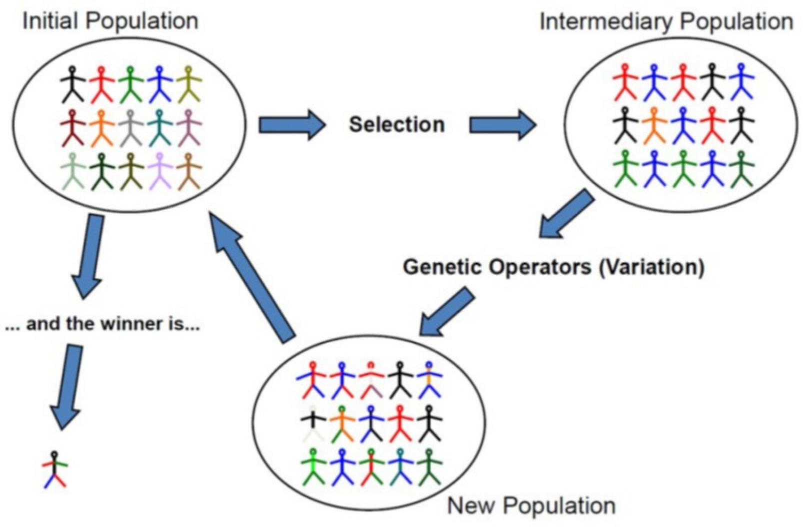


Figure 2.3: A graphic depiction of the iterative work-flow of an evolutionary algorithm, including GP, that is driven by the concepts of Darwin's Theory of Evolution.(Reprinted from [34])

At this point it is important to state that GP originated from Genetic Algorithms (GA) introduced by Holland in 1975 [35] and Goldberg in 1989 [36]. Just like GP, GAs are algorithms based on the mechanisms of natural selection and genetics. GAs are used to perform optimization by mimicking the biological systems' robustness, efficiency and flexibility [36]. Despite the fact that the evolutionary process is very similar to that of GPs, there is a critical difference between GAs and GPs, which is *representation*. Individuals in GAs are not computer programs, but rather fixed-length character strings that correspond to chromosomes in nature [36]. This representation, when used with GPs, limits the internal states of the system due to its fixed-length and cannot keep up with dynamically varying programs [2].

2.3.1.1 Representation of the GP Individuals

Abstract syntax tree representation is widely used with GP where a tree represents a computer program, a function that solves the given problem. The tree is constructed using *nodes*. The node values can be either *functions* or *terminals*. The terminals are specified as a constant value from a predefined set [37]. Node values are selected from two sets: Function Set (F) and Terminal Set (T). The types of functions may vary, but for the purposes of this thesis, examples of binary operations such as addition, subtraction, multiplication, and division would suffice, as would examples of unary operations such as sin, cos, and log. Because it corresponds to the structure of the tree, the representation is considered the *genotype* of an individual. The fitness of individuals is referred to as the *phenotype* of an individual. Having a tree-based representation may appear as having an infinite search space but the trees are limited by restricting their depth. In contrast to the fixed-length representation of GAs, tree representation of GP allows trees to grow. This may increase the generalization and solution capacity of GP compared to GAs.

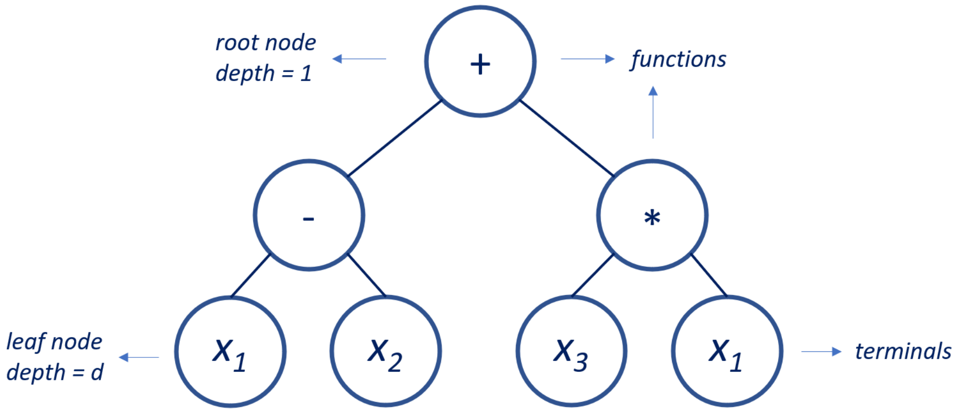


Figure 2.4: Abstract syntax tree representation in GP of the function $f(x_1, x_2, x_3) = x_1 - x_2 + x_3 * x_1$. (Inspired from [38])

2.3.1.2 Initialization

Initialization is the process of creating the trees of the first population. This is the first step for GP as it is for any evolutionary algorithm. The mechanisms of natural selection depend heavily on *diversity*. In GP, this refers to the variation in the genotype (representation) or phenotype (fitness) of the individuals within a population. Genetic search will often be more robust if the population is made up of a wider variety of individuals since it will encourage the exploratory stage of the search [39]. At this work, three major initialization methods is implemented.

Grow: The grow algorithm begins by creating the root of the tree. A random symbol from the Function Set (F) is chosen with uniform probability, and the operand is chosen from the combination set of F and Terminal Set (T). If the node value is a terminal, the node is a leaf, indicating that the branch stopped growing. This process is repeated until either the maximum depth is reached or all of the nodes at depth d have terminal values [40]. A pseudocode for the algorithm is below.

```
GROW(depth $d$, max depth $D$)
    Returns: a tree of depth $\leq D - d$
    If $d = D$, return a random terminal
    Else
○       Choose a random function or terminal $f$
        If $f$ is a terminal, return $f$
        Else
            For each argument $a$ of $f$,
                Fill $a$ with GROW($d + 1$,$D$)
            Return $f$ with filled arguments
```

Figure 2.5: Grow Algorithm (Taken from [40])

Full: Full initialization begins in the same way that grow does. It chooses a random symbol from F to become the root of the tree. However, nodes from depth 2 to $d - 1$ are only chosen using the function set F, and nodes at the maximum depth are chosen from the terminal set T. Thus, full differentiates from grow only in the line marked with ◦. It never selects a terminal at that stage, only a function [40].

The difference between these methods is that trees created with full initialization have all of their branches the same length, whereas trees created with grow initialization have branches with irregular shapes.

Ramped Half-and-Half: The most common and traditional method, and the one used in this work, is the Ramped Half-And-Half (RHH) method defined by Koza [2]. It makes use of the full algorithm together with the grow algorithm. This method is advantageous because it allows trees with a wide range of sizes and shapes to be generated. When used independently, full initialization and grow initialization produce less diverse populations since the generated trees may be too similar to one another. RHH is used to increase the initial diversity of the population diversity. The RHH method generates two groups of trees with depths ranging from 2 to d. As it stated in its name, 50% of the trees are generated by full and 50% is generated by grow.

2.3.1.3 Fitness Evaluation

The success of individuals is measured by calculating their fitness. During the evolutionary process, individuals with better fitness are more likely to be able to pass on their genes to the future generations following the Darwinian principle [31]. This phenomenon is ensured by the selection process.

2.3.1.4 Selection

Selection defines the procedure to select parents from an existing population to produce offspring that will form the new generations of evolution process. Even though different selection methods such as fitness proportionate selection (roulette wheel) [35], ranking selection [41] and tournament selection [42] can be used for the implementation of GP, tournament selection is opted for this work. This choice is motivated in Chapter 5.

Tournament selection is a selection algorithm with two fragments, the first of which is random and the second of which is deterministic. With a pre-defined tournament size k, to select one of the parents from a population, a subset of the population with size k is created randomly [43]. Out of this smaller set, the fitness of every individual is compared and the best one is chosen to become a parent. This step is entirely deterministic. The tournament size establishes a variable called selection pressure. Lower tournament sizes indicate a low selection pressure, while higher values indicate a high selection pressure, which corresponds to how likely the best individuals are to survive [26]. Always selecting the best individual of the whole population might sound like a good idea at first but maintaining diversity is crucial. By keeping the tournament size small, the selection pressure is reduced, and it is ensured that individuals with lower fitness are occasionally chosen to maintain diversity.

2.3.1.5 Genetic Operators

There are mainly two types of genetic operators. Operators used in GP will be presented without details since it differs from GSGP. But it is essential to comprehend the purpose of these operators which are in fact analogous their biological definitions. They introduce variation into the evolutionary process of the individual.

Crossover: This sexual recombination operation is carried out between two parents to produce two offspring. Each offspring inherits a portion of the representation of one parent and a portion of the representation of the other parent, just as the chromosomes of living beings [2].

Mutation: Unlike crossover, to perform this operation, only one individual is required. Mutation can occur by altering a part of a tree representation. This operation also introduces diversity and has the potential to produce beneficial results [2].

2.3.2 Geometric Semantic Genetic Programming

The point of focus of this thesis is GSGP, a GP modification. It is distinct from the GP due to its novel genetic operators. Traditional crossover and mutation algorithms are replaced with special operators, which distinguishes this method from GP. In GSGP, *semantics* is exploited by the genetic operators. Semantics is defined as the vector of the output values of an individual on the input data [26]. Traditional GP, according to Moraglio, ignores the "meaning" of the problems during the search process [12]. For example, in GP, the crossover is performed by swapping a subtree of parents without taking into account the effect on semantics, making this procedure blind. Since the success of the individuals is measured by fitness, it is coherent to consider the meaning, the semantics of the individuals during evolutionary process. Geometric operators which are used in GSGP are focalize on this meaning.

2.3.2.1 Semantics and Fitness Landscape

For a better understanding of GSGP, it is important to return to the context of optimization problems. A population of the tree-based representations of individuals can be defined as a *genotypic space* or *syntactic space* since this space contains the structures [26]. Since every individual has a semantic, it is also possible to define a space with semantics. Individuals in *semantic space* are represented by their semantics rather than tree structures. Due to the fact of semantics being a vector, this makes it possible for the individuals to be represented by a point in this semantic space [12]. In supervised learning, it is also possible to indicate the target, the global optimum in this space. Since the objective of optimization is to find an individual with best fitness or at least to have an approximated solution, fitness can be calculated as, using any metric, distance (e.g. Euclidean distance) between semantics and the target.

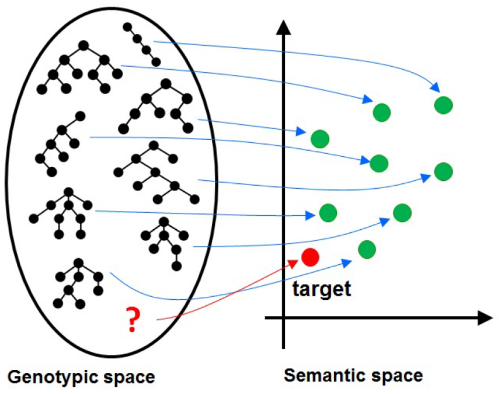


Figure 2.6: Relationship between genotypic and semantic space. The semantic space is depicted in 2D in the figure, which corresponds to the unrealistic case in which only two training instances exist. (Reprinted from [40])

2.3.2.2 Operators in Semantic Space

It was suggested in the literature in 1950 that when using evolution, it is important to find a method to prevent completely random mutations and to perform this operation in a way that has the potential to improve the success of an individual [16]. Two operators, geometric crossover and geometric mutation were introduced by Moraglio and Poli in 2004 [44]. As previously stated, the traditional crossover and mutation operators function on the genotypic (syntactic) representations of individuals. They proposed a framework for crossover and mutation operators based on landscape topology and geometry, which is associated with semantic space. This framework ensured an automatic method of deriving mutation and crossover from the neighborhood structure of a landscape. This clarified the connection between representation, genetic operators, neighborhood structure and distance in the landscape [44].

Geometric crossover is performed with two parents to obtain a single offspring located between its parents in space. It should be noted that this operation only creates one offspring.

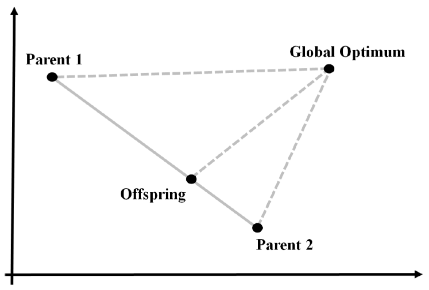


Figure 2.7: A graphical representation of the effect of geometric crossover, in the simple bidimensional case. (Reprinted from [26])

Geometric mutation, also known as box mutation, mutates an individual by randomly perturbing its coordinates within a certain range (box).

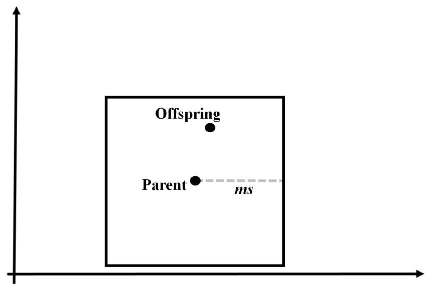


Figure 2.8: A graphical representation of the effect of geometric mutation (box mutation), in the simple bi-dimensional case. (Reprinted from [26])

2.3.2.3 Geometric Semantic Crossover (GSC)

When an offspring is created with geometric crossover in semantic space, where fitness is calculated as the distance of an individual to the global optimum, it will be located on a linear line between the parents. The betweenness property of geometric crossover ensures that offspring do not deviate further from the global optimum than the worst parent already is. This ensures that the fitness of the offspring is never worse than the fitness of the worst parent. Moraglio defined the GSC operator, which corresponds to geometric crossover on the semantic space [12, 26].

Geometric Semantic Crossover. Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, GSC returns the real function $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$.

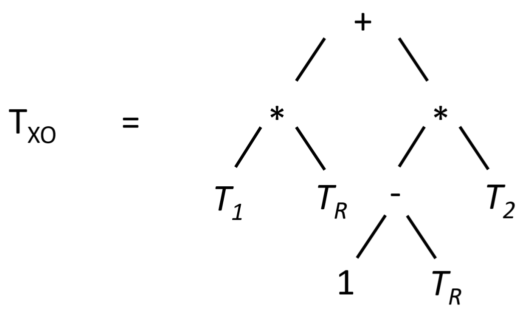


Figure 2.9: Tree representation of the GSC formula. (Inspired from [34])

2.3.2.4 Geometric Semantic Mutation (GSM).

Similarly, in semantic space, when an individual is mutated with geometric mutation, mutation can always improve fitness by allowing the point to move closer to the global optimum. Moraglio also defined the GSM operator, which corresponds to geometric mutation (box mutation) on the semantic space [12, 26]

Geometric Semantic Mutation (GSM). Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, GSM with mutation step ms returns the real function $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions.

The ms is used to tune the proportion of the perturbation (i.e., the limits of the box). Overall, GSGP have the ability to introduce a unimodal fitness landscape on the training set since genetic operations can always produce offspring that are closer to the target. This is phenomenon renders regression and classification problems easy to solve on training set.

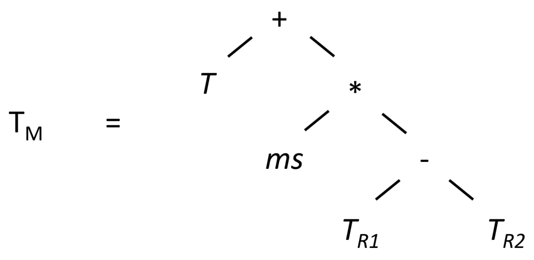


Figure 2.10: Tree representation of the GSM formula. (Inspired from [34])

2.3.2.5 Real-Life Applications of GSGP

GSOs, in addition to presenting a unimodal fitness landscape, had some limitations that rendered them useless in practice when they were first introduced. By their definitions, GSOs were causing the representation of the offspring to become larger for every generation. For future works, Moraglio suggested simplifying the syntax of the individuals to obtain a simpler genotype while preserving their semantics. Vanneschi et al. and Castelli et al. presented a novel implementation that made GSGP utilizable for complex real-life applications, for the first time [13, 45].

2.3.3 GSGP Implementation

This novel implementation of Vanneschi et al. [45] is based on saving new generation records that include operators and parents. As in standard GP, it begins by generating a random initial population. The individuals' representations and semantics are then stored in two tables. A new table is created for each new generation of individuals T. The table contains information about the operator with which the individual was created (crossover or mutation), as well as the ID. The ID is a memory pointer (reference) of the parents and their semantics, which were calculated using geometric semantic operators. Individuals from the new population are also saved in this manner.

An individual created by a crossover between parents T_1 and T_2 is represented by the triplet $T =< ID(T_1), ID(T_2), ID(R) >$, where R is the random tree used by the crossover. With the same form, an individual created by a mutation operator on T_2 is represented as $T =< ID(T_2), ID(R_1), ID(R_2) >$ where R_1 and R_2 are random trees. Random trees, along with their representations and semantics, must also be stored on a table.

Id	Individual
T_1	$x_1 + x_2x_3$
T_2	$x_3 - x_2x_4$
T_3	$x_3 + x_4 - 2x_1$
T_4	x_1x_3
T_5	$x_1 - x_3$

(a)

Id	Individual
R_1	$x_1 + x_2 - 2x_4$
R_2	$x_2 - x_1$
R_3	$x_1 + x_4 - 3x_3$
R_4	$x_2 - x_3 - x_4$
R_5	$2x_1$

(b)

Id	Operator	Entry
T_6	crossover	$\langle \mathrm{ID}(T_1), \mathrm{ID}(T_4), \mathrm{ID}(R_1) \rangle$
T_7	crossover	$\langle \mathrm{ID}(T_4), \mathrm{ID}(T_5), \mathrm{ID}(R_2) \rangle$
T_8	crossover	$\langle \mathrm{ID}(T_3), \mathrm{ID}(T_5), \mathrm{ID}(R_3) \rangle$
T_9	crossover	$\langle \mathrm{ID}(T_1), \mathrm{ID}(T_5), \mathrm{ID}(R_4) \rangle$
T_{10}	crossover	$\langle \mathrm{ID}(T_3), \mathrm{ID}(T_4), \mathrm{ID}(R_5) \rangle$

(c)

Figure 2.11: Illustration of the example described above. (a) The initial population P; (b) The random trees used by crossover; (c) The representation in memory of the new population P'. (Reprinted from [45])

Since the trees are stored with their semantics (evaluations), there is no need to explicitly build the tree and evaluate the entire tree after the creation of an offspring. Due to memory constraints, individuals that are not used for the creation of any offspring are deleted from the memory before passing on to the next generation because they will never be used again.

In terms of time complexity, this method is linear with respect to population size and the number of generations since it requires $O(n)$ space to store the individuals for g number of generations ($O(ng)$). At the end, it is sufficient to reconstruct the best individual making reconstruction a one-time process and it can be done offline from the algorithm. Overall, this novel implementation ensures GSGP to be applied efficiently on real-life data with high success rates.

2.4 Linear Scaling

While performing regression with GP, in cases where individuals that have a shape which is very similar to the one of the target functions, many solutions may receive a bad fitness value due to having different slope and/or being located in a distant position of the Cartesian space. LS was introduced by Keijzer to tackle this issue and improve the performance of GP on symbolic regression [6]. It is a method introduced to facilitate the task of GP of searching for the best function matching a set of known data.

LS modifies the fitness function in a very simple way, rescaling each individual by using their slope and intercept, two constants that can be easily calculated with a cost that is linear in the size of the training set. Let $P(x_i)$ be the output of GP individual P on the ith observation of the training set. A linear regression on the target values t can be performed using the equations:

$$\begin{aligned} b &= \frac{\sum_{i=1}^{n} \left[\left(t_i - \bar{t} \right) \left(P(x_i) - \overline{P} \right) \right]}{\sum_{i=1}^{n} \left(P(x_i) - \overline{P} \right)^2} \\ a &= \bar{t} - b\,\overline{P} \end{aligned} \tag{2.1}$$

where n is the number of training observations (fitness cases) and $\overline{P}$ and $\bar{t}$ denote the average output and the average target value respectively. Values b and a respectively calculate the slope and intercept of the set of outputs $P(x_i)$, such that the sum of the squared errors between t and $a + bP$ is minimized. After this, any error measure can be calculated on the scaled formula $a + bP$, for instance the RMSE:

$$\text{RMSE}(t, a + b\,P) = \sqrt{\frac{\sum_{i=1}^{n} (a + b\,P(x_i) - t_i)^2}{n}} \tag{2.2}$$

If a is different from 0 and b is different from 1, the procedure outlined above is guaranteed to reduce the RMSE for any formula [6].

By efficiently calculating the slope and intercept for each individual, the burden of searching for these two constants is thus removed from the evolution. GP is then free to search for the expression whose shape is most similar to that of the target function.

3 Literature Review

In accordance with the objectives of this work, this chapter focuses on the literature to capture the works on semantics in GP and "LS in GP" which inspired this thesis. Section 3.1 presents semantics in GP and Section 3.2 presents LS in GP.

3.1 Semantics in GP

Several semantically aware methods existed in the literature prior to Moraglio's [12] definition and presentation of GSGP. Moraglio and Poli [46] worked on the question "Is it possible to unify evolutionary algorithms within a general mathematical framework?" from 2004 to 2007. They created a formal structure for this unification over time. By introducing geometric crossover, they were able to unify fitness landscapes and genetic operators [47]. They presented a representation-independent topological definition for this operator and proved its independence using a variety of representation types as permutation [48, 49], syntactic trees [50], biological sequences [51], and sets [46].

Beadle and Johnson [52] published a novel implementation that compared Koza's [2] traditional GP with their method and reported that their method significantly improved GP performance. The technique is known as semantically driven crossover (SDC). During SDC, if the offspring has semantics equal to its parent, the program does not allow the offspring to be produced, which enhances the diversity in the population. They added the same feature to the mutation operator as well [53]. Beadle and Johnson also published a semantically driven initialization (SDI) algorithm, which performs the task of creating semantically unique individuals in the initial population [54].

Jackson [55] worked on the initial population's phenotypic (behavioral) diversity by enforcing the algorithm to generate semantically unique individuals. Krawiec et al. [56, 57], proposed a crossover operator for GP with tree representations. The operator is approximately geometric crossover in semantic space and presents perfect fitness-distance correlation on the semantic space. It is important to state that their method did not provide insight into the relationship between syntactic and semantic searches.

Nguyen et al. [58] applied a new form of semantic-aware crossover for GP that can be applied to both Boolean and continuous problems, extending on the ideas of Beadle et al. [52]. Beadle et al.'s work was limited to a study of Boolean problems. Nguyen et al. demonstrated that this method can improve GP performance. In a subsequent paper [59], they proposed a new method called sampling semantics for defining semantic distance between two subtrees. They performed mutation and crossover operations using these semantic distances.

Overall, these methods lacked one feature: they were not direct. There was no direct relationship between syntactic and semantic spaces in any of them. The offspring that will be a part of the new generation were found in a trial-and-error based system instead of having a guarantee that the offspring will respect the criterion needed by its construction. In his paper called "Geometric Semantic Genetic Programming" [12], Moraglio presented a novel form of GP by introducing geometric semantic operators. The operators enabled GP to search directly the semantic space. He also stated the fact that the representation of the offspring required simplification.

Vanneschi et al. [45], presented their novel method with a new efficient implementation of GSGP which made it possible to be used on complex real-life applications . They demonstrated that their GSGP outperformed standard GP systems with experiments on pharmacokinetics area. Castelli and Vanneschi et al., applied their method to several complex real-life problems such as predictions of concrete strength [60], Unified Parkinson's Disease Rating Scale assessment [61], human oral bioavailability [62] which continued to confirm the applicability and high performance of GSGP.

Castelli et al. [13] and Bakurov et al. [63] published works presenting GSGP frameworks and libraries. The code of Castelli et al. is an open-source GSGP software written in C++ with Moraglio's [12] geometric semantic operators and Vanneschi et al.'s [45] novel implementation. The software incorporates the entire GSGP process, from initialization to prediction with unknown data. The RHH method is used for initialization; GSC and GSM are used for genetic operators; and the program uses tree representations. The software developed by Bakurov et al. is a general-purpose optimization library that includes the GSGP algorithm. The multipurpose library includes common optimization problems such as the Knapsack Problem and the Traveling Salesperson Problem, as well as optimization algorithms such as Genetic Algorithms, Swarm Intelligence, and GSGP. Python is used to create the library. Because the GSGP code uses GPU-accelerated computing, the algorithm is faster than other GSGP algorithms. Neither library includes a visualization or statistical output package, nor does it support multiprocessing.

3.2 LS in GP

Regression and LS have long been used in general practice. It has been studied how to combine numerous expressions formed by GP using various types of multiple linear regression.

Iba et al. [64, 65] investigated the integration of multiple regression analysis method and GP-based search strategy. Multiple linear regression was also used to find coefficients in polynomial regression models. They looked into improving GP systems that search for polynomial models for non-linear dependencies. They used a set of transfer polynomials to become more flexible when constructing models. Nikolaev et al. [66] worked on the regularization of inductive GP tuned for learning polynomials. They presented a general approach to polynomial models when searching for higher order multivariate polynomial represented as tree structures.

Hiden et al. [67, 68] presented that GP can be used to extend partial least squares (PLS) and principal components analysis (PCA) for the modeling of non-linear process systems and stated that GP based systems outperformed standard GP and other non-linear PLS and PCA algorithm. In order to use GP with non-linear models, McKay et al. [69] have investigated the combination of GP with non-linear continuum regression.

However, multiple linear regression requires a matrix inversion, either over the dataset or the covariance matrix, which makes it a fairly costly procedure. The necessity to specify the number of coefficients to be used presents another issue with multiple linear regression. The likelihood of overfitting also increases as this number rises.

On the other hand, Keijzer's work showed a dramatic improvement in the performance of GP for symbolic regression by applying LS to the error measure [6]. In his first contribution, Keijzer demonstrated the benefits of LS on several synthetic test functions. Shortly after, he published another article, where he gave theoretical corroboration to the success of LS [70]. After Keijzer's contribution, LS has been used in several benchmark problems and real-life applications.

For instance, Archetti et al. [7], reported using LS with GP to improve the performance on several regression tasks related to the area of drug discovery. A few years later, the same authors also successfully applied LS with GP on another problem from the medical field, consisting in predicting the effect of an anticancer therapy on a specific cohort of patients [71]. In the same year, Raja et al. [8] also combined LS with their GP system for applications in the telecommunication area and concluded that the system that used LS outperformed the system that did not use it.

A general trend has also been to integrate LS in GP systems that also contain other novel methods. For instance, Pennachin et al. [72] used affine arithmetic to improve both the performance and the robustness of GP for symbolic regression, and they also performed LS of outputs before fitness evaluation. The presented results indicate that the proposed system reduces the number of fitness evaluations during training and improves generalization of GP, reducing overfitting. Similarly, Azad and Ryan [73] integrated LS and a method to maintain diversity in a GP system aimed at exploring lifetime learning.

Few years later, Virgolin et al. [9] applied LS to their GP-based algorithm, called GP-GOMEA, on a symbolic regression problem from the area of oncology. Later, in another work where several other real-world datasets were employed [74], the same authors confirmed the power of LS, successfully integrating LS in a semantic backpropagation-based GP system. To the best of our knowledge, this is the first and only contribution in which LS was integrated in a GP system that, although rather different from GSGP, uses semantic awareness to improve the search.

Recently, Ruberto et al. [10] tackled dynamic target problems by integrating LS with a GP system using hinge-loss function to evolve a set of discriminant functions for multi-class classification. The authors reported on the advantage of the version that uses LS. Later, these results were confirmed and extended, providing an upper bound to the error in dynamic symbolic regression [10, 75] and classification [76].

It is also possible to come across works (e.g. Medernach et al. [77], Sambo et al. [78]) that compare the performance of their GP system to the GP system with LS and present their superiority.

Despite the several successes on real-life applications, Costelloe and Ryan [11] pointed out that several methods that improve GP's training performance, including LS, may not improve GP's generalization ability as well. This consideration is important, since it partially reflects some of the findings of this work.

4

Methodology

This chapter outlines the methodology employed in this study, which seeks to investigate a system that combines the aforementioned benefits of both GSGP and LS. Specifically, the system utilizes GSOs to explore the search space, while guided by the LS fitness function. The system is referred to as GSGP-LS. The general layout of the library is explained in Section 4.1, followed by a detailed discussion of the library's various classes. The chapter follows by defining the most comprehensive class, GSGP, in Section 4.2. The Population class is discussed in Section 4.3, which includes relevant processes such as initialization, offspring creation, and population scoring in Subsections 4.3.1, 4.3.2, and 4.3.3, respectively. The Individual class is discussed in Section 4.4, which includes a subsection on reconstruction in Subsection 4.4.1. The chapter concludes with the presentation of the Node class in Section 4.5.

4.1 General Organization of the GSGP-LS Library

A GSGP algorithm is developed with the following system requirements:

- LS can be implemented.
- It is possible to solve both regression and classification problems.
- Hyperparameters are easily changeable for different datasets.
- Visualization of experiment results can be drawn.
- An acceptable runtime is ensured.

The extensibility of the library was a consideration during its design, with the intention of facilitating future research and development. The following subsections will elaborate on the library's architecture and entities. The library was constructed based on the theoretical knowledge that was previously presented. It can perform training of GSGP models, reconstructing the representation of the best individual, running trained models for predicting, obtaining statistical results and visualizations.

The code is written in Python programming language. Python has an active usage among the data science community. It contains libraries for easy data reading as well as common statistical analysis tools. It also has a high prototyping speed, all in all providing ease of development.

The library comprises a diverse set of classes and functions pertaining to the GSGP algorithm. Moreover, it incorporates functions and helper classes that enable the handling of dataset reading, basic statistical operations, graphing related code, multiprocessing support, and other utility functionality that lacks scientific relevance. The library's design embodies an entity-based object-oriented approach, whereby several modules listed herein implement a singular theoretical concept. The relevance between the entities and the theoretical concepts is shown in the table presented below. Commencing with the most comprehensive entity, the GSGP, the subsequent subsections introduce the Population, Individual, and Node.

Table 4.1: GSGP Entities

Theory	**Library**
An Optimization Problem	GSGP
A Tiny Sample of the Solution Space	Population
A Single Solution	Individual
Tree Representation	Node

4.2 GSGP

A GSGP problem is represented through the GSGP class, which encompasses the training and prediction procedures and features real-time reporting during the training process. The user-defined hyperparameters that are intrinsic to the GSGP algorithm are incorporated in this class. The hyperparameters, their respective definitions, and the ranges within which they are defined are comprehensively listed below. The following chapter will present the specific values that have been selected for these hyperparameters along with their corresponding justifications.

Table 4.2: GSGP Parameter Definitions

Parameter	**Definition**	**Data Type and Range**
max_iter	Number of generations	integer
pop_size	Population size	integer
f_selection	Selection method	string ('tournament')
		tournament.size: integer
elitism_size	Number of elites	integer
f_fitness	Fitness function	string ('rmse' or 'fscore')
prob_xo	Crossover probability	float $[0, 1]$
prob_mut	Mutation probability	float $[0, 1]$
mutation_step	Mutation step	float
	for random_mutation_step:	lower_range, upper_range: float

To train the GSGP model, this class invokes the operation to initialize a population, then continues the process by recording statistical information such as the training fitness and test fitness of each generation (fitness over time) as well as predictions made with the best individual. The general process differs depending on whether the problem type is regression or classification. The following sections will explore these differences in detail. The GSGP class uses the following modules to test and train the GSGP model.

4.3 Population

Population class represents a group of individuals, and it manages operations associated with the creation of initial and new generations. It invokes the initialization function, performs the procedure of evolving a population by creating a new generation. Furthermore, the class scores a population by calculating its average fitness, also keeping the record of the population's fittest individual. Since the prediction requires knowledge of the best individual in any given generation, user calls via GSGP are routed to this class, which then computes the necessary predictions.

It is crucial to note a distinction between regression and classification problems in this step. In classification problems, predictions are normalized using a sigmoid function, which confines the predictions to a range of 0 to 1. After normalization, a threshold value of 0.5 is utilized to classify the predictions, with those above 0.5 assigned to target class 1 and those below 0.5 assigned to target class 0.

4.3.1 Initialization

Initialization of a new population is performed by creating an initial population (represented as *T population*) with a given size for a population, pop_size. A simple pool of random trees (represented as *R population*) of the same size is also created at this class. The pseudo-code of initialization algorithm is presented below.

Algorithm 1 Initialize a new population

```
1: function RHH($group\_size, tree\_depths$)
2:     for all $depths$ in $tree\_depths$ do
3:         $i \leftarrow 0$
4:         while $i \le group\_size$ do
5:             Append the population with an individual initialized with grow method
6:             and having a current depth equals to $depths$
7:             Append the population with an individual initialized with full method
8:             and having a current depth equals to $depths$
9:             $i$ += 2
10:        end while
11:    end for
12: end function
```

```
function INITIALIZE FIRST GENERATION($population, data$)
    Create an empty $t\_population$
    Create an empty $r\_population$
    $tree\_depths \leftarrow [2, 3, 4, 5, 6]$
    $group\_size\_of\_t\_population \leftarrow$ Half of the t_population_size divided by the
    number of $tree\_depths$
    $group\_size\_of\_r\_population \leftarrow$ Half of the r_population_size divided by the
    number of $tree\_depths$
    Append $t\_population$ with $RHH(group\_size\_of\_t\_population, tree\_depths)$
    Append $r\_population$ with $RHH(group\_size\_of\_r\_population, tree\_depths)$
    for all individuals in $t\_population$ do
        Calculate initial semantics
    end for
    for all individuals in $r\_population$ do
        Calculate initial semantics
    end for
end function
```

4.3.2 Offspring Creation

During the offspring creation process, the Population class also performs the calculation of the semantics for the *T population* and *R population*, which are then stored in individuals. As shown in the pseudo-code of the evolution process below, the process starts with the selection of two parents using a given selection method, f_selection. Subsequently, a new individual is created probabilistically using one of three different methods - mutation, crossover, or replication - based on an independent random variable, with the probabilities determined by prob_xo and prob_mut. Finally, the semantics of the offspring are computed using those of its parents.

Algorithm 2 Generation of a new generation

```
1: function CREATE NEW GENERATION(population)

2:     Create an empty list $new\_generation$
3:     $number\_of\_individuals\_to\_create \leftarrow get\_pop\_size(population)$
4:     $number\_of\_elites \leftarrow get\_number\_of\_elites(population)$

5:     if $number\_of\_elites \geq 1$ then
6:         $number\_of\_individuals\_to\_create- = number\_of\_elites$
7:         Copy elites to the $new\_generation$
8:     end if

9:     for i in [0...number_of_individuals_to_create] do
10:        Select parents $T1$ and $T2$ from $population$
11:        $independent\_variable \leftarrow random\_uniform(0, 1)$

12:        if $independent\_variable \leq get\_xo\_probability(population)$ then
13:            Get a random individual $R$ from $r\_population$
14:            $new\_individual \leftarrow xo(T1, T2, R)$

15:        else if $independent\_variable \leq get\_xo\_probability(population) + get\_mut\_probability(population)$ then
16:            Get random individuals $R1$ and $R2$ from $r\_population$
17:            $new\_individual \leftarrow mutation(R1, R2, T1)$

18:        else
19:            $new\_individual \leftarrow T1$

20:        end if
21:        Append $new\_generation$ with $new\_individual$

22:    end for

23:    return $new\_generation$

24: end function
```

4.3.3 Scoring a Population

As the Population class is responsible for all individuals in the system, it also scores the current generation. The ordering of individuals varies based on the fitness function being utilized. Values closer to 1 indicate superior fitness for F1 score, while values closer to 0 indicate superior fitness for RMSE. To ensure that the library remains extensible, the *fitness functions* class must conform to the protocol of the fitness function. This is accomplished by sub-classing the abstract class of fitness functions and implementing the requisite methods. Any fitness function that adheres to this protocol can be utilized to generate scoring, as shown in the pseudo-code below. If the current individual is very decisive on a particular output, a bad fitness (very low or very high depending on the fitness function) score is assigned to that individual, as in Keijzer's paper [6].

The library is designed to operate with or without LS for comparability. As presented in the pseudocode, if the LS option is selected, a and b are calculated and the results are scaled according to the method outlined in Chapter 2, Theoretical Background.

Algorithm 3 Fitness calculation

```
1:  function ScoreCurrentGeneration($population, data$)

2:      for all individuals in population do

3:          for all rows in data do
4:              Get target value
5:              Get evaluation using semantics
6:              $row\_results \leftarrow tuple(target, evaluation)$
7:          end for

8:          $variance \leftarrow calculate\_variance(row\_results)$
9:          if $variance \geq 10^7$ or $variance \leq 10^{-7}$ then
10:             Assign a bad fitness value

11:         else
12:             if $Linear\ Scaling$ then
13:                 $a, b \leftarrow calculate\_linear\_scaling\_constants(row\_results)$
14:                 $row\_results \leftarrow$ Scaled results
15:             end if
16:             Calculate average fitness

17:         end if
18:         Assign the fitness value to the individual
19:     end for

20:     Sort population with respect to the fitness values
21:     return Individual with the best fitness

22: end function
```

4.4 Individual

Individual class represents an individual (i.e., a single solution to the problem). The class performs the mutation and crossover by taking an instance of an individual and returning the mutated individual or an offspring. GSC and GSM formulas used in this class are consistent with those outlined in the literature [12].

4.4.1 Reconstruction

It should be noted that a modified technique, distinct from the one presented in Chapter 2, has been studied to ensure the reconstruction capability. This technique leverages Python's garbage collection mechanism, which functions by keeping track of reference count - a counter that holds the number of references made for an object. Essentially, a Python variable is a reference to an object stored in memory. When the reference count of an object reaches zero, indicating that the object is no longer referred to by any variable, the garbage collector promptly removes it.

In this algorithm, each individual is represented by a tree structure. When an offspring is created in the Individual class, the crossover or mutation operation is performed with the parent's or a random tree's representation. Consequently, the tree representation of the offspring only contains references to the parent or the random tree, rather than the entire tree structure. Python's reference counting and garbage collection mechanism ensure that memory is periodically cleaned up, and any unused representations are erased from memory. In the creation of a new population, any individuals that were not used are erased.

As the tree grows, each node contains a recursive method, which can make the recursion memory-intensive, resulting in slower performance and limitations in terms of the number of nodes. This approach was implemented to investigate its potential, but it has its limitations. However, since the focus of this study is not on reconstruction, and this method is not required for experimental purposes, exploring ways to address these limitations is intriguing for future research.

4.5 Node

Node holds the data structure that objectifies a node in a tree representation. As outlined in the theory, each tree is constructed by combining individual nodes, and the Node structure is used to instantiate each node. To compute initial semantics for a tree, the *evaluate* function is called on this data structure. Although the semantics of a node can still be calculated with the evaluate function as the tree and data structure grow, it is preferable for computation time reasons to use the semantics of the parents without reevaluating them.

Values for the nodes are selected from a function set and a terminal set. The function set comprises four binary operations: addition, subtraction, multiplication, and division, implemented in Node. Division is defined as a protected division to avoid division by zero scenarios. Specifically, the division operation is performed only if the denominator is not zero; otherwise, the result is 1.To introduce non-linearity to the function set, several additional operations have been defined, including square root, negative x, sine, cosine, absolute value, reciprocal (1/x), exponential, and logarithm. These operations can be selected for use if needed. Terminal nodes are randomly selected from the available columns in the dataset's features. Additionally, terminal nodes can be expanded to include random numbers if desired by the user.

In accordance with the theoretical details, the RHH initialization method is also implemented in this section. The initialization process can be performed by RHH, full, or grow methods.

5 Experimental Study

This chapter presents the findings of the application of the methodology introduced in the previous chapter, along with accompanying experimental studies. To facilitate reproducibility, the experimental setup is outlined in Section 5.1. Subsection 5.1.1 includes parameter setting and Subsection 5.1.2 outlines case studies on regression, synthetic and classification datasets. The associated experimental results for all problems are given in 5.2 with separate subsections for regression experiments 5.2.1, synthetic dataset experiments 5.2.2 and classification experiments 5.2.3. Ultimately, the results are discussed in Section 5.3.

5.1 Experimental Setup

The objective of this work is to assess the effectiveness of integrating the LS technique with the GSGP algorithm. To this end, multiple experiments were conducted using both regression and classification problems. For each problem, the experiments were repeated using the GSGP alone and in combination with LS. Each configuration was implemented using the library presented in the Chapter 4.

Given the non-deterministic nature of the GSGP algorithm, with elements of randomness present in the initialization, evolution, mutation, crossover, and parent selection steps, the outcome of a single run is not considered reliable for comparison purposes. To obtain more accurate results, multiple independent runs of the algorithm were performed. 60 independent runs were conducted for each configuration, with the dataset randomly partitioned into 70% training and 30% test sets for each run.

5.1.1 Parameter Settings

It is worth noting that the primary focus of this work was not on optimizing hyperparameters but rather on making a fair comparison between the two approaches. In addition, optimizing hyperparameters would probably give origin to a different hyperparameter set for every case, thus generating a disorder. As such, the hyperparameters

Table 5.1: Parameter settings.

Parameter	**Value**
Generations	500
Population size	100
Initialization	RHH
	Maximum initial depth: 6
	Maximum tree depth: ∞
Function set	$\{+, -, \times, \div_p\}$
Terminal set	$\{features\}$
Selection	Tournament
	Tournament size: 2
Number of elites	1
Crossover probability	0.3
Mutation probability	0.7
Mutation step	$\mathcal{U}(0, 1]$
Train/test split	70/30

used in the experiments were relatively standard and can be found in relevant literature. The parameters used in the experiments are presented at the Table 5.1.

GSGP is recognized for its effectiveness when using smaller population sizes and larger number of generations compared to GP [79]. The selection of the number of generations and population size in the experiments was based on parameter settings from the literature [45, 80, 81]. The RHH initialization technique was used in all experiments as it is the most commonly employed method and effectively introduces diversity in the initial population [2]. The maximum initial depth was set to 6, as per the recommendations in the literature [2]. No constraints were placed on tree depths during the GSGP execution. The function set for the experiments included basic arithmetic operators such as addition, subtraction, multiplication, and protected division that returns 1 for the cases where denominator is 0. The set of terminal values was constructed using the features of the instances, as outlined in the theoretical framework. Tournament selection, as used in the literature, was used for the experiments, with a tournament size of 2 to decrease selection pressure and prevent premature convergence [82]. The number of elites (best individual carried into the next generation) was maintained at 1 for the same reason. The genetic operator probabilities were in line with the general guidelines for each method, without any particular tuning. As suggested in the literature, the mutation step is a random number between 0 and 1, that is generated independently of the previous ones at each mutation event [14]. The algorithm uses the same parameters with and without LS. RMSE is employed as the evaluation metric for regression problems, while F1 score is utilized for classification problems.

5.1.2 Case Studies

Nine complex real-world symbolic regression problems, frequently utilized as benchmark problems for GP experiments, were considered and referred to as *regression datasets*. Furthermore, six theoretical benchmarks were evaluated, directly taken from the work that introduced LS and referred to as *synthetic datasets* [6]. Additionally, three classification problems were included in the study to demonstrate the application of LS on GSGP for classification problems for the first time. These problems are referred to as the *classification datasets*.

5.1.2.1 Regression Datasets

The *Boston Housing* dataset, which pertains to housing prices, was chosen for its widespread use in regression analysis in the data science community. The *Concrete Compressive Strength* dataset was selected as it represents a challenging task in modeling the behavior of modern high-performance concrete, which is a highly complex material. The *Unified Parkinson's Disease Rating Scale* dataset constitutes a regression problem over a dataset containing data relative to patients diagnosed with Parkinson's disease. It is widely utilized in numerous GSGP studies. The *Istanbul Stock Exchange* dataset, which concerns financial markets, was also included due to the difficulty in generating useful stock market forecasts as a result of the dynamic and chaotic nature of the underlying stock market process. Four problems in the field of drug discovery, whose objective is the prediction of key pharmacokinetic parameters (*Human Oral Bioavailability*, *Median Oral Lethal Dose*, *Plasma Protein Binding Levels* and *Docking Energy*), and a dataset whose objective is the prediction of the response of cancer patients to a pharmacologic therapy, *Fludarabine*, were also selected for the experiments. These biochemical datasets present unique challenges, such as high multi-dependency among features by the nature of compenents and high dimensionality of the feature space, which require methods to handle overfitting and minimize generalization error [83]. The number of instances and attributes for each dataset are provided in Table 5.2.

Table 5.2: Number of instances and attributes of regression datasets.

Dataset	**Instances**	**Attributes**
Boston Housing	506	14
Concrete Compressive Strength	1030	9
Unified Parkinson's Disease Rating Scale (UPDRS)	5875	20
Istanbul Stock Exchange	536	8
Human Oral Bioavailability	359	242
Median Oral Lethal Dose: LD50	234	627
Plasma Protein Binding Levels: PPB	131	627
Docking Energy	992	268
Fludarabine	60	1376

In the following sections, the attributes used to describe the instances will be disclosed when applicable. For each attribute, its name, data type (type), minimum (min), maximum (max), average (avg), median (med), and standard deviation (SD) will be reported using abbreviated names. The values will be rounded to the nearest two decimal places, as this serves to conserve space.

Boston Housing

The dataset, provided by the Statistical Library at Carnegie Mellon University, is used to forecast housing prices using various variables of the area of Boston, Massachusetts [84]. These features are listed below and their descriptive statistics have been reported in Table 5.3. The target is the median value of owner-occupied homes (MEDV) in \$1000's. A graphical representation of the price distribution is illustrated in Figure 5.1a.

- **CRIM**: Per capita crime rate by town.
- **ZN**: Proportion of residential land zoned for lots over 25,000 sq.ft.
- **INDUS**: Proportion of non-retail business acres per town.
- **CHAS**: Charles River dummy variable (1 if tract bounds river; 0 otherwise) .
- **NOX**: Nitric oxides concentration (parts per 10 million).
- **RM**: Average number of rooms per dwelling.
- **AGE**: Proportion of owner-occupied units built prior to 1940.
- **DIS**: Weighted distances to five Boston employment centres.
- **RAD**: Index of accessibility to radial highways.
- **TAX**: Full-value property-tax rate per \$10,000.
- **PTRATIO**: Pupil-teacher ratio by town.
- **B**: $1000(Bk - 0.63)^2$ where Bk is the proportion of African-American people by town.
- **LSTAT**: % lower status of the population.
- **MEDV**: Median value of owner-occupied homes in \$1000's.

Table 5.3: Attributes of Boston Housing.

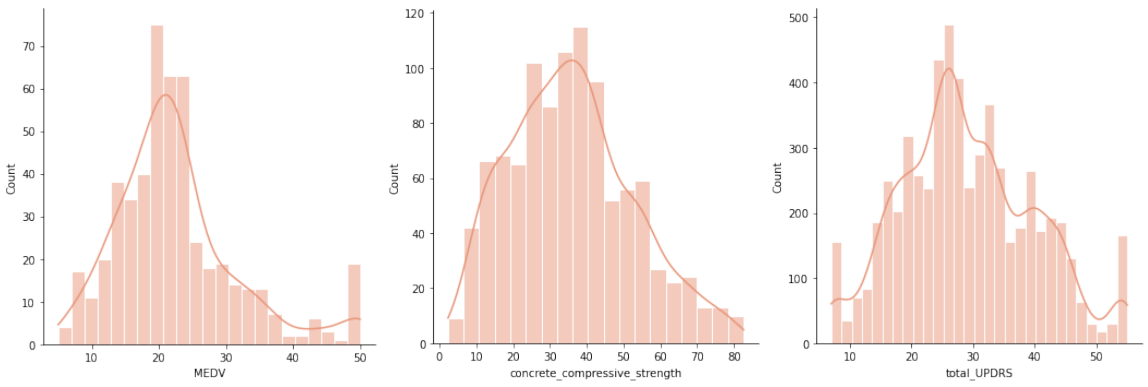
	Type	**Min.**	**Max.**	**Avg.**	**Med.**	**SD**
CRIM	float	0.0	89.0	3.6	0.3	8.6
ZN	float	0.0	100.0	11.4	0.0	23.3
INDUS	float	0.5	27.7	11.1	9.7	6.9
CHAS	int	0.0	1.0	0.1	0.0	0.3
NOX	float	0.4	0.9	0.6	0.5	0.1
RM	float	3.6	8.8	6.3	6.2	0.7
AGE	float	2.9	100.0	68.6	77.5	28.1
DIS	float	1.1	12.1	3.8	3.2	2.1
RAD	int	1.0	24.0	9.5	5.0	8.7
TAX	int	187.0	711.0	408.2	330.0	168.5
PRATIO	float	12.6	22.0	18.5	19.0	2.2
B	float	0.3	396.9	356.7	391.4	91.3
LSTAT	float	1.7	38.0	12.7	11.4	7.1
MEDV	float	5.0	50.0	22.5	21.2	9.2

Table 5.4: Attributes of Concrete Compressive Strength.

	Type	**Min.**	**Max.**	**Avg.**	**Med.**	**SD**
cement (kg/m^3)	float	102.0	540.0	281.2	272.9	104.5
blast_furnace_slag (kg/m^3)	float	0.0	359.4	73.9	22.0	86.3
fly_ash (kg/m^3)	float	0.0	200.1	54.2	0.0	64.0
water (kg/m^3)	float	121.8	247.0	181.6	185.0	21.4
superplasticizer (kg/m^3)	float	0.0	32.2	6.2	6.4	6.0
coarse_aggregate (kg/m^3)	float	801.0	1145.0	972.9	968.0	77.8
fine_aggregate (kg/m^3)	float	594.0	992.6	773.6	779.5	80.2
age (days)	int	1.0	365.0	45.7	28.0	63.2
concrete_compressive_strength	float	2.3	82.6	35.8	34.4	16.7

Concrete Compressive Strength

This dataset aimes at predicting the strength of high-performance concrete depending on several features of the ingredients [60, 85]. In addition to the fundamental components utilized in traditional concrete, high-performance concrete incorporates supplementary cementitious materials. To measure the compressive strength of concrete, cylinder samples must be subjected to compression testing using a machine. The resulting measurement is typically expressed in megapascals (MPa) and serves as the target value in this dataset (concrete_compressive_strength). Attributes used to describe each instance are presented in the Table 5.4 and distribution of the target values is depicted in Figure 5.1b.



(a) Boston Housing (b) Concrete Compressive Strength (c) UPDRS

Figure 5.1: Histogram of the target columns.

Unified Parkinson's Disease Rating Scale (UPDRS)

The dataset comprises a variety of biomedical vocal measurements and additional characteristics of 42 patients diagnosed with early-stage Parkinson's disease [86]. The patients participated in a six-month trial during which a telemonitoring device was utilized for remote monitoring of symptom progression, and data was collected accordingly. The primary objective of this data is to predict the clinician's overall scores on the Unified Parkinson's Disease Rating Scale (UPDRS) as reported in [61]. The target value in the dataset, represented as total_UPDRS, corresponds to these scores and its distribution is shown in Figure 5.1c. The attributes are listed below and their summary statistics are presented in the Table 5.5.

- **age**: Subject age.
- **sex**: Subject gender '0' - male, '1' - female.
- **test_time**: Time since recruitment into the trial. The integer part is the number of days since recruitment.
- **Jitter(%), Jitter(Abs), Jitter:RAP, Jitter:PPQ5, Jitter:DDP**: Several measures of variation in fundamental frequency.
- **Shimmer, Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, Shimmer:APQ11, Shimmer:DDA**: Several measures of variation in amplitude.
- **NHR, HNR**: Two measures of ratio of noise to tonal components in the voice.
- **RPDE**: A nonlinear dynamical complexity measure.
- **DFA**: Signal fractal scaling exponent.
- **PPE**: A nonlinear measure of fundamental frequency variation.
- **total_UPDRS**: Clinician's total UPDRS score, linearly interpolated.

Table 5.5: Attributes of UPDRS.

	Type	Min.	Max.	Avg.	Med.	SD
age	int	36.0	85.0	64.8	65.0	8.8
sex	int	0.0	1.0	0.3	0.0	0.5
test$_time$	float	-4.3	215.5	92.9	91.5	53.4
Jitter(%)	float	0.0	0.1	0.0	0.0	0.0
Jitter(Abs)	float	0.0	0.0	0.0	0.0	0.0
Jitter:RAP	float	0.0	0.1	0.0	0.0	0.0
Jitter:PPQ5	float	0.0	0.1	0.0	0.0	0.0
Jitter:DDP	float	0.0	0.2	0.0	0.0	0.0
Shimmer	float	0.0	0.3	0.0	0.0	0.0
Shimmer(dB)	float	0.0	2.1	0.3	0.3	0.2
Shimmer:APQ3	float	0.0	0.2	0.0	0.0	0.0
Shimmer:APQ5	float	0.0	0.2	0.0	0.0	0.0
Shimmer:APQ11	float	0.0	0.3	0.0	0.0	0.0
Shimmer:DDA	float	0.0	0.5	0.1	0.0	0.0
NHR	float	0.0	0.7	0.0	0.0	0.1
HNR	float	1.7	37.9	21.7	21.9	4.3
RPDE	float	0.2	1.0	0.5	0.5	0.1
DFA	float	0.5	0.9	0.7	0.6	0.1
PPE	float	0.0	0.7	0.2	0.2	0.1
total_UPDRS	float	7.0	55.0	29.0	27.6	10.7

Istanbul Stock Exchange

The data was gathered to study the influence of international markets on the Istanbul Stock Exchange (ISE) and predict the direction of movements in the ISE100 index [87]. It comprises daily price data for various stock market indices that are co-integrated with the ISE100. Target is the Istanbul Stock Exchange National 100 Index which represented as TL BASED ISE. The distribution of TL BASED ISE is plotted in Figure 5.2a. The attributes are as follows and their basic statistical analysis is presented in Table 5.6. The values are not rounded to the third decimal point instead of first because of low values.

- **SP**: The Standard and Poor's 500 (a stock market index monitoring the stock performance of 500 large companies recorded on stock exchanges in the United States) return index.
- **DAX**: Stock market return index of Germany.
- **FTSE**: Stock market return index of UK.
- **NIKKEI**: Stock market return index of Japan.
- **BOVESPA**: Stock market return index of Brazil.
- **EU**: MSCI European index.

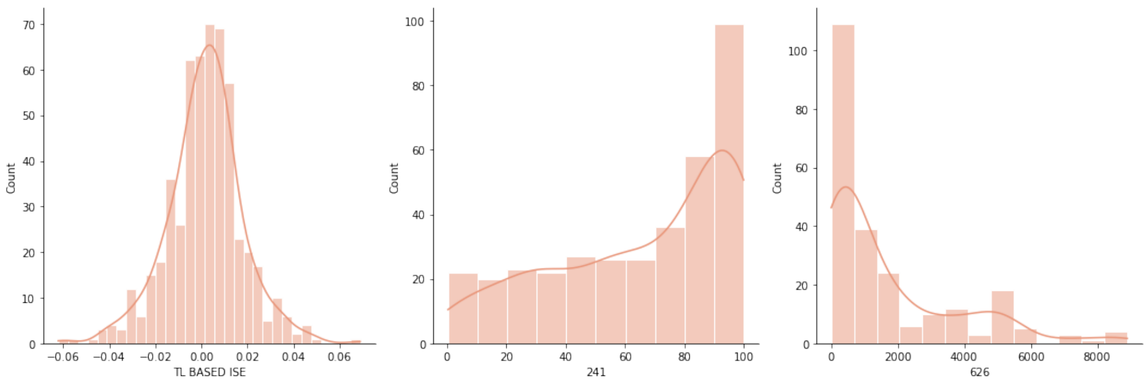
- **EM**: MSCI emerging markets index.
- **TL BASED ISE**: Istanbul stock exchange national 100 index.

Table 5.6: Attributes of Istanbul Stock Exchange.

	Type	Min.	Max.	Avg.	Med.	SD
SP	float	-0.054	0.068	0.001	0.001	0.014
DAX	float	-0.052	0.059	0.001	0.001	0.015
FTSE	float	-0.055	0.05	0.001	0.0	0.013
NIKKEI	float	-0.05	0.061	0.0	0.0	0.015
BOVESPA	float	-0.054	0.064	0.001	0.0	0.016
EU	float	-0.049	0.067	0.0	0.0	0.013
EM	float	-0.039	0.048	0.001	0.001	0.011
TL BASED ISE	float	-0.062	0.069	0.002	0.002	0.016

Human Oral Bioavailability

The dataset serves as a complex, real-world application in the field of pharmacokinetics. It involves forecasting the human oral bioavailability of a set of potential new drug compounds based on a set of molecular descriptors [88].
Human oral bioavailability, indicated by %F, is the parameter that measures the percentage of the initial orally administered drug dose that effectively reaches systemic blood circulation after passing through the liver. Each instance is a vector of molecular descriptor values identifying a candidate new drug and each column represents a molecular descriptor. The target is the human oral bioavailability, %F. The distribution of %F is plotted in Figure 5.2b. The attributes have not been detailed because of the large quantity and this will remain the same for the subsequent regression datasets.



(a) Istanbul Stock Exchange (b) Human Oral Bioavailability (c) LD50

Figure 5.2: Histogram of the target columns.

Median Oral Lethal Dose: LD50

The dataset being studied, which is drawn from the field of pharmacokinetics, concerns the prediction of the median lethal dose of a molecular compound [7]. This is a widely used metric for evaluating the toxicity of drugs. The acronym LD stands for Lethal Dose, and the term LD50 refers to the amount of a substance, administered in one dose, that results in the death of 50% of a group of test animals. Each data point consists of a vector representing a molecular compound, with the corresponding target being the LD50 value. The distribution is depicted in Figure 5.2c

Plasma Protein Binding Levels: PPB

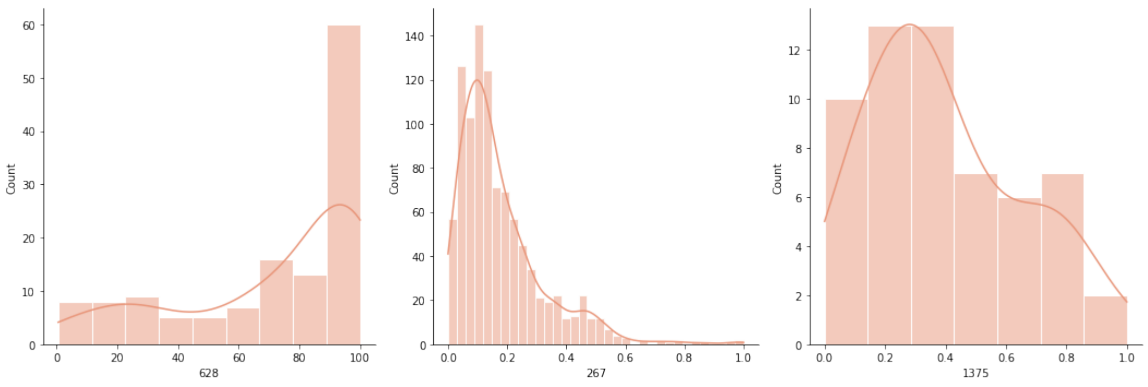
This dataset, from the field of pharmacokinetics, aims to predict the percentage of the initial drug dose that binds to plasma proteins [7]. This measure is of paramount importance as it relates to the distribution of drugs within the body and their ability to reach their intended target. The instances consist of vectors representing molecular compounds, with the corresponding PPB value serving as the target. The distribution of these values is plotted in Figure 5.3a.

Docking Energy

The dataset in question, belonging to the field of drug discovery and design, aims to predict the interaction energy between a candidate drug and its target tissue [83]. This metric, known as docking energy, quantifies the strength of binding between the molecules of the drug and those of the target. Each data point consists of a vector representing a molecular compound, with the corresponding target being the docking energy and the distribution is shown in Figure 5.3b.

Fludarabine

The dataset is used to predict the response of a set of cancer patients to the pharmacologic of the Fludarabine drug [89]. It was built looking for a functional relationship between gene expressions and responses to the Fludarabine. Each instance represents a gene expression and the features are the expression level of one particular gene. Target is the therapeutic response to the drug. The distribution of its values is illustrated in Figure 5.3c.



(a) PPB (b) Docking Energy (c) Fludarabine

Figure 5.3: Histogram of the target columns.

5.1.2.2 Synthetic Datasets

As previously stated in Chapter 3, Keijzer demonstrated a significant enhancement in the performance of GP in symbolic regression through the integration of LS, as presented in his paper [6]. He carried out a set of experiments to illustrate the practical benefits of utilizing LS. In order to eliminate any biases that may arise from an arbitrary definition of testing functions, he used problems that have been sourced from previous research that are relevant to the application and enhancement of symbolic regression. Besides being a good scientific practice to test a method (in this study, LS) on the same case studies that were used in the work that introduced that method, motivations for choosing these benchmarks are the same as in [6], i.e.: "many of the problems above mix trigonometry with polynomials, or make the problems in other ways highly non-linear". Also, it is relevant to point out that, as stated in [6], "being of low dimensionality does not make the problems easy however".

The specifics regarding the sampling approach and other problem-specific details are presented in Table 5.7. The problem names are kept the same as the paper. The datasets were hand-tailored according to these specifics. The training and testing intervals are indicated using the *[start:step:stop]* notation when the set is constructed with systematic intervals. The *rnd(min,max)* notation symbolizes random sampling within a specified range, while the mesh ([start:step:stop]) notation signifies regular sampling in two-dimensional space.

Table 5.7: Synthetic dataset settings.

Problem	Equation	range (train)	range (test)
4	$f(x) = x^3 exp^{-1} cos(x) sin(x)(sin^2(x) * cos(x) - 1)$	[0:0.05:10]	[0.05:0.05:10.05]
5	$f(x, y, z) = \frac{30xz}{(x-10)y^2}$	x,z = rnd(-1,1)	[0:0.05:10]
		y = rnd(1,2)	[0:0.05:10]
6	$f(x) = \sum_i^x 1/i$	[1:1:50]	[1:1:120]
7	$f(x) = logx$	[1:1:100]	[1:0.1:100]
8	$f(x) = \sqrt{x}$	[0:1:100]	[0:0.1:100]
9	$f(x) = arcsinh(x)$	[0:1:100]	[0:0.1:100]

5.1.2.3 Classification Datasets

For this section of the study, three different classification problems were considered. Table 5.8 provides the number of instances and attributes for each dataset as well as the ratio of target classes. The *Banknote Authentication Dataset*, created with authentic and forged banknote images, was chosen for its widespread use in binary classification analysis in the data science community. The *Spotify Funk Songs* dataset is a dataset that was established with colleagues several years ago, involving songs extracted from several playlist. Finally, *Breast Cancer Wisconsin (Diagnostic)* dataset contains predictions on breast mass images. It is widely utilized in numerous ML and GP studies.

Table 5.8: Classification Datasets.

	Instances	**Class Ratio**	**Attributes**
		(class -1: class 1)	
Banknote Authentication	1372	762 : 610	5
Spotify Funk Songs	985	711 : 274	14
Breast Cancer Wisconsin (Diagnostic)	569	357 : 212	31

Banknote Authentication

This dataset is used to differentiate between authentic and forged banknotes. The dataset was sourced from the UCI ML repository and is credited to Volker Lohweg of the University of Applied Sciences as the owner, with Helene Darksen also of the University of Applied Sciences credited as the donor [90]. Images acquired from authentic and forged banknote-like specimens were used to extract data. An industrial camera is used for digitization. To extract features from images, Wavelet Transform tool were used. A Wavelet Transform can be used to represent the image as a sum of different frequency sub-bands which is useful for feature extraction. The resulting image is a set of coefficients that represent the different frequency components of the image. The features are the variance, skewness, curtosis of the Wavelet Transformed image and the entropy of the image. Their characteristics are summarized in Table 5.9. The target value is -1 for authentic banknotes and 1 for the forged banknotes.

Table 5.9: Attributes of Banknote Authentication.

	Type	Min.	Max.	Avg.	Med.	SD
variance	float	-7.0	6.8	0.4	0.5	2.8
skewness	float	-13.8	13.0	1.9	2.3	5.9
curtosis	float	-5.3	17.9	1.4	0.6	4.3
entropy	float	-8.5	2.4	-1.2	-0.6	2.1
class	int	-1.0	1.0	-0.1	-1.0	1.0

Spotify Funk Songs

The dataset comprises songs from fifteen distinct playlists featuring various genres on Spotify, one of the largest audio streaming service providers. The playlists encompass diverse themes such as funky jams, heavy metal, romantic ballads, and relaxing piano. The platform, named Spotify For Developers, enables developers to extract detailed information about albums, tracks, and playlists. The dataset was established by me and my colleagues for an ML project aimed at predicting suitable songs for a funk band [91]. The attributes of the dataset are elaborated below and a statistical summary has been provided in Table 5.10. The songs can be classified according to the target column, which is binary in nature. If the song belongs to a playlist related to the funk genre, it is labeled as 1, and -1 otherwise.

- **danceability**: How suitable a track is for dancing.
- **energy**: Represents a perceptual measure of intensity and activity.
- **key**: The estimated overall key of the track.
- **loudness**: The overall loudness of a track in decibels.
- **mode**: Modality (major or minor) of the track.
- **speechness**: Presence of spoken words in a track.
- **acousticness**: A measure indicating the level of acousticness.
- **instrumentalness**: Measures the level of instrumentalness (no vocals).
- **liveness**: Detects the presence of an audience in the recording.
- **valence**: Describes the musical positiveness conveyed by a track.
- **tempo**: The overall estimated tempo of a track in beats per minute (BPM).
- **duration_ms**: The duration of the track in milliseconds.
- **time_signature**: An estimated overall time signature (beats per measure) of a track.

Table 5.10: Features of Spotify Funk Songs

	Type	Min.	Max.	Avg.	Med.	SD
danceability	float	0.0	0.9	0.5	0.5	0.2
energy	float	0.0	1.0	0.5	0.4	0.3
key	int	0.0	11.0	5.2	5.0	3.6
loudness	float	-43.9	-1.9	-14.2	-11.6	8.6
mode	int	0.0	1.0	0.7	1.0	0.5
speechiness	float	0.0	0.8	0.1	0.0	0.1
acousticness	float	0.0	1.0	0.5	0.5	0.4
instrumentalness	float	0.0	1.0	0.4	0.1	0.4
liveness	float	0.0	1.0	0.2	0.1	0.1
valence	float	0.0	1.0	0.4	0.3	0.3
tempo	float	0.0	211.3	116.1	112.8	31.6
duration_ms	int	62693.0	1215573.0	253833.2	229360.0	125928.2
time_signature	int	0.0	5.0	3.8	4.0	0.5
target	int	-1.0	1.0	-0.4	-1.0	0.9

Breast Cancer Wisconsin (Diagnostic)

The dataset comprises instances of digital images of fine needle aspirates (FNA) of breast masses [92]. Fine-needle aspiration is a diagnostic procedure employed to investigate lumps or masses. This technique involves the insertion of a thin, hollow needle into the mass to obtain a sample of cells, which, upon being stained, are subsequently examined under a microscope. The features of the dataset describe the characteristics of the cell nuclei present in the images. For each cell nucleus, there are 10 features with float data type:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness ($perimeter^2/area - 1.0$)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 31 attributes in total with the target. The extensive number of attributes has led to the decision to not display the statistical summary for each one. The target is the diagnosis which is classified as -1 for benign cases and 1 for malignant cases.

5.2 Experimental Results

5.2.1 Regression Experiments

The following figures from 5.4 to 5.18 display the performance of the GSGP and GSGP-LS algorithms on both the training and test sets for the nine considered regression datasets. The median fitness, measured in terms of RMSE, of the best individual is plotted against the generation number for each configuration, based on the results of 60 independent runs. The median was chosen over the mean as it is more robust to outliers, providing a more reliable representation of the data.

From the Figures 5.4, 5.7, 5.12, it can be deduced that GSGP-LS consistently outperforms GSGP on the training set for all considered problems. Concerning the results on the test set, it can be noticed from the Figure 5.4 that GSGP-LS outperforms GSGP on three of the considered problems:

- Boston Housing *(Boston)*
- Concrete Compressive Strength *(Concrete)*
- Unified Parkinson's Disease Rating Scale *(Parkinson)*

Nevertheless, as it can be observed in Figures 5.7 and 5.12, GSGP-LS suffers from overfitting issues on six of the considered problems:

- Istanbul Stock Exchange *(Istanbul)*
- Human Oral Bioavailability *(Bioavailability)*
- Median Oral Lethal Dose: LD50 *(LD50)*
- Plasma Protein Binding Levels: PPB *(PPB)*
- Docking Energy *(Docking)*
- Fludarabine *(Fludarabine)*

To assess the statistical significance of these results Mann-Whitney U test with statistical significance at $\alpha = 0.05$ was performed for both training and test sets, for each problem, at each generation, with the null hypothesis that the distribution of the RMSE of the best individual originated from the 60 runs is the same for GSGP and GSGP-LS. The results of comparing the two algorithms on the training set are deemed statistically significant, as indicated by consistently low p-values of approximately 10^{-12}. The p-values are reported in Figures 5.6, 5.9, 5.11. The evolution of the p-value resulting from the comparison of the two algorithms on the test set is reported in Figures 5.5, 5.8, 5.10. The significance threshold in each plot is also shown by red horizontal line at 0.05.

The results of the test set for the Boston, Concrete, and Parkinson datasets, as illustrated in Figure 5.4, provide robust evidence of the superiority of GSGP-LS over GSGP. The decline in the RMSE is not significant for GSGP-LS, as it commences from a much lower RMSE value. It is noteworthy to consider that these datasets possess a high ratio of instances to attributes and have been extensively studied, without any reported tendency to overfit. The p-value plots in Figures 5.5, 5.6 consistently demonstrate a value near zero, supporting the aforementioned observation.

Figures 5.7, 5.12 demonstrate clear overfitting for all datasets it demonstrates. The Bioavailability and PPB datasets share a characteristic on the test set: despite GSGP-LS starting from a lower value, it begins to increase and results in a poorer model, whereas this trend is not observed for the GSGP algorithm. The increase is more subtle in the case of Bioavailability and more pronounced in PPB. The possible causes are discussed in Subsection 5.3. Even though the p-values in Figure 5.8, 5.10 being below the threshold, this is due to the lines not having converged yet.

PPB is not the only dataset exhibiting a noticeable increase in RMSE. The LD50 and Fludarabine datasets also demonstrate similar characteristics, starting from a low RMSE and then increasing to a level even higher than that of GSGP. As a result, their p-values are initially close to zero but eventually exceed the threshold. While this does not directly cause overfitting, it is worth noting that these three datasets share a critical aspect: they all have a very low ratio of instances to attributes.

Istanbul and Docking does not suffer from low instance to attribute ratio but it can be observed that GSGP-LS cannot show a superior performance. It converges with GSGP and remains stagnant for many generations with Docking. Although it initially begins with a lower RMSE, the two methods have a comparable performance with Istanbul. For both, the p-value starts from a value close to zero, then gradually increases as the two methods start to have comparable performance and eventually the p-value starts to decline after about 400-450 generations. Then, GSGP starts performing better and RMSE starts increasing with GSGP-LS algorithm.

The fact that the GSGP-LS error curves on the test set for Bioavailability, LD50, PPB and Fludarabine are increasing since the very first generations should also be noticed. The analysis and discussion on the matter is deepened in Subsection 5.3.

Lastly, it is interesting to notice that, for both training and test sets, the initial RMSE of GSGP-LS is already lower than that at the end of evolution for standard GSGP. On the training set, this outcome was expected, given the known benefits of LS on the initial population [6].

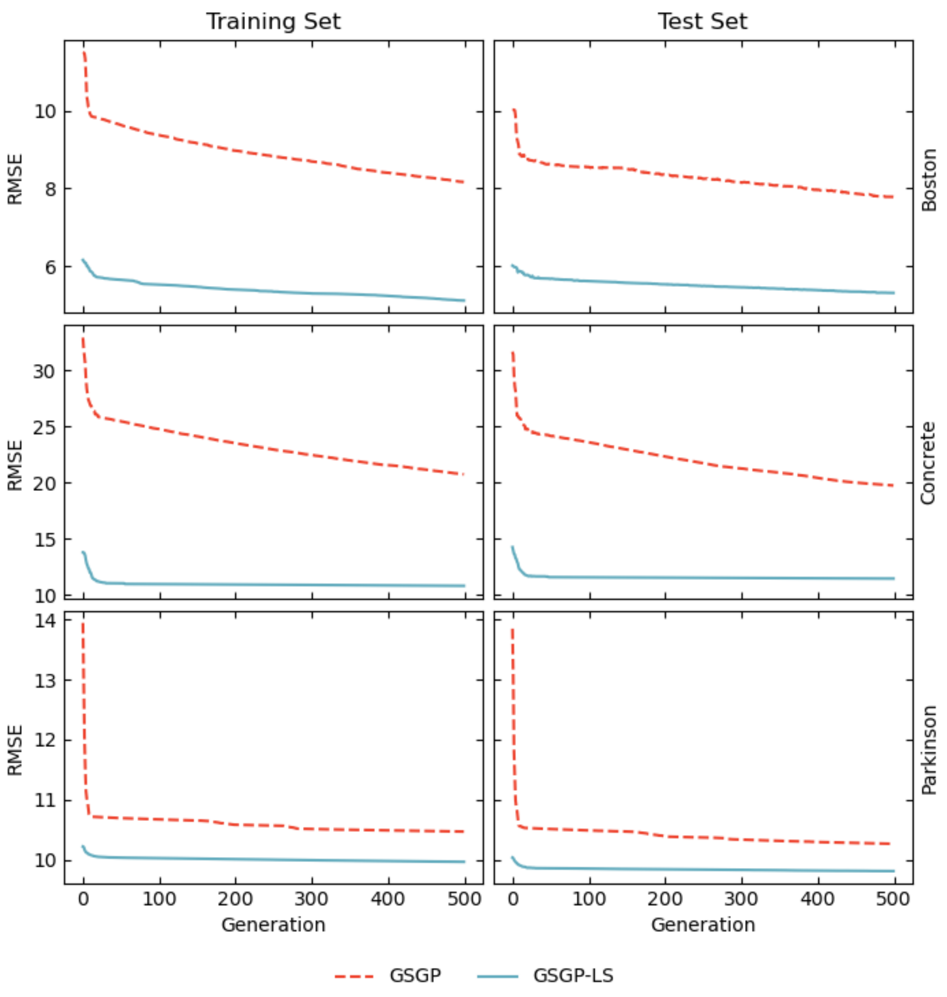


Figure 5.4: Results of the experimental comparison between GSGP and GSGP-LS with Boston, Concrete and Parkinson datasets.

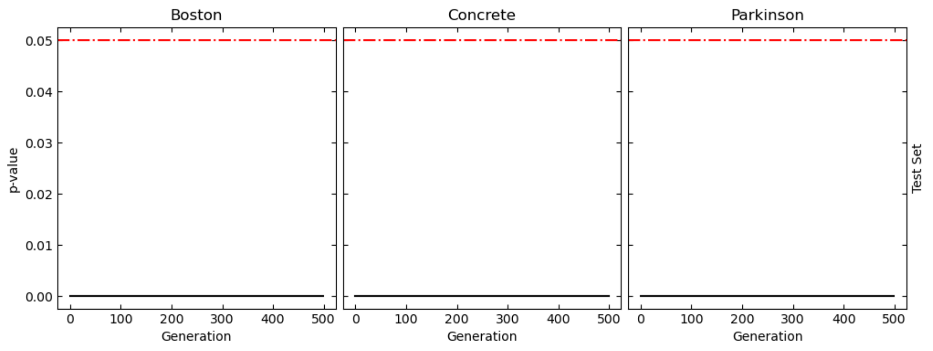


Figure 5.5: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for Boston, Concrete and Parkinson datasets.

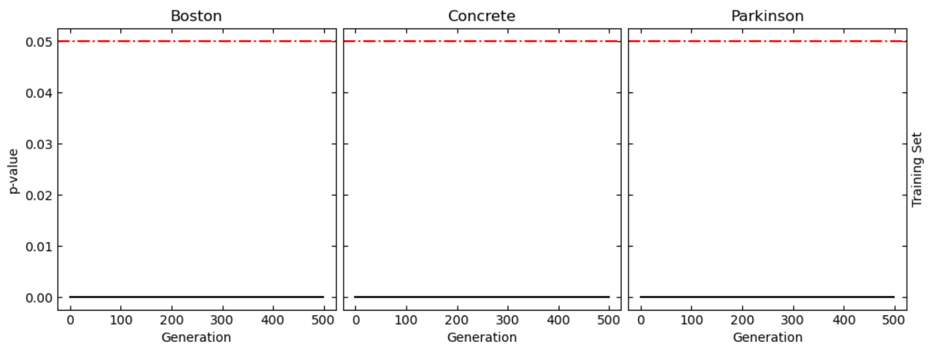


Figure 5.6: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for Boston, Concrete and Parkinson datasets.

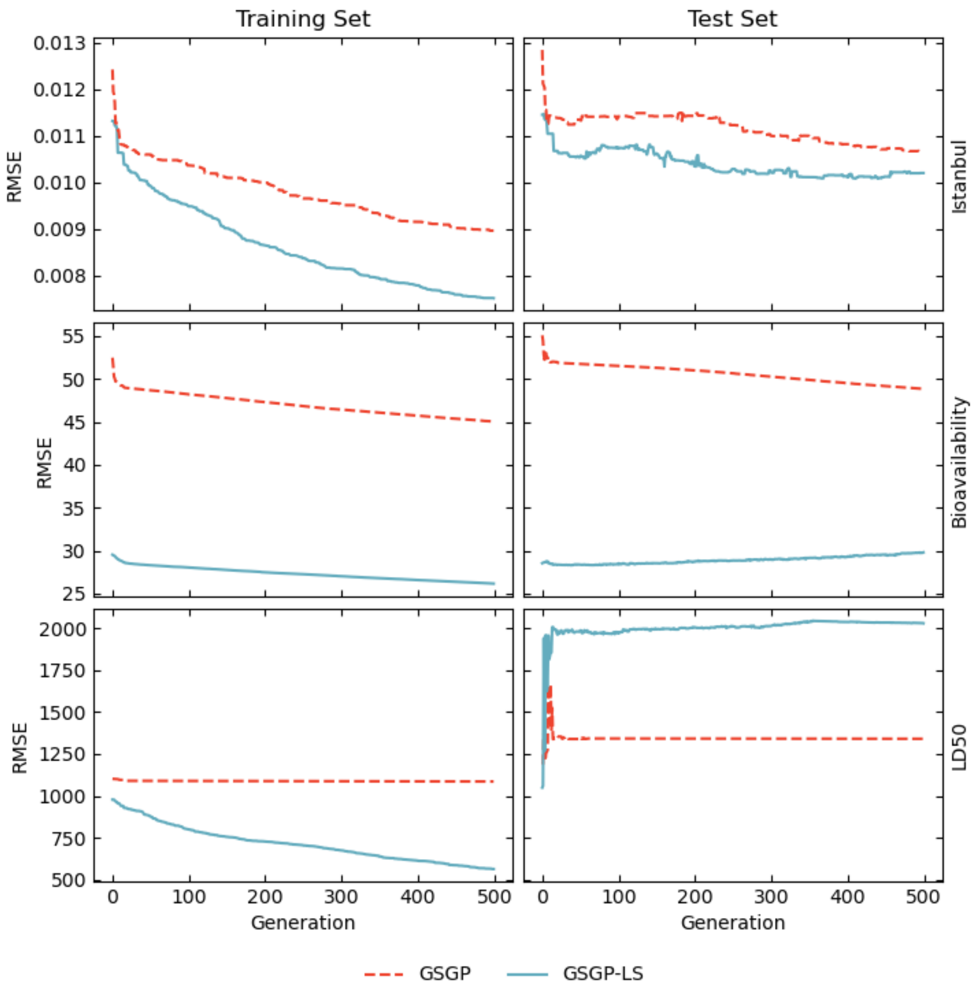


Figure 5.7: Results of the experimental comparison between GSGP and GSGP-LS with Istanbul, Bioavailability, LD50 datasets.

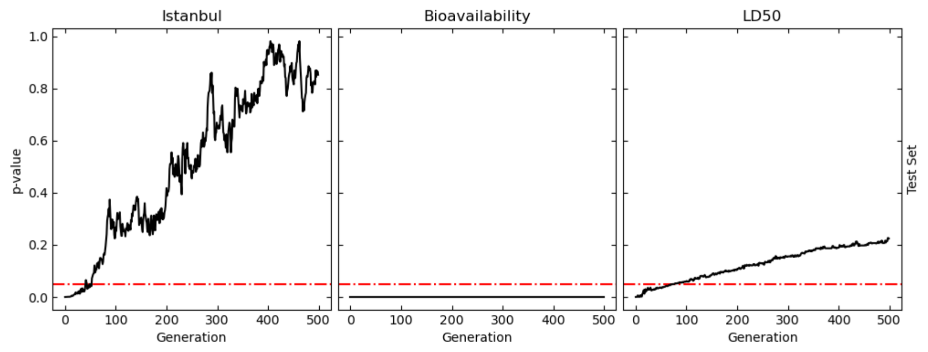


Figure 5.8: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for Istanbul, Bioavailability, LD50 datasets.

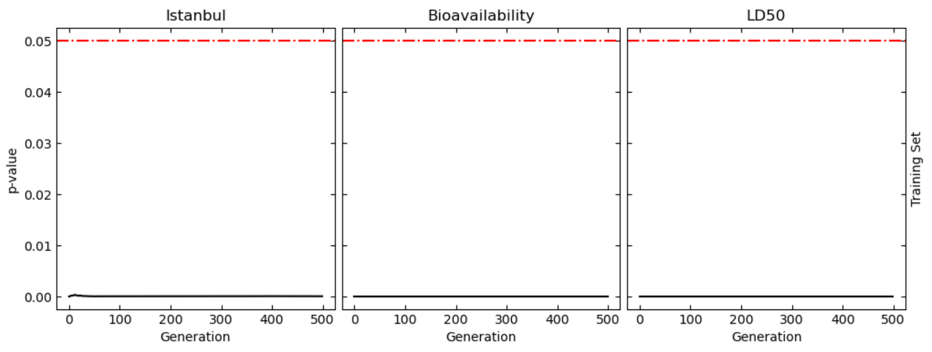


Figure 5.9: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for Istanbul, Bioavailability, LD50 datasets.

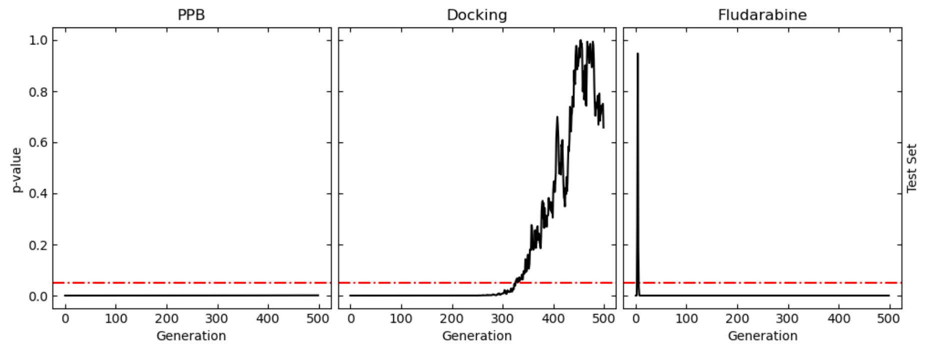


Figure 5.10: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for PPB, Docking and Fludarabine datasets.

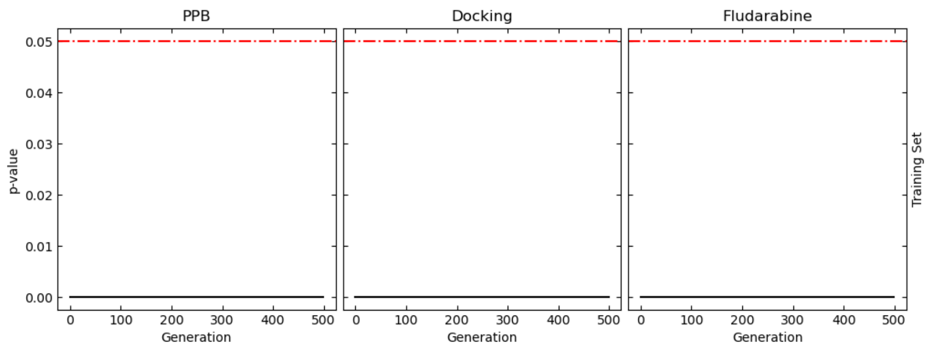


Figure 5.11: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for PPB, Docking and Fludarabine datasets.

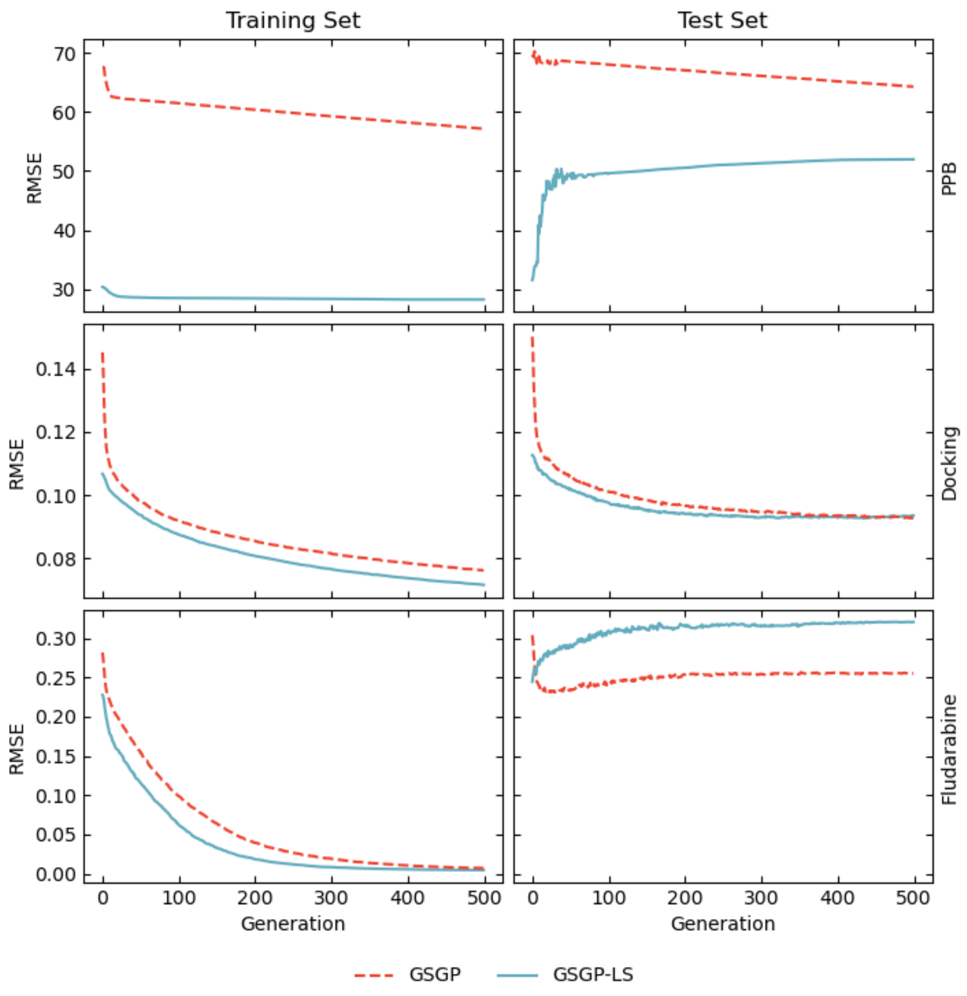


Figure 5.12: Results of the experimental comparison between GSGP and GSGP-LS with PPB, Docking, Fludarabine datasets.

5.2.2 Synthetic Dataset Experiments

The Figures 5.15, 5.16 showcase the performance comparison between the GSGP and GSGP-LS algorithms on six problems, using the median RMSE of the best individual as a metric. The results are based on 60 separate runs and plotted against the generation number. The evolution of the p-value resulting from the comparison of the two algorithms on the test set is reported in Figures 5.13, 5.17 and training set is reported in Figures 5.14, 5.18.

By the Figure 5.15, it can be observed that problems 4 and 5 exhibit overfitting issues on both training and testing sets, whereas problems 6, 7, 8, and 9 indicate that GSGP-LS consistently outperforms GSGP on both sets as shown in Figures 5.15, 5.16. The statistical significance of their results is evident from the Figures 5.13, 5.17.

The RMSE plot of problem 4 follows a trend similar to that of the Istanbul and Docking datasets, where GSGP-LS initially outperforms GSGP but eventually experiences a decline in performance. First, it becomes comparable with GSGP algorithm and then GSGP outperforms it. p-value, as in line with this behaviour, starts from a level close to zero, exceeds threshold and starts decreasing around generation 350.

The inferior performance of GSGP-LS on problem 4 can be attributed to the limited nonlinearity of the function set, which may not match the shape of functions with a sinusoidal component. It can be noted that problem 5 exhibits a relatively constant behavior throughout the process, a deviation from the other functions, which can be attributed to the presence of division in its mathematical formula. These two topics will be explored in greater detail in Subsection 5.3.

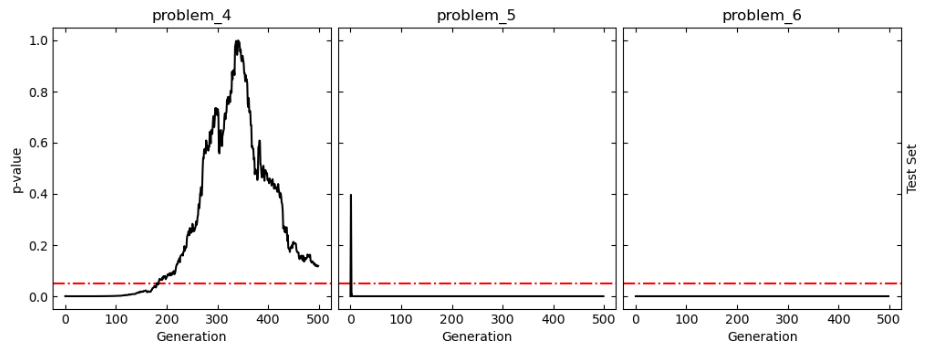


Figure 5.13: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for problem_4, problem_5 and problem_6 datasets.

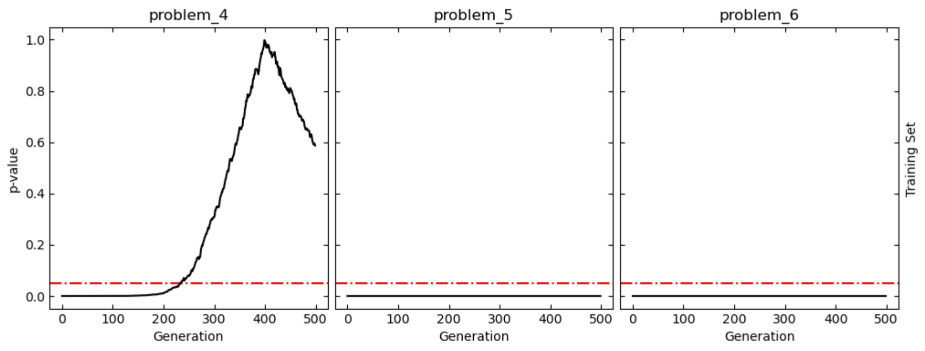


Figure 5.14: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for problem_4, problem_5 and problem_6 datasets.

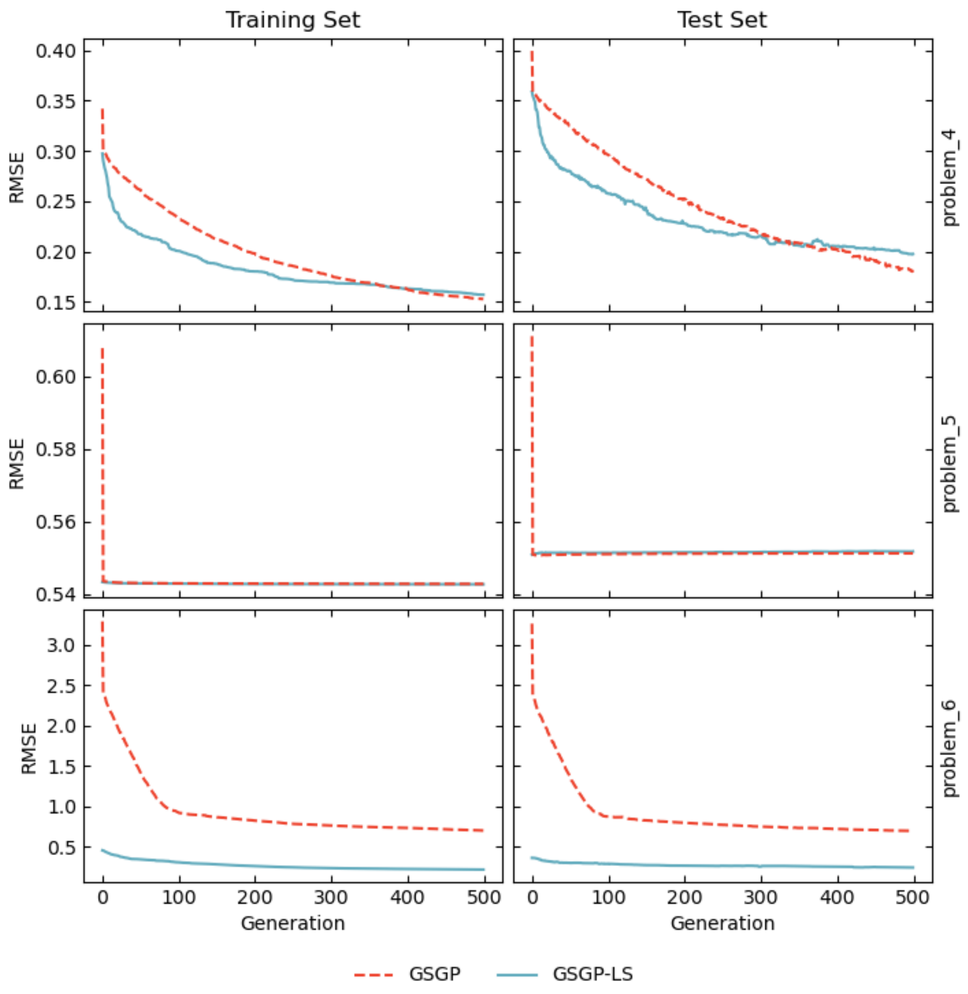


Figure 5.15: Results of the experimental comparison between GSGP and GSGP-LS with problem_4, problem_5 and problem_6 datasets.

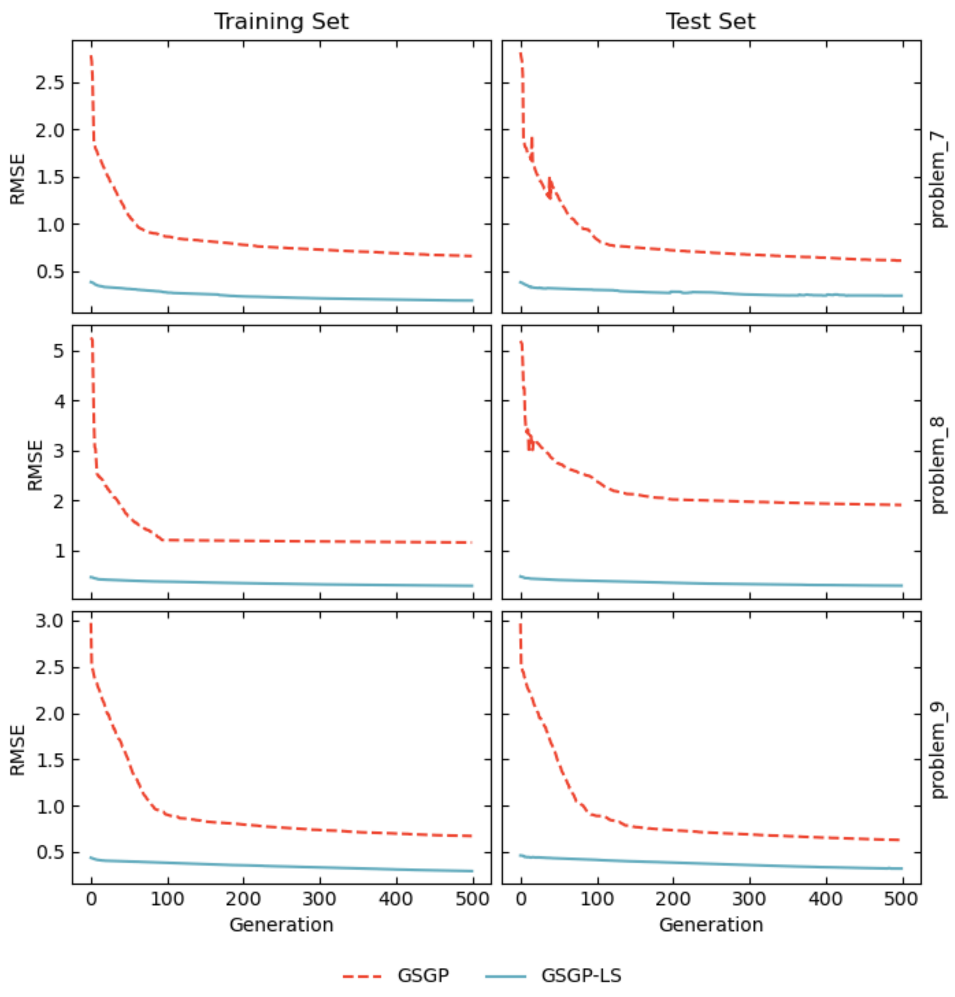


Figure 5.16: Results of the experimental comparison between GSGP and GSGP-LS with problem_7, problem_8 and problem_9 datasets.

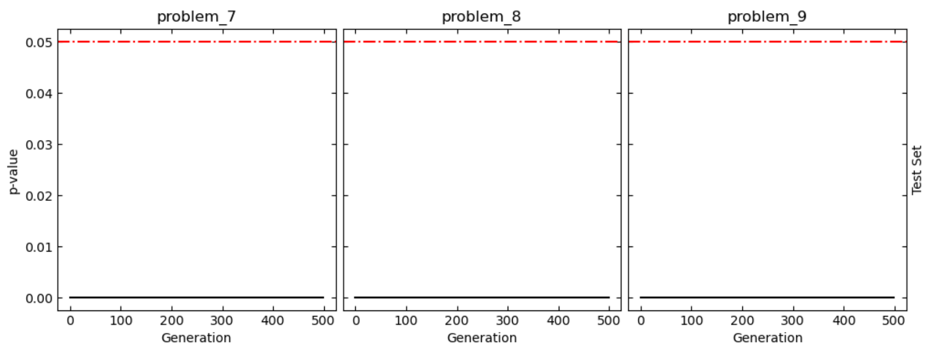


Figure 5.17: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for problem_7, problem_8 and problem_9 datasets.

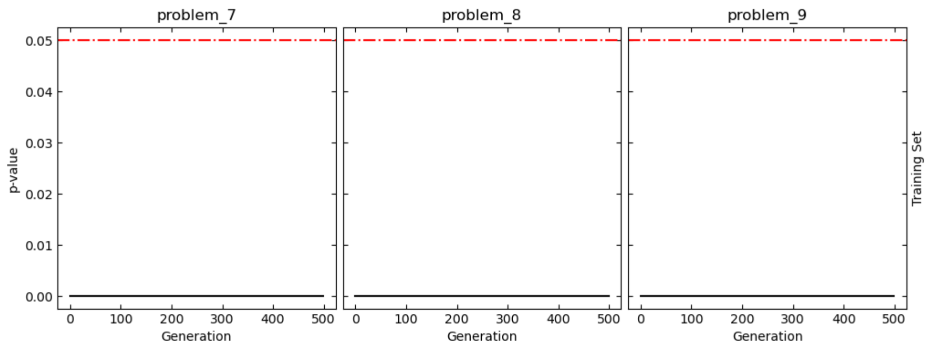


Figure 5.18: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for problem_7, problem_8 and problem_9 datasets.

5.2.3 Classification Experiments

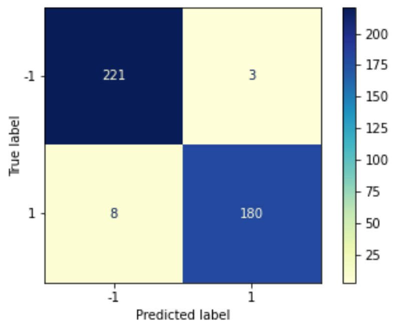
The subsequent Tables 5.11, 5.12, 5.13 presents the performance of the GSGP and GSGP-LS algorithms on the three selected classification datasets. A comprehensive classification report, based on the median values obtained from 60 independent trials, has been generated for both algorithms. The F1 score has been employed as the measure of fitness for the algorithms, with the tables providing information on precision, recall, accuracy, and support as well. For each problem and algorithm, the median classification report and the confusion matrix of the best model, which is the one with the highest F1 score, are displayed. The values in the tables are rounded to three decimal places for the purpose of clarity.

It is worth mentioning that the class labels for the targets were altered from the original and commonly used labels to -1 and 1, as a result of poor performance in the initial experiments. Further discussion on this is given in Section 5.3.

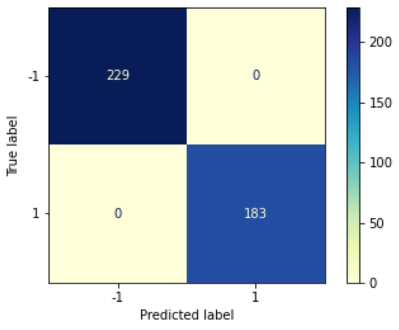
The Tables 5.11, 5.12, 5.13 provide insight into the observation that the GSGP-LS algorithm exhibits superior performance with a higher F1 score and accuracy for all the problems. The best individual of GSGP-LS predicts both classes faultlessly, as evidenced by the confusion matrix in Figure 5.19b. Furthermore, as demonstrated in Figures 5.26b, 5.21b, the total of false positive and false negative cases decreased with the implementation of GSGP-LS. Figure 5.22 demonstrates the F score per generation for all the datasets for GSGP and GSGP-LS. Not only it verifies the superiority of GSGP but also it is possible to observe that GSGP-LS algorithm reaches higher F1 scores faster. The p-values for these datasets are also demonstrated in Figures 5.24, 5.23. GSGP performs slightly better for the training set Wisconsin dataset but it can be observed that the F1 score of GSGP and GSGP-LS are both in the narrow interval of $(0.93, 0.96)$ and GSGP-LS outperforms GSGP for the test set.

Table 5.11: Median Classification Report of Banknote

GSGP	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.948	0.943	0.932	225
1	0.929	0.932	0.915	187
Accuracy				0.925
GSGP-LS	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.97	0.996	0.983	225
1	0.995	0.963	0.98	187
Accuracy				0.982



(a) GSGP

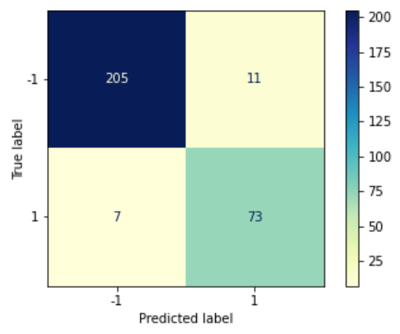


(b) GSGP-LS

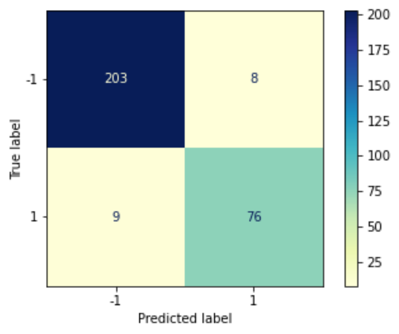
Figure 5.19: Confusion Matrices of the best individuals of Banknote Authentication

Table 5.12: Median Classification Report of Spotify Funk Songs

GSGP	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.925	0.95	0.933	204
1	0.88	0.821	0.835	92
Accuracy				0.904
GSGP-LS	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.949	0.94	0.944	213.5
1	0.845	0.867	0.856	82.5
Accuracy				0.919



(a) GSGP

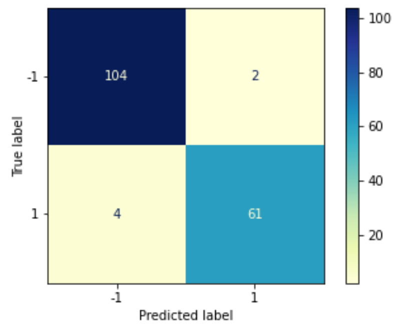


(b) GSGP-LS

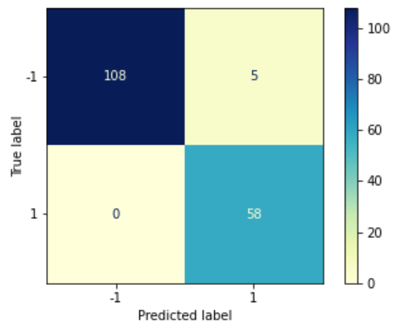
Figure 5.20: Confusion Matrices of the best individuals of Spotify Funk Songs

Table 5.13: Median Classification Report of Breast Cancer Wisconsin

GSGP	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.944	0.961	0.948	105.5
1	0.936	0.899	0.915	65.5
Accuracy				0.936
GSGP-LS	**Precision**	**Recall**	**F1 Score**	**Support**
-1	0.977	0.947	0.963	111
1	0.905	0.959	0.932	60
Accuracy				0.953



(a) GSGP



(b) GSGP-LS

Figure 5.21: Confusion Matrices of the best individuals of Breast Cancer Wisconsin

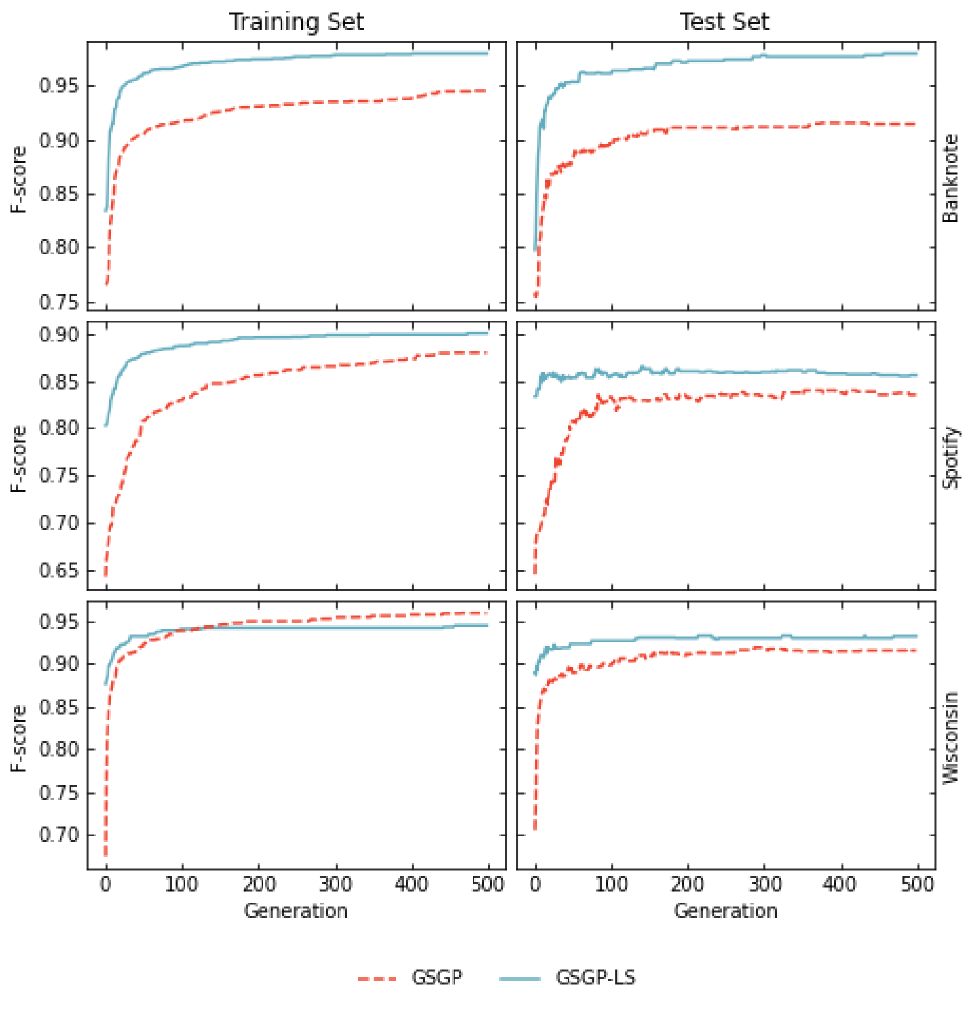


Figure 5.22: Results of the experimental comparison between GSGP and GSGP-LS with Banknote, Spotify and Wisconsin datasets.

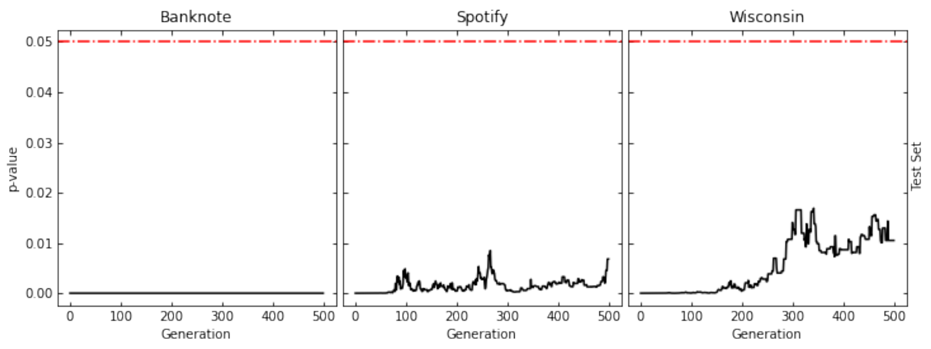


Figure 5.23: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test data for Banknote, Spotify and Wisconsin datasets.

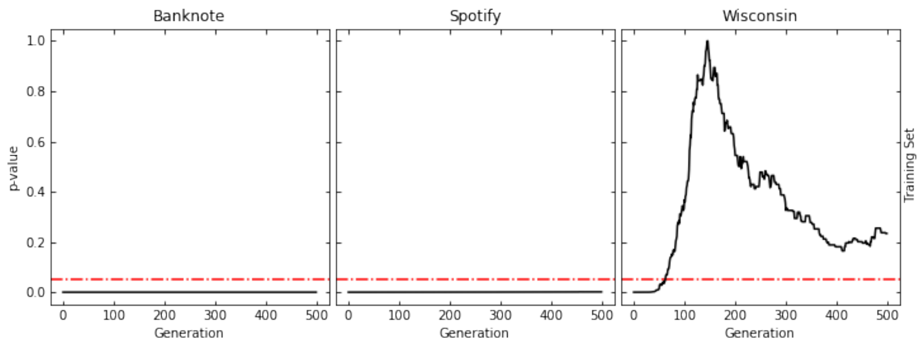


Figure 5.24: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on training data for Banknote, Spotify and Wisconsin datasets.

5.3 Discussion on Results

Even though the results obtained with GSGP-LS appear generally promising, some overfitting issues were encountered in real-life problems and theoretical benchmark problems.

For real-life problems, it can be observed that LS worsens the overfitting issues on the Bioavailability, LD50, PPB, Docking and Fludarabine datasets. Thus, it can be concluded that LS can induce or worsen overfitting on some problems on GSGP. Taking a closer look at these "problematic" datasets, it can be noticed that they share two aspects which make them particularly difficult for the regression task, and have even already caused some of them to be criticised in the literature [93]. First, they have a large amount of features compared to the amount of instances available (i.e., there are many more rows than columns in the dataset). And second, there are similar observations which map to different target values. For these reasons, these datasets are relatively prone to overfitting.

The synthetic datasets have revealed overfitting issues on problems 4 and 5. The function of the problem 4 includes exponential, sine, and cosine functions, thus it is reasonable to surmise that the LS is struggling to accurately fit the shape of the problem.

To address this issue, the experiment was repeated using a function set. To add non-linearity to the function set, the following functions were included: addition, multiplication, reciprocal, negation, square root, and sine ($\{+, *, 1/x, -x, \sqrt{x}, sin(x)\}$). RMSE plots of the problem 4 with new function set can be observed in Figure 5.25. Additionally, p-values of the test set is plotted in Figure 5.26. Examining the results, it can be concluded that it is worth investigating the impact of the function set.

Going back to the results of regression problems, additional insights can be gained on possible ways to overcome the overfitting issue. Since the test error starts with a reasonable value and only significantly increases after some generations, as per definition of overfitting, the most trivial solution would be to stop the evolution earlier. However, deciding a proper stopping condition to tackle the problem is not

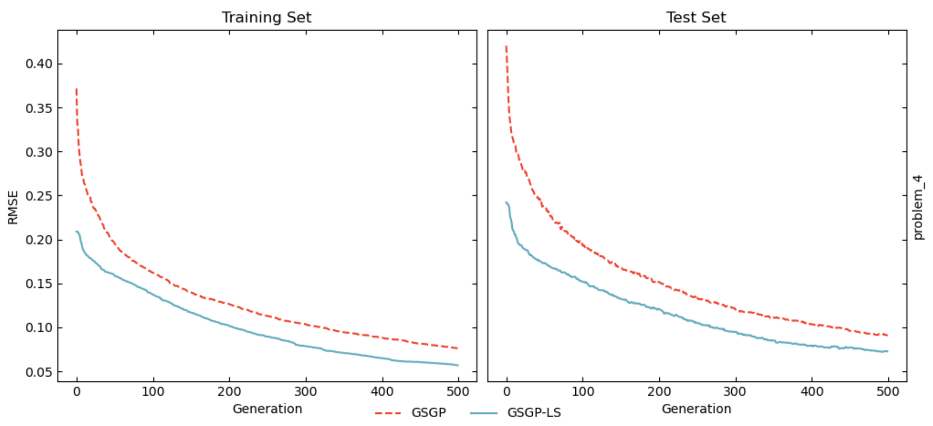
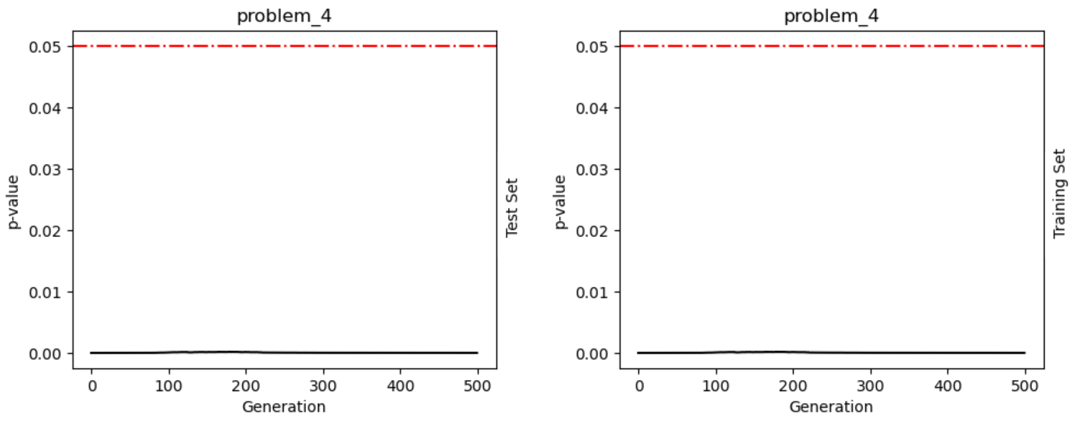


Figure 5.25: Results of the experimental comparison between GSGP and GSGP-LS with problem_4 with the alternative function set.



(a) Test Set (b) Training Set

Figure 5.26: Evolution of p-values resulting from the comparison between GSGP and GSGP-LS on test and training data for problem_4 with the alternative function set.

straightforward.

The simplest solution in that sense would be to introduce a validation set to simulate the error on unseen data along evolution, to detect immediately if the solution is starting to overfit. However, this strategy is only viable with large enough datasets, since it implies splitting the dataset in more parts, hence reducing the amount of observations in each. In fact, five datasets where we observed overfitting suffer from lack of instances, thus becoming not ideal candidates for this solution.

A more sophisticated early stopping criterion, which seems more suitable for the datasets at hand, has been proposed in [94] to leverage the semantic information available in GSGP for deciding when to end the evolutionary optimization. Notably, such a criterion would also yield the positive side effect of improving the overall computation efficiency of the process. However, both the curves of the test set fitness of GSGP-LS are increasing since the very beginning of the run on LD50, PPB and

Fludarabine. This induces a thought: although worth investigation, early stopping may not be enough to generate reliable models in those cases. For this reason, in the Chapter 6, other strategies that should be explored in the future to limit overfitting are proposed.

In the case of problem 5, a plateau in RMSE values was noted early on in the generations. This could be attributed to the division in the definition of problem 5. The division was defined in the function set as a protected division, which returned a constant value of 1 for cases where the denominator was 0. This could result in a constant behavior for the model. The overfitting in this scenario is not due to the problem itself, but rather to the choice of protected division. The algorithm approximates the division with a value that is not the actual result of the division, which causes in a deviation from the data. As a result, the algorithm discards the good results and is unable to improve its performance.

During the classification experiments, no overfitting was observed. However, an important aspect was noted regarding the use of classification and LS: the class values assigned to the targets play a crucial role in ensuring the proper functioning of the LS method. When the model used 0 and 1 as labels, it failed to correctly predict more than half of the instances of class 1, resulting in subpar performance.

To comprehend the reasoning behind this, it is imperative to recall the linear scaling equations 5.1.

$$\begin{aligned} b &= \frac{\sum_{i=1}^{n}\left[\left(t_i - \overline{t}\right)\left(P(x_i) - \overline{P}\right)\right]}{\sum_{i=1}^{n}\left(P(x_i) - \overline{P}\right)^2} \\ a &= \overline{t} - b\,\overline{P} \end{aligned} \tag{5.1}$$

where P is a GSGP individual, $P(x_i)$ is the output of the individual calculated on observation x_i, $\overline{P}$ is the average output, t is target values, $\overline{t}$ is the average target values, a and b are LS constants. After calculating a and b, individual P is evaluated by calculating the error of $a + bP$.

The problem occurs due to the utilization of the labels 0 for the targets. Since 0 is always the minority class in the datasets used in this study, the mean of the labels, $\overline{t}$, will always be slightly smaller than 0.5:

$\overline{t} + \epsilon = 0.5$, where $\epsilon > 0$

This leads to a situation where the subtraction of t_i and $\overline{t}$ will result in a value slightly higher than 0.5, leading to an approximation of 0 for b when the label is 1:

$t_i - \overline{t} - \epsilon = 0.5$, where $\epsilon > 0$

$b \approx 0$

as a and P_{scaled} are defined as:

$a = \bar{t} - b\overline{P}$,

$P_{scaled} = a + bP$

a decreases to a slightly lesser value than $\bar{t}$ and P_{scaled} becomes slightly smaller than a:

$a + \epsilon = \bar{t}$,

$P_{scaled} + \epsilon = a$, where $\epsilon > 0$

causing the most of the predictions for class 1 to be considered as 0.

In this scenario, the use of 0 as label in the equation results in an approximately constant output. The predictions are not based on the class, but instead, arise from the inherent properties of 0 as the identity element of multiplication. To resolve this issue, the class labels were altered to -1 and 1, and subsequent experimentation revealed that the root cause was the use of 0 in the equations.

6 Conclusions and Future Work

This final chapter of the dissertation presents a brief overview of the proposed work and summarizes the main conclusions related to Geometric Semantic Genetic Programming (GSGP) with Linear Scaling (LS). Additionally, suggestions for further research are put forward.

The presented technique, referred to as GSGP, is grounded in a theoretical framework that has been thoroughly elucidated. GSGP is a variation of Genetic Programming (GP) that employs two innovative genetic operators, Geometric Semantic Operators (GSO), in lieu of the conventional crossover and mutation operators. Another technique, LS, focuses on altering the fitness of solutions without modifying the genetic operators themselves. The theoretical underpinnings of both GSGP and LS are delineated in the background section, and a detailed description of the implementation of both techniques is presented in the methodology. The effects of augmenting GSGP with LS was explored in this work. In particular, encouraged by the success that was obtained in [15], the work aimed at investigating the combination of the beneficial traits of these two methods, which can both outperform standard GP, by improving the genetic operators – for GSGP – and the fitness – for LS.

The analysis involved a thorough experimental evaluation on the task of symbolic regression for six theoretical benchmarks and nine real-life problems of various difficulties besides three real-life problems from different fields for classification task. GSGP was compared against GSGP with LS (GSGP-LS), both in terms of efficiency, i.e., how fast evolution is able to achieve the desired goal, and in terms of generalization, i.e., how well the induced model is able to generalize to unseen data. The findings demonstrate substantial enhancements in most instances, both in the training and testing sets, when compared to the standard GSGP approach. Moreover, the results indicate that LS accelerates the evolutionary search process compared to GSGP. These conclusions hold for both regression and classification problems. Notably, in the classification problems, no instances of overfitting were detected, and the datasets under examination had not previously been flagged as prone to overfitting.

However, it was observed that the integration with LS makes GSGP more prone to overfitting when the addressed problem is characterized by particularly difficult data which was observed with regression problems. Nonetheless, from the behavior of LS during the evolution, it was concluded that search process could be stopped earlier to achieve comparable or better results than without LS, for both GP and GSGP, with the desirable side effect of saving computation time. For GSGP, the approach of early stopping based on semantic neighbourhood presented in [94] can be considered particularly promising. Besides early stopping, other methods have recently demonstrated their effectiveness in controlling overfitting. For instance, one may imagine to develop a system in which LS is turned on and off dynamically during the evolution. A similar approach has been recently presented for a suitable use of local search inside GSGP [95]. In that contribution, GSGP was enhanced with local search at the beginning of the run, but local search was later disabled, for an appropriate control of overfitting. A similar idea may let GSGP-LS to exploit the advantages of LS in the initial generations, and later the evolution could continue using GSGP without LS to control overfitting. Other methods that have been defined to control overfitting for GSGP and GP are the dynamic interleaving of training instances [96] and soft target regularization [97]. These methods look promising for GSGP-LS. Last but not least, the use of an explicit feature selection in a preprocessing phase [98], to integrate the implicit feature selection already performed by GP during the learning, like for instance the approach proposed in [99], should be investigated. Furthermore, future studies and experiments focusing on GSGP-LS and classification should be conducted using larger and more complex datasets.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. iii).

[2] J. R. Koza. "Genetic programming - on the programming of computers by means of natural selection". In: *Complex Adaptive Systems*. 1993 (cit. on pp. 1, 9, 11, 12, 19, 32).

[3] D. Augusto and H. Barbosa. "Symbolic regression via genetic programming". In: *Proceedings. Vol.1. Sixth Brazilian Symposium on Neural Networks*. 2000, pp. 173–178. DOI: 10.1109/SBRN.2000.889734 (cit. on p. 1).

[4] I. Icke and J. C. Bongard. "Improving genetic programming based symbolic regression using deterministic machine learning". In: *2013 IEEE Congress on Evolutionary Computation*. 2013, pp. 1763–1770. DOI: 10.1109/CEC.2013.6557774 (cit. on p. 1).

[5] M. Nicolau and J. McDermott. "Genetic Programming Symbolic Regression: What Is the Prior on the Prediction?" In: *Genetic Programming Theory and Practice XVII*. Ed. by W. Banzhaf et al. Cham: Springer International Publishing, 2020, pp. 201–225. ISBN: 978-3-030-39958-0. DOI: 10.1007/978-3-030-39958-0_11 (cit. on p. 1).

[6] M. Keijzer. "Improving Symbolic Regression with interval arithmetic and linear scaling". In: *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*. Ed. by C. Ryan et al. Vol. 2610. LNCS. Essex: Springer, Berlin, Heidelberg, New York, 2003, pp. 71–83. ISBN: 3-540-00971-X (cit. on pp. 1, 17, 18, 21, 28, 33, 40, 45).

[7] F. Archetti et al. "Genetic programming for computational pharmacokinetics in drug discovery and development". In: *Genetic Programming and Evolvable Machines* 8.4 (2007), pp. 413–432. ISSN: 1573-7632 (cit. on pp. 1, 21, 39).

[8] A. Raja et al. "Real-Time, Non-Intrusive Evaluation of VoIP". In: EuroGP'07. Valencia, Spain: Springer-Verlag, 2007, 217–228. ISBN: 9783540716020 (cit. on pp. 1, 21).

[9] M. Virgolin et al. "Symbolic Regression and Feature Construction with GP-GOMEA Applied to Radiotherapy Dose Reconstruction of Childhood Cancer Survivors". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, 1395–1402. ISBN: 9781450356183. DOI: 10.1145/3205455.3205604 (cit. on pp. 1, 22).

[10] S. Ruberto, V. Terragni, and J. H. Moore. "SGP-DT: Towards Effective Symbolic Regression with a Semantic GP Approach Based on Dynamic Targets". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, 25–26. ISBN: 9781450371278. DOI: 10.1145/3377929.3397486 (cit. on pp. 1, 22).

[11] D. Costelloe and C. Ryan. "On Improving Generalisation in Genetic Programming". In: *Genetic Programming*. Ed. by L. Vanneschi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 61–72. ISBN: 978-3-642-01181-8 (cit. on pp. 1, 2, 22).

[12] A. Moraglio, K. Krawiec, and C. Johnson. "Geometric Semantic Genetic Programming". In: vol. 7491. 2012-09, pp. 21–31. ISBN: 9783642329364. DOI: 10.1007/978-3-642-32937-1_3 (cit. on pp. 1, 2, 12–15, 19, 20, 29).

[13] M. Castelli, S. Silva, and L. Vanneschi. "A C++ framework for geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 16.1 (2015), pp. 73–81 (cit. on pp. 2, 16, 20).

[14] L. Vanneschi et al. "Geometric Semantic Genetic Programming for Real Life Applications". In: *Genetic Programming Theory and Practice XI*. Ed. by R. Riolo, J. H. Moore, and M. Kotanchek. New York, NY: Springer New York, 2014, pp. 191–209. ISBN: 978-1-4939-0375-7 (cit. on pp. 2, 32).

[15] L. Vanneschi et al. "Improving Maritime Awareness with Semantic Genetic Programming and Linear Scaling: Prediction of Vessels Position Based on AIS Data". In: *Applications of Evolutionary Computation*. Ed. by A. M. Mora and G. Squillero. Cham: Springer International Publishing, 2015, pp. 732–744. ISBN: 978-3-319-16549-3 (cit. on pp. 2, 61).

[16] A. M. Turing. "I.—Computing Machinery and Intelligence". In: *Mind* LIX.236 (1950-10), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433 (cit. on pp. 3, 13).

[17] T. M. Mitchell. *The discipline of machine learning*. Vol. 9. 2006 (cit. on p. 3).

[18] T. M. Mitchell and T. M. Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997 (cit. on p. 3).

[19] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X (cit. on pp. 4, 5).

[20] H. Simon. "Why should machine learn: An artificial intelligence approach". In: *California: Tioga-Palo* (1983) (cit. on p. 4).

[21] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020 (cit. on p. 5).

[22] J. Lever. "Classification evaluation: It is important to understand both what a classification metric expresses and what it hides". In: *Nature methods* 13.8 (2016), pp. 603–605 (cit. on pp. 5, 6).

[23] A. Botchkarev. "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006* (2018) (cit. on p. 6).

[24] T. Chai and R. R. Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature". In: *Geoscientific model development* 7.3 (2014), pp. 1247–1250 (cit. on p. 6).

[25] A. Antoniou and W.-S. Lu. *Practical optimization*. Springer, 2007 (cit. on p. 6).

[26] L. Vanneschi and S. Silva. *Lectures on Intelligent Systems*. Natural Computing Series. Springer, 2023. ISBN: 978-3-031-17921-1. DOI: 10.1007/978-3-031-17922-8. URL: https://doi.org/10.1007/978-3-031-17922-8 (cit. on pp. 6, 7, 9, 12–15).

[27] E. Pitzer and M. Affenzeller. "A comprehensive survey on fitness landscape analysis". In: *Recent advances in intelligent engineering systems* (2012), pp. 161–191 (cit. on p. 7).

[28] S. H. Zanakis and J. R. Evans. "Heuristic "Optimization": Why, When, and How to Use It". In: *Interfaces* 11.5 (1981), pp. 84–91. ISSN: 00922102, 1526551X. URL: http://www.jstor.org/stable/25060151 (visited on 2022-11-23) (cit. on p. 8).

[29] F.-S. Wang and L.-H. Chen. "Heuristic Optimization". In: *Encyclopedia of Systems Biology*. Ed. by W. Dubitzky et al. New York, NY: Springer New York, 2013, pp. 885–885. ISBN: 978-1-4419-9863-7. DOI: 10.1007/978-1-4419-9863-7_411. URL: https://doi.org/10.1007/978-1-4419-9863-7_411 (cit. on p. 8).

[30] J. H. Holland. "Genetic Algorithms and Adaptation". In: *Adaptive Control of Ill-Defined Systems*. Ed. by O. G. Selfridge, E. L. Rissland, and M. A. Arbib. Boston, MA: Springer US, 1984, pp. 317–333. ISBN: 978-1-4684-8941-5. DOI: 10.1007/978-1-4684-8941-5_21. URL: https://doi.org/10.1007/978-1-4684-8941-5_21 (cit. on p. 8).

[31] C. Darwin. *On the Origin of Species by Means of Natural Selection*. or the Preservation of Favored Races in the Struggle for Life. London: Murray, 1859 (cit. on pp. 8, 11).

[32] A. K. Kar. "Bio inspired computing – A review of algorithms and scope of applications". In: *Expert Systems with Applications* 59 (2016), pp. 20–32. ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2016.04.018`. URL: `https://www.sciencedirect.com/science/article/pii/S095741741630183X` (cit. on p. 8).

[33] J. R. Koza and R. Poli. "Genetic programming". In: *Search methodologies*. Springer, 2005, pp. 127–164 (cit. on p. 8).

[34] Leonardo Vanneschi. *Genetic Programming: Introduction, Motivations and Applications*. Lecture Notes. 2020 (cit. on pp. 9, 15).

[35] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992 (cit. on pp. 9, 12).

[36] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675 (cit. on p. 9).

[37] J. R. Koza. "Survey of genetic algorithms and genetic programming". In: *Proceedings of WESCON'95*. 1995, pp. 589–. DOI: `10.1109/WESCON.1995.485447` (cit. on p. 10).

[38] L. Rosenfeld. "Ensembled Geometric Semantic Genetic Programming: An ensemble-based initialization technique for Geometric Semantic Genetic Programming". MA thesis. Universidade Nova de Lisboa, 2021 (cit. on p. 10).

[39] N. T. Hien and N. X. Hoai. "A brief overview of population diversity measures in genetic programming". In: *Proc. 3rd Asian-Pacific Workshop on Genetic Programming, Hanoi, Vietnam*. 2006, pp. 128–139 (cit. on p. 10).

[40] S. Luke and L. Panait. "A Survey and Comparison of Tree Generation Algorithms". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. San Francisco, California: Morgan Kaufmann Publishers Inc., 2001, 81–88. ISBN: 1558607749 (cit. on pp. 11, 13).

[41] J. J. Grefenstette and J. E. Baker. "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism". In: *Proceedings of the Third International Conference on Genetic Algorithms*. George Mason University, USA: Morgan Kaufmann Publishers Inc., 1989, 20–27. ISBN: 1558600063 (cit. on p. 12).

[42] A. Brindle and U. of Alberta. Department of Computing Science. *Genetic Algorithms for Function Optimization*. University of Alberta Department of Computing Science Technical Report Tr. Thesis (Ph.D.)–University of Alberta, 1981. URL: https://books.google.pt/books?id=1VQsNAEACAAJ (cit. on p. 12).

[43] Y. Fang and J. li. "A Review of Tournament Selection in Genetic Programming". In: 2010-10, pp. 181–192. ISBN: 978-3-642-16492-7. DOI: 10.1007/978-3-642-16493-4_19 (cit. on p. 12).

[44] A. Moraglio and R. Poli. "Topological Crossover for the Permutation Representation". In: *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation*. GECCO '05. Washington, D.C.: Association for Computing Machinery, 2005, 332–338. ISBN: 9781450378000. DOI: 10.1145/1102256.1102330. URL: https://doi.org/10.1145/1102256.1102330 (cit. on pp. 13, 14).

[45] L. Vanneschi et al. "A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics". In: 2013-04, pp. 205–216. ISBN: 978-3-642-37206-3. DOI: 10.1007/978-3-642-37207-0_18 (cit. on pp. 16, 20, 32).

[46] A. Moraglio. "Towards a Geometric Unification of Evolutionary Algorithms". PhD thesis. 2007-11 (cit. on p. 19).

[47] A. Moraglio and R. Poli. "Topological Interpretation of Crossover". In: 2004-04. ISBN: 978-3-540-22344-3. DOI: 10.1007/978-3-540-24854-5_131 (cit. on p. 19).

[48] A. Moraglio and R. Poli. "Geometric crossover for the permutation representation". In: *Intelligenza Artificiale* 5 (2011-01), pp. 49–63. DOI: 10.3233/IA-2011-0004 (cit. on p. 19).

[49] A. Moraglio and R. Poli. "Topological crossover for the permutation representation". In: 2005-01, pp. 332–338. DOI: 10.1145/1102256.1102330 (cit. on p. 19).

[50] A. Moraglio and R. Poli. "Geometric Landscape of Homologous Crossover for Syntactic Trees". In: vol. 1. 2005-10, 427 –434 Vol.1. ISBN: 0-7803-9363-5. DOI: 10.1109/CEC.2005.1554715 (cit. on p. 19).

[51] A. Moraglio, R. Poli, and R. Seehuus. "Geometric Crossover for Biological Sequences". In: 2006-04, pp. 121–132. ISBN: 978-3-540-33143-8. DOI: 10.1007/11729976_11 (cit. on p. 19).

[52] L. Beadle and C. Johnson. "Semantically Driven Crossover in Genetic Programming". In: 2008-07, pp. 111 –116. ISBN: 978-1-4244-1822-0. DOI: 10.1109/CEC.2008.4630784 (cit. on pp. 19, 20).

[53] L. Beadle and C. Johnson. "Semantically Driven Mutation in Genetic Programming". In: 2009-05, pp. 1336–1342. DOI: 10.1109/CEC.2009.4983099 (cit. on p. 19).

[54] L. Beadle and C. Johnson. "Semantic analysis of program initialisation in genetic programming". In: *Genetic Programming and Evolvable Machines* 10 (2009-09), pp. 307–337. DOI: 10.1007/s10710-009-9082-5 (cit. on p. 19).

[55] D. Jackson. "Phenotypic Diversity in Initial Genetic Programming Populations". In: vol. 6021. 2010-04, pp. 98–109. ISBN: 978-3-642-12147-0. DOI: 10.1007/978-3-642-12148-7_9 (cit. on p. 19).

[56] K. Krawiec and P. Lichocki. "Approximating geometric crossover in semantic space". In: 2009-01, pp. 987–994. DOI: 10.1145/1569901.1570036 (cit. on p. 19).

[57] K. Krawiec and B. Wieloch. "Analysis of Semantic Modularity for Genetic Programming". In: *Foundations of Computing and Decision Sciences* 34 (2009-01), pp. 265–285 (cit. on p. 19).

[58] Q. U. Nguyen, N. Hoai, and M. O'Neill. "Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression". In: 2009-04, pp. 292–302. ISBN: 978-3-642-01180-1. DOI: 10.1007/978-3-642-01181-8_25 (cit. on p. 19).

[59] Q. U. Nguyen et al. "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression". In: *Genetic Programming and Evolvable Machines* 12 (2011-06), pp. 91–119. DOI: 10.1007/s10710-010-9121-2 (cit. on p. 20).

[60] M. Castelli, L. Vanneschi, and S. Silva. "Prediction of high performance concrete strength using Genetic Programming with geometric semantic genetic operators". In: *Expert Systems with Applications: An International Journal* 40 (2013-12), pp. 6856–6862. DOI: 10.1016/j.eswa.2013.06.037 (cit. on pp. 20, 35).

[61] M. Castelli, L. Vanneschi, and S. Silva. "Prediction of the Unified Parkinson's Disease Rating Scale assessment using a genetic programming system with geometric semantic genetic operators". In: *Expert Systems with Applications* 41 (2014-08), 4608–4616. DOI: 10.1016/j.eswa.2014.01.018 (cit. on pp. 20, 36).

[62] M. Castelli, L. Manzoni, and L. Vanneschi. "An Efficient Genetic Programming System with Geometric Semantic Operators and its Application to Human Oral Bioavailability Prediction". In: (2012-08) (cit. on p. 20).

[63] I. Bakurov et al. "General Purpose Optimization Library (GPOL): A Flexible and Efficient Multi-Purpose Optimization Library in Python". In: *Applied Sciences* 11 (2021-05). DOI: 10.3390/app11114774 (cit. on p. 20).

[64] H. Iba, T. Sato, and H. de Garis. "System identification approach to genetic programming". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 1994, 401–406 vol.1. DOI: 10.1109/ICEC.1994.349917 (cit. on p. 21).

[65] H. Iba and N. Nikolaev. “Genetic programming polynomial models of financial data series”. In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*. Vol. 2. 2000, 1459–1466 vol.2. DOI: 10.1109/CEC.2000.870826 (cit. on p. 21).

[66] N. Nikolaev and H. Iba. “Regularization approach to inductive genetic programming”. In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 359–375. DOI: 10.1109/4235.942530 (cit. on p. 21).

[67] H. G. Hiden. “Data-based modelling using genetic programming.” PhD thesis. University of Newcastle upon Tyne, 1998 (cit. on p. 21).

[68] M.-J. Willis et al. “Genetic programming: An introduction and survey of applications”. In: *Second international conference on genetic algorithms in engineering systems: innovations and applications*. IET. 1997, pp. 314–319 (cit. on p. 21).

[69] B. McKay et al. “Non-linear continuum regression using genetic programming”. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*. 1999, pp. 1106–1111 (cit. on p. 21).

[70] M. Keijzer. “Scaled Symbolic Regression”. In: *Genetic Programming and Evolvable Machines* 5.3 (2004), 259–269. ISSN: 1389-2576. DOI: 10.1023/B:GENP.0000030195.77571.f9 (cit. on p. 21).

[71] F. Archetti, I. Giordani, and L. Vanneschi. “Genetic programming for anticancer therapeutic response prediction using the NCI-60 dataset”. In: *Computers & Operations Research* 37 (2010-08), pp. 1395–1405. DOI: 10.1016/j.cor.2009.02.015 (cit. on p. 21).

[72] C. Pennachin, M. Looks, and J. A. de Vasconcelos. “Robust Symbolic Regression with Affine Arithmetic”. In: *Genetic and Evolutionary Computation COnference (GECCO)*. 2010 (cit. on p. 21).

[73] R. M. A. Azad and C. Ryan. “A Simple Approach to Lifetime Learning in Genetic Programming-Based Symbolic Regression”. In: *Evolutionary Computation* 22.2 (2014-06), pp. 287–317. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00111. eprint: https://direct.mit.edu/evco/article-pdf/22/2/287/1509584/evco_a_00111.pdf (cit. on p. 21).

[74] M. Virgolin, T. Alderliesten, and P. A. N. Bosman. “Linear Scaling with and within Semantic Backpropagation-Based Genetic Programming for Symbolic Regression”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, 1084–1092. ISBN: 9781450361118. DOI: 10.1145/3321707.3321758 (cit. on p. 22).

[75] S. Ruberto, V. Terragni, and J. Moore. “A semantic genetic programming framework based on dynamic targets”. In: *Genetic Programming and Evolvable Machines* 22 (2021-12), pp. 1–31. DOI: 10.1007/s10710-021-09419-3 (cit. on p. 22).

[76] S. Ruberto, V. Terragni, and J. H. Moore. “Towards Effective GP Multi-Class Classification Based on Dynamic Targets”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’21. Lille, France: Association for Computing Machinery, 2021, 812–821. ISBN: 9781450383509. DOI: 10.1145/3449639.3459324 (cit. on p. 22).

[77] D. Medernach et al. “A New Wave: A Dynamic Approach to Genetic Programming”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO ’16. Denver, Colorado, USA: Association for Computing Machinery, 2016, 757–764. ISBN: 9781450342063. DOI: 10.1145/2908812.2908857. URL: https://doi.org/10.1145/2908812.2908857 (cit. on p. 22).

[78] A. S. Sambo et al. “Feature Engineering for Improving Robustness of Crossover in Symbolic Regression”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, 249–250. ISBN: 9781450371278. DOI: 10.1145/3377929.3390078. URL: https://doi.org/10.1145/3377929.3390078 (cit. on p. 22).

[79] M. Castelli et al. “The influence of population size in geometric semantic GP”. In: *Swarm and Evolutionary Computation* 32 (2017), pp. 110–120. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2016.05.004 (cit. on p. 32).

[80] L. Vanneschi, I. Bakurov, and M. Castelli. “An initialization technique for geometric semantic GP based on demes evolution and despeciation”. In: 2017-06, pp. 113–120. DOI: 10.1109/CEC.2017.7969303 (cit. on p. 32).

[81] M. Castelli et al. “Geometric Semantic Genetic Programming with Local Search”. In: 2015-07. DOI: 10.1145/2739480.2754795 (cit. on p. 32).

[82] L. Vanneschi et al. “Alignment-based genetic programming for real life applications”. In: *Swarm and Evolutionary Computation* 44 (2018-09). DOI: 10.1016/j.swevo.2018.09.006 (cit. on p. 32).

[83] F. Archetti, I. Giordani, and L. Vanneschi. “Genetic programming for QSAR investigation of docking energy”. In: *Applied Soft Computing* 10.1 (2010), pp. 170–182. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2009.06.013. URL: https://www.sciencedirect.com/science/article/pii/S1568494609000787 (cit. on pp. 33, 39).

[84] D. Harrison and D. L. Rubinfeld. “Hedonic housing prices and the demand for clean air”. In: *Journal of Environmental Economics and Management* 5.1 (1978), pp. 81–102. ISSN: 0095-0696. DOI: https://doi.org/10.1016/0095-0696(78)90006-2 (cit. on p. 34).

[85] I.-C. Yeh. "Modeling of strength of high-performance concrete using artificial neural networks". In: *Cement and Concrete Research* 28.12 (1998), pp. 1797–1808. ISSN: 0008-8846. DOI: https://doi.org/10.1016/S0008-8846(98)00165-3 (cit. on p. 35).

[86] M. A. Little et al. "Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection". In: *BioMedical Engineering OnLine* 6.1 (2007), p. 23. ISSN: 1475-925X. DOI: 10.1186/1475-925X-6-23 (cit. on p. 36).

[87] O. Akbilgic et al. "A novel Hybrid RBF Neural Networks model as a forecaster". In: *Statistics and Computing* 24 (2013-05). DOI: 10.1007/s11222-013-9375-7 (cit. on p. 37).

[88] F. Archetti et al. "Genetic Programming for Human Oral Bioavailability of Drugs". In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO '06. Seattle, Washington, USA: Association for Computing Machinery, 2006, 255–262. ISBN: 1595931864. DOI: 10.1145/1143997.1144042. URL: https://doi.org/10.1145/1143997.1144042 (cit. on p. 38).

[89] L. Vanneschi, D. Codecasa, and G. Mauri. "A Comparative Study of Four Parallel and Distributed PSO Methods". In: *New Generation Comput.* 29 (2011-04), pp. 129–161. DOI: 10.1007/s00354-010-0102-z (cit. on p. 39).

[90] Volker Lohweg, Helene Doerksen. *Banknote Authentication Data Set*. The UCI Machine Learning Repository. Accessed: 2023-27-02. 2012. URL: https://archive.ics.uci.edu/ml/datasets/banknote+authentication (cit. on p. 41).

[91] Burak Karacayir, Berfin Sakallioglu, F. Zulal Kiraz. *Song Prediction for a Funk Band using Machine Learning Algorithms*. ML Project. 2020 (cit. on p. 42).

[92] N. Street, W. Wolberg, and O Mangasarian. "Nuclear Feature Extraction For Breast Tumor Diagnosis". In: *Proc. Soc. Photo-Opt. Inst. Eng.* 1993 (1999-01). DOI: 10.1117/12.148698 (cit. on p. 43).

[93] G. Dick, A. P. Rimoni, and P. A. Whigham. "A re-examination of the use of genetic programming on the oral bioavailability problem". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 1015–1022 (cit. on p. 57).

[94] I. Gonçalves et al. "Unsure when to stop? ask your semantic neighbors". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 929–936 (cit. on pp. 58, 62).

[95] M. Castelli et al. "Geometric Semantic Genetic Programming with Local Search". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: Association for Computing Machinery, 2015, 999–1006. ISBN: 9781450334723. DOI: 10.1145/2739480.2754795 (cit. on p. 62).

[96] I. Gonçalves and S. Silva. “Balancing Learning and Overfitting in Genetic Programming with Interleaved Sampling of Training Data”. In: *Genetic Programming*. Ed. by K. Krawiec et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 73–84. ISBN: 978-3-642-37207-0 (cit. on p. 62).

[97] L. Vanneschi and M. Castelli. “Soft target and functional complexity reduction: A hybrid regularization method for genetic programming”. In: *Expert Systems with Applications* 177 (2021), p. 114929. ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2021.114929` (cit. on p. 62).

[98] I. Guyon and A. Elisseeff. “An Introduction to Variable and Feature Selection”. In: *J. Mach. Learn. Res.* 3.null (2003), 1157–1182. ISSN: 1532-4435 (cit. on p. 62).

[99] N. M. Rodrigues et al. “SLUG: Feature Selection Using Genetic Algorithms and Genetic Programming”. In: *Genetic Programming*. Ed. by E. Medvet, G. Pappa, and B. Xue. Cham: Springer International Publishing, 2022, pp. 68–84. ISBN: 978-3-031-02056-8 (cit. on p. 62).

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon,

February 2023